7th Int'l Workshop on

Tree Adjoining Grammar

and Related Formalisms

TAG+7

20-22 May 2004

Simon Fraser University

Vancouver, Canada

Proceedings of the Workshop

TAG+7

Seventh International Workshop on

Tree Adjoining Grammar and Related Formalisms

Proceedings of the Workshop

Program Co-Chairs

Owen Rambow (Columbia University)
Matthew Stone (Rutgers University)

Local Arrangements

Chung-hye Han (Simon Fraser University)
Anoop Sarkar (Simon Fraser University)

20-22 May, 2004
Simon Fraser University
Vancouver, British Columbia, Canada

http://www.cs.rutgers.edu/TAG+7

PREFACE

We are pleased to present the current volume of technical papers accepted for presentation at the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7). And we are honored to be able to acknowledge the many people and organizations who have contributed to the event. The meeting depends on the hard work of paper authors and of our program committee, who along with outside reviewers provided feedback to authors of all submitted abstracts. We also thank our invited speakers, Barbara Partee, Jeff Pelletier and Giorgio Satta, for offering their participation and support to the meeting. As an accompaniment to the conference, Chung-hye Han and Anoop Sarkar have arranged a tutorial program on TAG, featuring Aravind Joshi, Anthony Kroch, Giorgio Satta, and Robert Frank – and we appreciate the dedication of these organizers and presenters to fostering TAG as part of a broader enterprise of formal research on language. (They even persuaded us to contribute to the tutorial session.)

Funding for the workshop has been provided by a grant from the Social Sciences and Humanities Research Council of Canada, by Simon Fraser University, and by its Department of Linguistics, School of Computer Science and Cognitive Science Program. Chung-hye Han and Anoop Sarkar are responsible not just for securing this underwriting for the meeting, but for all the details of local arrangements – a heroic task that must underpin any successful meeting. TAG+7 owes its existence to their organization and initiative.

The volume offers the 29 research papers which are to be presented at TAG+7. 14 are to be delivered in spoken presentations at the meeting and 15 are to be presented as posters. The topics of the papers cover mathematics of language, linguistic syntax and formal semantics, computational approaches to grammar design and grammatical inference, and explorations of constrained grammar formalisms in new fields such as computational biology. So, by all indications, this seventh workshop in the TAG+ series promises once again to reunite a diverse array of researchers in the cognitive science of language and in language technology, and to foster the productive interactions that have been the hallmark of TAG research.

So let's get started!

Owen Rambow and Matthew Stone
Program co-Chairs

PROGRAM COMMITTEE

Owen Rambow, Columbia University, USA (Co-chair)
Matthew Stone, Rutgers University, USA (Co-chair)
Srinivas Bangalore, ATT Labs – Research, USA
Tilman Becker, DFKI, Germany
John Chen, Columbia University, USA
Mark Dras, Macquarie University, Australia
Denys Duchier, LORIA, France
Fernanda Ferreira, Michigan State University, USA
Dan Flickinger, Stanford University, USA
Robert Frank, Johns Hopkins University, USA
Daniel Gildea, University of Rochester, USA
Jan Hajic, Charles University, Czech Republic
Caroline Heycock, University of Edinburgh, UK
Laura Kallmeyer, University of Paris 7, France
Geert-Jan Kruijff, University of the Saarland, Germany
David McDonald, Zoesis, USA
Eleni Miltsakaki, University of Pennsylvania, USA
Alexis Nasr, University of Paris 7, France
Martha Palmer, University of Pennsylvania, USA
James Pustejovsky, Brandeis University, USA
James Rogers, Earlham College, USA
Vijay Shanker, University of Delaware, USA
Giorgio Satta, University of Padova, Italy
Edward Stabler, UCLA, USA
Mark Steedman, University of Edinburgh, USA
Yuka Tateisi, Tokyo, Japan
David Weir, University of Sussex, UK
Fei Xia, IBM, USA

ADDITIONAL REVIEWERS

David Chiang, University of Pennsylvania, USA
Daniel Hardt, Copenhagen Business School, Denmark
Hong Yu, Columbia University, USA

TABLE OF CONTENTS

Spoken Presentations

Poster Presentations

LTAG Semantics of Focus*

Olga Babko-Malaya
Department of Computer and Information Science, University of Pennsylvania
3330 Walnut Street, Philadelphia, PA 19104-6389
malayao@linc.cis.upenn.edu

Abstract

This paper presents LTAG semantics of focus and focus-sensitive quantifiers which adopts alternative semantics of focus (Rooth 1985 and subsequent work). It proposes that focused lexical items make its contribution at the level of elementary trees, so that each elementary tree is associated with two semantic representations: its ordinary semantic representation and its focus representation. Based on the semantic framework, discussed in Kallmeyer and Joshi 2003 and Kallmeyer and Romero 2004, the paper develops a compositional analysis of focus representations, and extends this analysis to focusing adverbs and adverbs of quantification.

1 Alternative Semantics of Focus.

According to alternative semantics, introduced in Rooth 1985, every constituent has two semantic values: an ordinary semantic value, which determines its contribution to the truth conditions, and a focus semantic value, which determines the set of alternatives, or propositions under discussion. The focus semantic value is the set of propositions obtained by making substitutions in the position of the focused phrase. For example, the focus semantic value of the sentence **'**Mary dates $[Bill]_F$' is the set of propositions of the form "Mary dates y", whereas the focus semantic value of '$[Mary]_F$ dates Bill' is the set of propositions of the form "x dates Bill".

The contribution of focus is thus to evoke a set of alternatives, which can be contrasted with the ordinary semantic value. This can be illustrated by the question-answer paradigm. Consider, for example, the question 'Who dates Bill?'. This question determines the set of potential answers 'Mary dates Bill', 'Sue dates Bill', etc, which are alternatives to the actual answer. An appropriate answer to this question is '$[Mary]_F$ dates Bill', where the position of focus correlates with the questioned position in wh-questions. The contribution of focus in an answer is thus to indicate that propositions of the form 'x dates Bill' are alternatives to the actual answer.

Ordinary semantic value is not directly affected by focus, however, focus has a truth-conditional effect in the case of quantifiers like 'only'. Consider the sentences in (1) from Rooth 1985 in the context where John introduced Bill and Tom to Sue, and there were no other introductions. In this context, the sentence in (1a) is false, and the sentence in (1b) is true.

(1) a.John only introduced $Bill_F$ to Sue
b.John only introduced Bill to Sue_F

The analysis of 'only' proposed in Rooth 1985 assumes that 'only' is a universal quantifier which quantifies over the set of alternatives. The sentence in (2a), for example, is true in case any proposition of the form 'John introduced x to Sue' is a proposition "John introduced Bill to Sue'.

(2) a. John only introduced $Bill_F$ to Sue
b. $\forall q(q \wedge \lambda p\ [\exists y\ (p= \text{introduce}(j, y, s)](q) \rightarrow q=\text{introduce}(j, b, s))$

The main question addressed in the paper is how the set of alternatives can be computed within LTAG-based semantics. The next section introduces the semantic framework adopted in the paper, which is based on semantic feature unification. Section 3 proposes an analy-

* I would like to thank Maria-Isabel Romero, Aravind Joshi, and all participants of the XTAG meetings for discussions and numerous suggestions, as well as anonymous reviewers for their valuable comments. All remaining errors are mine.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 1-8.

sis of focus which assumes that focused lexical items make its contribution at the level of elementary trees, so that each elementary tree is associated with two semantic representations: its ordinary semantic representation and its focus representation. Sections 4 and 5 extend the proposed analysis to focus-sensitive quantifiers.

2 LTAG Semantics with Semantic Unification.

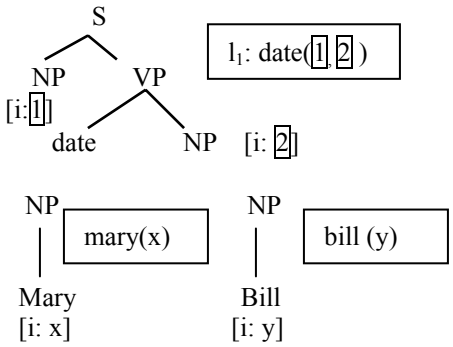
In LTAG framework (Joshi and Schabes 1997), the basic units are (elementary) trees, which can be combined into bigger trees by substitution or adjunction. LTAG derivations are represented by derivation trees that record the history of how the elementary trees are put together. Given that derivation steps in LTAG correspond to predicate-argument applications, it is usually assumed that LTAG semantics is based on the derivation tree, rather than the derived tree (Kallmeyer and Joshi 2003).

Semantic composition which we adopt is based on LTAG semantics with semantic unification (Kallmeyer and Romero 2004). In the derivation tree, elementary trees are replaced by their semantic representations and corresponding feature structures. Semantic representations are as defined in Kallmeyer and Joshi 2003, except that they do not have argument variables. These representations consist of a set of formulas (typed λ-expressions with labels) and a set of scope constraints. The scope constraints $x \leq y$ are as in Kallmeyer and Joshi 2003, except that both x and y are propositional labels or propositional variables.

Each semantic representation is linked to a feature structure. Feature structures, as illustrated by different examples below, include a feature i whose values are individual variables, features p and MaxS, whose values are propositional labels, and a feature S, whose values are situations. Semantic composition consists of feature unification. After having performed all unifications, the union of all semantic representations is built.

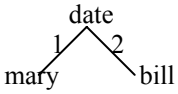
Consider, for example, the semantic representations and feature structures[2] associated with the elementary trees of the sentence shown in (3).

(3) Mary dates Bill



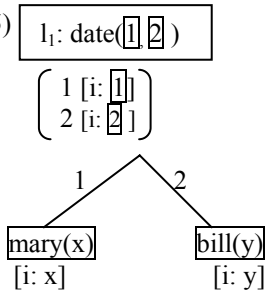
The derivation tree that records the history of how elementary trees are put together is shown in (4):

(4)



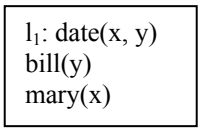
Semantic composition proceeds on the derivation tree and consists of feature unification:

(5)



Performing two unifications, 1=x, 2=y, we arrive at the final interpretation of this sentence:

(6)



This representation is interpreted conjunctively, with free variables being existentially bound.

3 LTAG based Alternative Semantics.

In order to incorporate the semantics of focus we propose that each elementary tree is associated with two semantic representations, which correspond to its ordinary semantic value and its focus semantic value. The focus semantic value is built parallel to the meaning of questions, where the focused constituent is replaced by a wh-phrase. As in the alternative semantics, ordinary

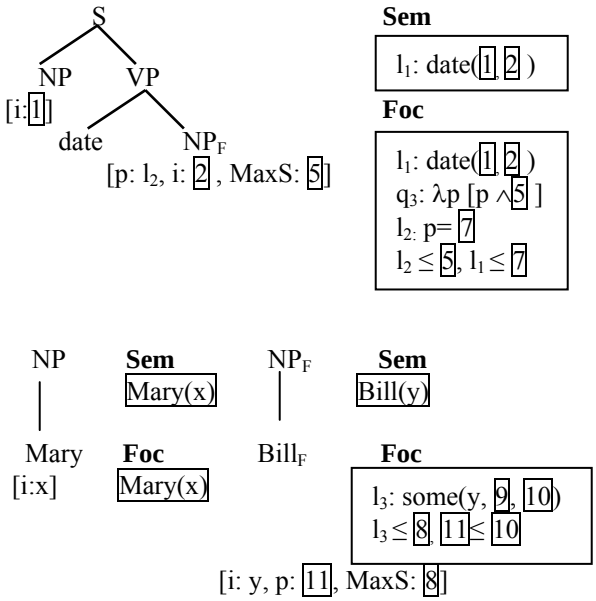
[2] For simplification, top-bottom feature distinction is omitted.

and focus semantic values are viewed as separate semantic representations.

3.1 Compositional Analysis of Focus

Semantic representations and feature structures for the sentence 'Mary dates Bill', where 'Bill' is focused is given below. As this sentence illustrates, each tree has two semantic representations: which we refer to as *Sem* and *Foc* below, and feature structures shared by the two representations.

(7) Mary dates $Bill_F$



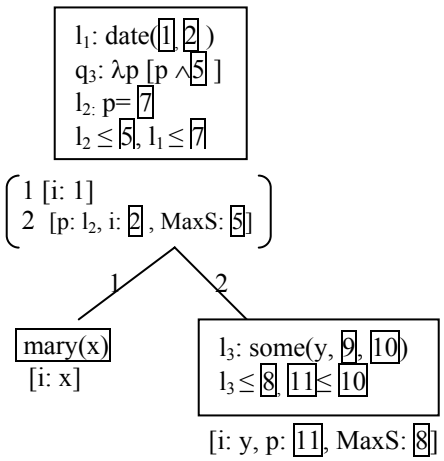
The focus representation of a non-focused phrase is simply a copy of its ordinary semantic representation, as illustrated by the NP 'Mary' above. The focus semantic value of the S tree corresponds to the semantic interpretation of a question, and is based on the LTAG-semantics of questions discussed in Romero et al 2004. And, finally, the focus representation of a focused phrase, as illustrated by the NP_F 'Bill' above, introduces an indefinite quantifier, where the restricted clause is left as an open variable.

Whereas the present analysis of the focus semantic representation assumes the semantics of questions discussed in Romero et al 2004, it differs in the following respect. Romero et al 2004 assume a multi-component analysis of wh-phrases, which is parallel to quantificational NPs discussed in Kallmeyer and Joshi 2003. Quantificational NPs under these approaches are associated with a multi-component TAG which consists of two elementary trees: S tree, which introduces the proposition containing the quantifier, and NP-tree, which introduces the restrictive clause. In the case of wh-quantifiers, the S tree, or the scope part of the wh-NP, introduces the indefinite quantifier and adjoins to the S' node in the wh-tree. As the representations in (7) show, in the case of focused constituents, both ordinary semantic and focus representations are associated with the same tree. Furthermore, the S tree headed by the verb is not a wh-tree, and does not have an S' node. And, finally, the restrictive clause of the indefinite is not provided by the syntax and is determined contextually. Given these syntactic differences, we suggest that the focus semantic value of a focused phrase is not multi-component, and the indefinite quantifier is part of the focus semantic value of the NP tree.

The use of multi-component representations for wh-phrases is largely motivated by scope constraints. As we will show in section 3.3 below, the present analysis does not present any difficulties for the analysis of scope of focused constituents, given the assumption that the scope feature which is responsible for the right scope interpretations is associated with the focused constituent (such as NP_F in (7) above).

Semantic composition of the focus representation is shown in (8):

(8)



Performing unifications leads to the following feature identities: 1=x, 2=y, 11= l_2, 8=5 . The feature MaxS, associated with the focused trees, is the scope feature, introduced in Romero et al 2004 to account for the correct maximal scope of quantificational NPs. Given these feature identities, the final representation of the focus semantic value is as follows:

(9)

q_3: λp [p∧5]	$l_2 \leq 10$
l_1: date(x, y)	$l_2 \leq 5$
l_2: p=7	$l_1 \leq 7$
Mary(x)	$l_3 \leq 5$
l_3: some(y, 9, 10)	

The scope constraints restrict possible assignments for the remaining variables. The only disambiguation (i.e. a function from propositional variables to propositional labels that respect the scope constraints in the sense of Kallmeyer and Joshi 2003) possible in this case is: l_3=5, l_1=7, l_2=10. This disambiguation leads to the desired interpretation, where the label q_3 corresponds to the set of alternatives.

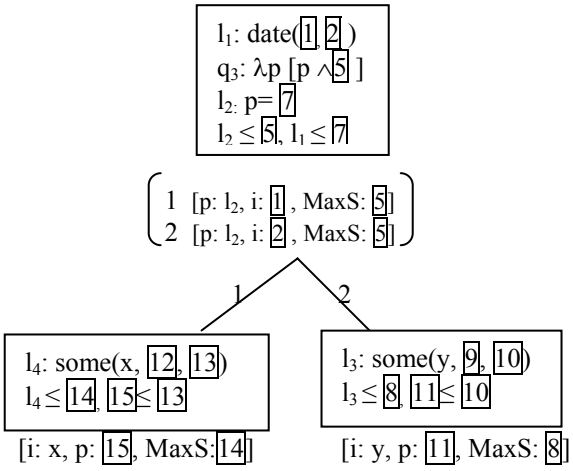
The analysis of focus presented above assumes that composition of ordinary and focus semantic representations uses the same feature structures, specifically the ones shown in (7). This means, for example, that the variable 2 is identified with y in both ordinary semantic and focus interpretations, although in the ordinary semantic representation this variable refers to Bill, and in the focus representation it is existentially bound. This does not present a problem as long as the two final representations are being viewed as separate semantic values, as the present analysis assumes.

The assumption that the same feature structures are being used in the process of composing the two representations also implies that not all variables will get values in the final representation. For example, features p and MaxS, introduced by the NP_F in (7), are only needed for the compositional interpretation of the focus semantic value, but do not play any role for the composition of the ordinary semantic representation.

3.2 Multiple Foci

Let us now consider a sentence where two constituents are focused, as in '$Mary_F$ dates $Bill_F$'. According to alternative semantics of focus, both focused phrases are replaced by existentially quantified variables in the focus semantic value, so that the set of alternatives for this sentence is of the form 'x dates y'.

(10) $Mary_F$ dates $Bill_F$



The composition of the focus semantic value of this sentence is shown in (10). Since both NPs are focused, each of them introduces an existential quantifier in the focus representation. Both NPs also include the feature MaxS in their feature structure, whose value is the propositional variable 5. The following feature identities are being performed: 1=x, 2=y, 11=l_2, 8=5, 14=5, l_2=15, so that the maximal scopes of both existential quantifiers 8 and 14 are identified with the maximal scope of the focused phrases 5. This results in the underspecified representation of scope in the final representation shown in (11):

(11)

q_3: λp [p∧5]	$l_2 \leq 10$
l_1: date(x, y)	$l_2 \leq 5$
l_2: p=7	$l_1 \leq 7$
l_4: some(x, 12, 13)	$l_4 \leq 5$
l_3: some(y, 9, 10)	$l_3 \leq 5$

The two logically equivalent interpretations which respect scope constraints are given in (12).

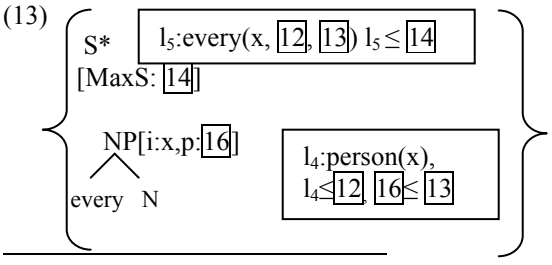
(12) λp[p∧ some(y, 9, some(x, 12, p=date(x, y))]
λp[p∧ some(x, 12, some(y, 9, p=date(x, y))]

3.3 Deriving Scope of Quantificational NPs

And, finally, let us consider a sentence with a quantificational NP, such as 'Everybody likes $Bill_F$'. The set of alternatives in this case is the set of propositions of the form 'everybody likes y', where everybody has narrow scope with respect to the indefinite quantifier. The analysis of scope of wh-phrases is discussed in detail in Romero et al 2004, where right scope interpretations are achieved by introducing MaxS features and scope constraints for quantificational and wh-phrases, which are both analyzed as multi-component TAGs. If the focused constituent is not multi-component, and the indefinite quantifier is introduced by a NP tree, as we suggested above, the question which arises is whether we can derive the desired scope interpretations.

The multi-component representation of the quantifier 'everybody' and its semantics[3] is shown in (13):

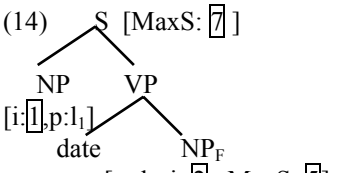
(13)



[3] Since 'everybody' is not focused, its ordinary and focus representations are the same.

The feature structures associated with the S node and non-focused NP are modified as follows, following Romero et al 2004:

(14)



Sem

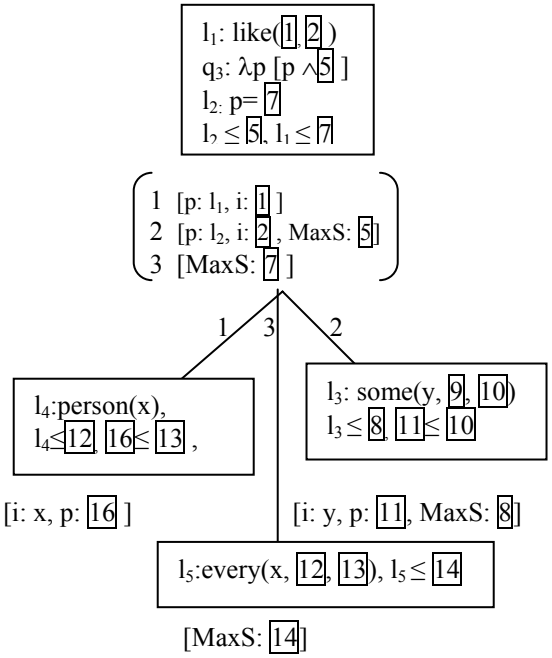
l_1: like(1, 2)

Foc

l_1: like(1, 2)
q_3: λp [p ∧5]
$l_{2:}$ p= 7
$l_2 \leq 5$, $l_1 \leq 7$

Semantic composition of the focus semantic value of the sentence 'Everybody likes $Bill_F$' is shown below. The MaxS feature of 'everybody' is introduced by the S-tree, as previous analyses assume, however, the MaxS feature of the focused phrase is introduced by the NP-tree. Given that the NP_F constituent is semantically composed with the same S tree, this modification will not change the resulting interpretation. Performing feature unifications leads to the following feature identities: 1=x, 2=y, 11=l_2, 8=5, 14=7, 16=l_1, so that the MaxS feature of the focused phrase is unified with 5, and the MaxS scope of the quantifier is unified with 7.

(15) Everybody likes $Bill_F$



The final focus representation of this sentence is shown below:

Foc

q_3: λp [p∧5]	$l_2 \leq 10$, $l_1 \leq 13$
l_1 : like(x, y)	$l_2 \leq 5$
$l_{2:}$ p=7	$l_1 \leq 7$
l_4: person(y)	$l_4 \leq 12$
l_3: some(y, 9, 10)	$l_5 \leq 7$
l_5: every(x, 12, 13)	$l_3 \leq 5$

This representation gives us the desired scope interpretation:

(16) λp[p∧some(y, 9, p=every(y, person(y) , like(x, y))]

4 Focusing Adverbs.

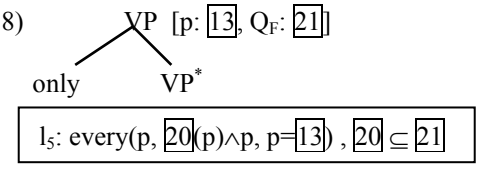
Let us now turn to sentences with focus-sensitive quantifiers, illustrated in (17a). Given the interpretation of this sentence in (17b), it is true in case any proposition of the form 'Mary dated y' is a proposition 'Mary dated Bill'.

(17) a. Mary only dated $Bill_F$
b. ∀q(q ∧ λp [∃y (p= date(m, y)](q) ->
q=date(m, b))

In Rooth 1985, compositional interpretation of sentences with focus-sensitive quantifiers proceeds in such a way that 'only' takes two arguments: the ordinary semantic value and the focus semantic value of its sister constituent. The ordinary semantic value of the sister of 'only' specifies the nuclear scope of the universal quantifier, whereas the focus semantic value specifies its restrictive clause.

The LTAG based analysis of focus-sensitive adverbs proposed in this section follows this approach in assuming that focusing adverbs like 'only' quantify over sets of alternatives determined by the focus representation. We further assume that this type of quantifiers does not introduce focus semantic values.[4] Specifically, the semantic representation and feature structures associated with the elementary tree headed by 'only' is given in (18):

(18)

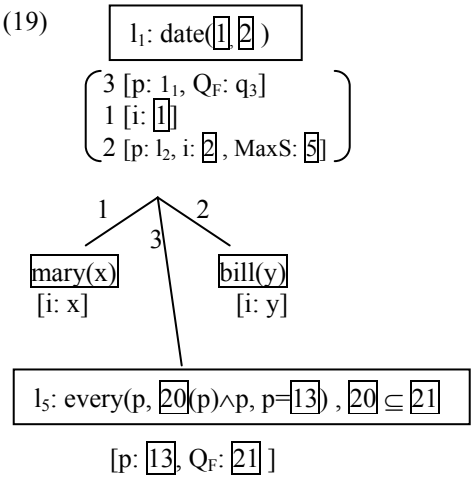


[4] This assumption does not apply to sentences with two or more focus-sensitive quantifiers, in which case we probably need two or more focus representations. The analysis of sentences of this type is left for future research.

The feature structure of the VP node in (18) introduces a new feature Q_F, which ranges over sets of propositions. Furthermore, we assume that the index 'F' on this feature indicates that its value is a label or a variable in the focus semantic representation.

The feature Q_F is also added to the VP node of the S-tree, as part of its 'top' feature structure. Composition of the ordinary semantic representation under these assumptions is shown in (19):

(19)



Performing feature unifications leads to the following identities: 1=x, 2=y, 21=q_3, 13= l_1, where the label q_3 is part of the focus representation of this sentence (compositional interpretation of the focus representation is shown in (8)).

As the final ordinary and focus semantic representations show, the variable q_3, which corresponds to the set of alternatives is shared by the two representations. This assumption contradicts our original proposal that the two representations are viewed as being completely independent of each other.

Sem:

l_1: date(x, y)
Bill(y)
Mary(x)
l_5: every(p, 20(p)∧p, p= l_1)
20 ⊆ q_3

Foc:

q_3: λp [p∧5]	$l_2 \leq 10$
l_1: date(x, y)	$l_2 \leq 5$
l_2: p=7	$l_1 \leq 7$
Mary(x)	
l_3: some(y, 9, 10)	$l_3 \leq 5$

The analysis of sentences with 'only' proposed above follows Rooth 1985 in assuming that the restrictive clause of the focus-sensitive quantifier is identified with the focus semantic value as the result of semantic composition. This approach is known as a semantic theory of focus. On the other hand, Rooth 1992, 1996, von Fintel 1994, Schwarzschild 1997 develop pragmatic theories of focus interpretation, which assume that the restrictive clause of the quantifier 'only' is a pragmatically determined variable, which can be optionally linked to the focus semantic value as the result of pragmatic factors.

The two approaches have different consequences for the present analysis of focus. Under the pragmatic approach, the restrictive clause (i.e. the variable 20 in (19)) is not identified with the label q_3 as the result of semantic composition, but is left as a free (i.e. pragmatically determined) variable. The semantic and focus interpretations can thus be viewed as being completely separate. The feature Q_F is also not needed in this case. However, this approach is problematic in view of the data discussed in the recent paper by Beaver and Clark 2003, who showed that there is a difference in the interpretation of focus in the case of sentences with 'only' and adverbs of quantification. For example, as illustrated by the data in (20)-(23), presuppositions can override the placement of focus in the interpretation of sentences with 'always', but not in the case of 'only' :

(20) Mary always managed to complete her $[\text{exams}]_F$

Whenever Mary took exams, she completed them
?Whenever Mary completed something, it was an exam

(21) Mary only managed to complete her $[\text{exams}]_F$

*What Mary did when taking exams was completing them
What Mary completed was an exam and nothing else

(22) Mary always remembers to go to $[\text{church}]_F$

Whenever it's time for church, Mary remembers to go
?Whenever Mary remembers to do something, it's always to go to church

(23) Mary only remembers to go to $[\text{church}]_F$

*The only thing Mary does when it's time to go to church, is remember to go
The only place Mary remembers to go is church.

Given these data as well as other contrasts in the behavior of 'only' and 'always', Beaver and Clark 2003 suggest that focus-sensitivity of operators like 'only' results from a grammatical mechanism, whereas quantifiers like 'always' are focus-sensitive as the result of pragmatic factors. In order to distinguish between the two types of focus-sensitivity, we proposed a semantic analysis of focus in the case of 'only', which relies on the assumption that some features allow us to relate a variable in the ordinary semantic representation with a label in the focus representation.

5 Adverbs of Quantification.

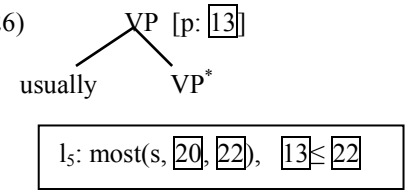
Adverbs of quantification are analyzed below as quantifiers over events or situations (Berman 1987, von Fintel 1994 among others). These quantifiers are focus-sensitive, as the examples in (24)-(25) illustrate. The sentence in (24), with 'John' being focused, has the following interpretation: 'most minimal situations in which Mary took somebody to the movies are situations where Mary took John to the movies'. The sentence in (25), on the other hand, is understood as 'most minimal situations where somebody took John to the movies are situations where Mary took John to the movies'.

(24) a. Mary usually took JOHN to the movies.
b. most(s, ∃x (take-to-the-movies(m, x, s)), take-to-the-movies(m, j, s))

(25) a. MARY usually took John to the movies
b. most(s, ∃x (take-to-the-movies(x, j, s)), take-to-the-movies(m, j, s))

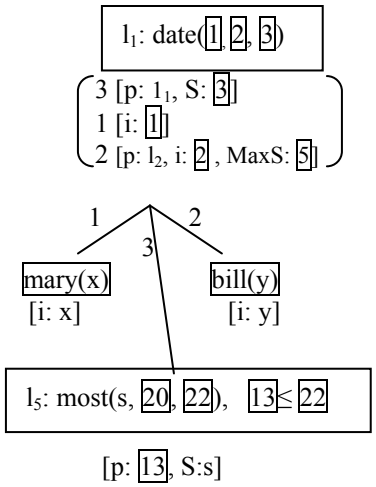
The semantic representation and feature structures of the quantifier 'usually' is given in (26). As in the case of 'only', focus-sensitive adverbs do not have a focus semantic value. Unlike 'only', the restrictive clause of the quantifier is left as a free variable:

(26)



Composition of the ordinary semantic representation of the sentence 'Mary usually dated $Bill_F$, where 'Bill' is focused, is given in (27). The semantic representations and feature structures in these representations include situation variables (see also Romero et al 2004).

(27) Mary usually dated $Bill_F$



Performing feature unifications leads to the following identities: 1=x, 2=y, 13=l_1 and 3=s, and results in the following final interpretation:

Sem:

l_1: date(x, y, s)
Bill(y)
Mary(x)
l_5: most(s, 20, 22)
$l_1 \leq 22$

The propositional variable 20, which corresponds to the restrictive clause of the quantifier is left as a free, i.e. pragmatically determined, variable.

6 Conclusion

This paper proposed an analysis of focus which assumes alternative semantics proposed in Rooth 1985 and LTAG semantic unification framework, developed in Kallmeyer and Joshi 2003 and Kallmeyer and Romero 2004. The analysis of focus presented in the paper assumes that each elementary tree is associated with two semantic representations: its ordinary semantic representation and its focus representation, and that the same feature structures are being used for the compositional interpretation of both representations. Whereas the focus representation is analyzed parallel to questions, we have proposed that focused constituents differ from the corresponding wh-phrases in that they are not analyzed as multi-component TAGs, and the existential quantifier is introduced by the NP-tree, rather than the S-tree. We further have shown that given the semantic framework with feature structures, developed in Kallmeyer and Romero 2004, this modification does not present difficulties for the analysis of scope.

The present analysis of focus has also been extended to two types of focus-sensitive quantifiers. Following Beaver and Clark 2003, it assumed that 'only' differs from adverbs of quantifications in that the restrictive clause of the quantifier is linked to the set of alternatives as the result of a grammatical mechanism. Specifically, we proposed to introduce a new feature which allows us to relate a variable in the ordinary semantic denotation with a label in the focus representation. And, finally, we suggested a possible approach to adverbs of quantification, which were analyzed as focus-sensitive quantifiers over situations.

References:

Beaver, D. and B. Clark 2003 ""Always" and "Only": Why not all Focus Sensitive Operators are Alike", *Natural Language Semantics,* 11(4), pp. 323-362

Berman, S. 1987. "Situation-Based Semantics for Adverbs of Quantification". University of Massachusetts Occasional Papers 12.

Joshi, A.K. and Schabes, Y. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages*, Springer, Berlin, pp.69-123.

Kallmeyer, L. and A.K Joshi 2003. Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation* 1(1-2), 358.

Kallmeyer, L. and M. Romero 2004 "LTAG Semantics with Semantic Unification", in *Proceedings of TAG+ 7 Workshop*, Vancouver, BC.

Romero, M., L. Kallmeyer, and O. Babko-Malaya 2004 "LTAG Semantics for Questions", in *Proceedings of TAG+7 Workshop*, Vancouver, BC.

Rooth, M. 1985. *Association with Focus.* Ph.D thesis, Univ. of Massachusetts.

Rooth, M. 1992. A theory of focus interpretation. *Natural Language Semantics,* pp.75-116.

Rooth, M. 1996. Focus in S. Lapin (ed.) *The Handbook of Contemporary Semantic Theory,* London, Basil Blackwell, pp. 271-297.

Schwarzschild, R. 1997 "Why Some Foci Must Associate", Ms. Rutgers Univ.

von Fintel, K. 1994. *Restrictions on Quantifier Domains.* Ph.D thesis. Univ. of Massachusetts.

Uses and abuses of intersected languages

David Chiang
Department of Computer and Information Science
University of Pennsylvania
200 S 33rd St
Philadelphia PA 19104 USA

Abstract

In this paper we discuss the use of *intersection* as a tool for modeling syntactic phenomena and folding of biological molecules. We argue that intersection is useful but easily overestimated, because intersection coordinates grammars via their string languages, and if strong generative capacity is given priority over weak generative capacity, this kind of coordination turns out to be rather limited. We give two example uses of intersection which overstep this limit, one using CFGs and one using a range concatenation grammar (RCG). We conclude with an analysis and example of the different kinds of parallelism available in an RCG.

1 Introduction

Context-free languages (as well as the language classes of many other formalisms) are closed under union but not under intersection (Hopcroft and Ullman, 1979), the classic example being:

$$\{\mathbf{a}^*\mathbf{b}^n\mathbf{c}^n\} \cap \{\mathbf{a}^n\mathbf{b}^n\mathbf{c}^*\} = \{\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n\} \tag{1}$$

which is easily shown by the pumping lemma to be beyond the power of CFG. Since recognizing the intersection of two CFLs takes only twice as long as recognizing a single CFL, this appears to be a way to obtain some of the power of grammar formalisms like TAG without their computational complexity. But this extra power appears less significant once we consider that strong generative capacity—the set of *structural descriptions* generated by a grammar (Chomsky, 1963)—is of primary importance for most applications of formal grammars.

Assume that a grammar G generates a set of structural descriptions $\Sigma(G)$, and for each such structural description D, a string $\overline{D}$ can be recovered, so that the string language $L(G)$ is defined as $\{\overline{D} \mid D \in \Sigma(G)\}$. Extending this definition to intersections, we define

$$\Sigma(G_1 \cap G_2) = \{D_1 \otimes D_2 \mid D_i \in \Sigma(G_i), \overline{D_1} = \overline{D_2}\} \tag{2}$$

where $\otimes$ is some operation for composing structural descriptions such that if $\overline{D_1} = \overline{D_2}$, then $\overline{D_1 \otimes D_2} = \overline{D_1} = \overline{D_2}$, otherwise $D_1 \otimes D_2$ is undefined. Note that in (2), D_1 and D_2 are correlated only by their yields; they do not directly constrain each other at all. Thus, from the point of view of strong generative capacity, language intersection is better thought of as adding a constraint to the tail end of otherwise independent parallel processes. We call this type of parallelism *weak parallelism* and argue that for real applications it is easy to overestimate how much control this kind of parallelism offers. We illustrate this in our first example, which uses CFGs for RNA pseudoknots.

We then consider the range-concatenation grammar (RCG) formalism (Boullier, 2000), which includes an intersection operation, allowing it to integrate weak parallelism more tightly into the operation of the grammar. However, weak parallelism is still susceptible to the caveat from above, which we illustrate with a second example, an analysis of German scrambling. Finally, we analyze more carefully the different kinds of parallelism available in an RCG and illustrate how they can be combined to model protein β-sheets.

2 Brown and Wilson's intersected-CFL analysis of RNA pseudoknots

Our first example comes from the RNA structure prediction literature. An RNA molecule can be thought of as a string over an alphabet of *nucleotides* or *bases* $\{\mathbf{a}, \mathbf{u}, \mathbf{c}, \mathbf{g}\}$. Certain pairs of bases, called *complementary* pairs, have an affinity for each other: **a** with **u**, **c** with **g**. This causes a molecule to fold up into a *secondary structure*, which depends on the sequence of bases. A central problem is predicting, given a sequence, what structure or structures the sequence will fold into.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 9-15.

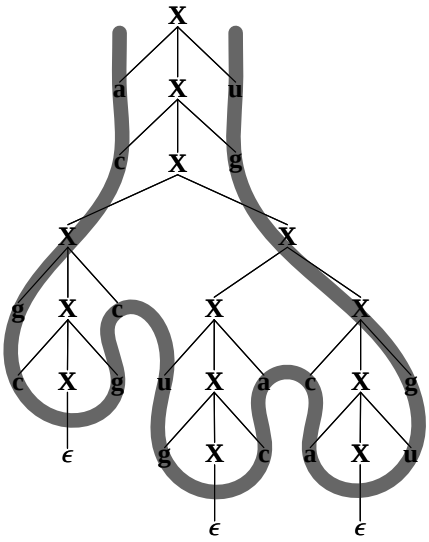


Figure 1: Example CFG derivation, with superimposed primary structure. Nonterminal symbols other than **X** are suppressed for clarity.

Searls (1992) was the first to observe similarities between this problem and syntactic analysis in natural language and to propose the use of formal grammars for biological sequence analysis. Consider the following CFG:

$$
\begin{aligned}
&\mathbf{S} \rightarrow \mathbf{Z} \\
&\mathbf{X} \rightarrow \mathbf{a}\overbrace{\mathbf{Z}\mathbf{u}} \mid \mathbf{u}\overbrace{\mathbf{Z}\mathbf{a}} \mid \mathbf{c}\overbrace{\mathbf{Z}\mathbf{g}} \mid \mathbf{g}\overbrace{\mathbf{Z}\mathbf{c}} \\
&\mathbf{Y} \rightarrow \mathbf{a}\mathbf{Y} \mid \mathbf{u}\mathbf{Y} \mid \mathbf{c}\mathbf{Y} \mid \mathbf{g}\mathbf{Y} \mid \epsilon \\
&\mathbf{Z} \rightarrow \mathbf{Y}\mathbf{X}\mathbf{Z} \mid \mathbf{Y}
\end{aligned}
\tag{3}
$$

This grammar generates the language Σ^*, but in such a way that only complementary bases are generated in the same derivation step (see Figure 1). RNA structures mostly contain only nested base pairings, like the *hairpin* (Figure 2a). For such structures, the above CFG is sufficient. However, some structures involve crossing dependencies, like the *pseudoknot* (Figure 2b), which crosses the nested base pairings of one hairpin with those of another.

There have been efforts to model pseudoknots using formalisms beyond CFG (Uemura et al., 1999; Rivas and Eddy, 2000). But Brown and Wilson (1996) attempt a different solution. They observe that $\{\mathbf{a}^m\mathbf{g}^*\mathbf{u}^m\mathbf{c}^*\}$ and $\{\mathbf{a}^*\mathbf{g}^n\mathbf{u}^*\mathbf{c}^n\}$ are context-free languages, but

$$\{\mathbf{a}^m\mathbf{g}^*\mathbf{u}^m\mathbf{c}^*\} \cap \{\mathbf{a}^*\mathbf{g}^n\mathbf{u}^*\mathbf{c}^n\} = \{\mathbf{a}^m\mathbf{g}^n\mathbf{u}^m\mathbf{c}^n\} \tag{4}$$

is beyond the power of CFG. Brown and Wilson propose to exploit this fact by interpreting the language (4) as a set of pseudoknots: the *m* **a**s and **u**s form one hairpin, and the *n* **g**s and **c**s are the other. And unlike with syntactic structures, there is an obvious way of combining the structural descriptions of the two component grammars: simply superimpose their base pairings.

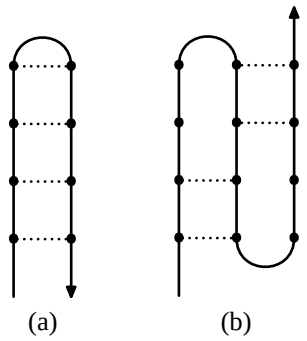


Figure 2: (a) RNA hairpin. (b) RNA pseudoknot.

However, in order for the pseudoknot to be well-formed, the two hairpins must interlock without colliding. That is, the base pairings must cross, but no two pairings should involve the same base. But the only reason the above example achieves this is because one hairpin has only **a**s and **u**s and the other has only **c**s and **g**s—that is, each symbol indicates overtly which hairpin it belongs to. For real molecules, both component grammars would have to generate at least all possible hairpins, or $\{vww^Rx\}$. In that case there would be no way of preventing the component grammars from missing each other or colliding.

Brown and Wilson recognize that there is a problem, but it is not clear whether they appreciate how serious it is. Their solution is to employ a special parsing strategy that uses the results of parsing with the first grammar to constrain the parse with the second; then the string is reparsed with the first, then again with the second. This procedure works only for their pair of grammars and only approximates the desired computation.

The root of the problem is that intersection only operates on strings, not structural descriptions. It allows parallel structural descriptions to be derived independently, then filters them on the basis of their string yields. We therefore call this kind of parallelism *weak parallelism*. The above example attempts to harness weak parallelism to generate only well-formed pseudoknots, but in order to do so it assumes that there is more information in the string languages than there really is.

3 Boullier's RCG analysis of German scrambling

Our second example comes from the range concatenation grammar (RCG) literature (Boullier, 2000). Here we briefly present a definition of a variant of RCG as a kind of deductive system. RCG clauses have the form

$$\psi :- \phi_1, \ldots, \phi_n.$$

(meaning "ψ is provable if $\phi_1, \ldots, \phi_n$ all are"). If $n = 0$, we simply write

$$\psi.$$

(which is trivially provable). The ψ and the ϕ_i in turn have the form

$$A(\alpha_1, \ldots, \alpha_m)$$

where A is a predicate (nonterminal) symbol and the α_j are strings of terminal symbols and variables (which range over strings of terminal symbols). Every α_j in ψ must be a substring of an $\alpha_{j'}$ in one of the ϕ_i. This condition ensures that in the derivation of a string w, all variables are instantiated only to substrings of w. (The standard definition of RCG does not have this requirement, because its variables range not over strings but pairs of string positions of w. The definition here is closer to that of simple literal movement grammars (Groenink, 1997).)

The language defined by an RCG is the set of all strings w such that $S(w)$ is provable, where S is a distinguished start predicate. The class of range-concatenation languages (RCLs) is exactly the set of languages recognizable in deterministic polynomial time (Bertsch and Nederhof, 2001).

Moreover, RCL, unlike CFL, is closed under intersection. The proof is very simple: given two grammars, we rename apart their predicate symbols; let S_1 and S_2 be the renamed start symbols and S be a new start symbol, and add the new clause

$$S(x) :- S_1(x), S_2(x).$$

Because the conjunction operator (comma) is part of the formalism, it can be used not only to intersect whole languages, but the yields of subderivations. This means that RCG gives finer control over weak parallelism, allowing us to localize it and use it in concert with other mechanisms in the grammar. The caveat from the previous section still applies, however, as we illustrate below.

Boullier (1999) explores possible uses of the extra power of RCG, and applies it to the phenomenon of German scrambling, in which the arguments of a verb cluster may appear in any order (see Figure 4). If we assume that an arbitrary number of verbs is allowed and arbitrary permutations of arguments is allowed, then scrambling can be shown to be beyond the power of linear context-free rewriting systems (Becker et al., 1992).

Boullier gives an RCG that he claims models German scrambling (Figure 3). The predicates **S**, **N**, and **V** use intersection to call the predicate **T** on every word, **N**′ on every noun, and **V**′ on every verb. This is an instance of *independent parallelism* (Rambow and Satta, 1999)—parallel processes in separate derivation branches—coupled with weak parallelism, which constrains the predicates to operate on the same strings.

$$\begin{aligned}
\mathbf{S}(XY) &:- \mathbf{N}(X, Y), \mathbf{V}(Y, X). \\
\mathbf{N}(nX, Y) &:- \mathbf{T}(n, X), \mathbf{N}'(n, Y), \mathbf{N}(X, Y). \\
\mathbf{N}(\epsilon, Y). & \\
\mathbf{V}(vX, Y) &:- \mathbf{T}(v, X), \mathbf{V}'(v, Y), \mathbf{V}(X, Y). \\
\mathbf{V}(\epsilon, Y). & \\
\mathbf{T}(a, bX) &:- \mathbf{T}(a, X). && a, b \in \Sigma, a \neq b \\
\mathbf{T}(a, \epsilon). & \\
\mathbf{N}'(n, vX). & && h(n) = v \\
\mathbf{N}'(n, vX) &:- \mathbf{N}'(n, X). && h(n) \neq v \\
\mathbf{V}'(v, nX). & && h(n) = v \\
\mathbf{V}'(v, nX) &:- \mathbf{V}'(v, X). && h(n) \neq v
\end{aligned}$$

Figure 3: Simplified version of Boullier's grammar. The function h maps from nouns to the verbs which take them as arguments.

These three predicates **T**, **N**′, and **V**′, in turn, check that each word is properly connected to the syntactic dependency structure. But as in Brown and Wilson's pseudoknot analysis, these three predicates rely on nonexistent information in the surface string.

First of all, **N**′ finds for each noun the verb on which it depends, and **V**′ likewise finds for each verb one of the nouns which depends on it. But whether a noun depends on a verb is assumed to be determinable (by the function h) from the noun and verb alone. In actuality, all that is known from a noun and verb is whether the noun can depend on the verb (because the verb might mark a certain case, for example), not whether it actually does. If h simply indicated the possibility of a noun depending on a verb, then this analysis' orchestration of its constraints would break down: several verbs might claim a single noun as an argument, or a verb might claim a noun which claims a different verb.

Second, the predicate **T** is used to check that all the nouns and all the verbs in each sentence are distinct, whereas in fact there is no reason why the same noun or verb would not be used twice in a single sentence. Passing over the fact that this constraint makes the generated language finite, all these constraints together indicate that the analysis assumes that dependency information is somehow overt. But this is not the case for real sentences. As in Brown and Wilson's system, this grammar tries to make weak parallelism do more than it can by assuming more information in the string language than is actually there.

(5) daß bisher noch niemand dem Kunden den Kühlschrank zu reparieren zu versuchen versprochen hat
that so far yet no one the client the refrigerator to repair to try promised has

that so far no one has promised to repair the refrigerator

(6) daß dem Kunden den Kühlschrank bisher noch niemand zu reparieren zu versuchen versprochen hat
that the client the refrigerator so far yet no one to repair to try promised has

that so far no one has promised to repair the refrigerator

Figure 4: Examples of German long-distance scrambling.

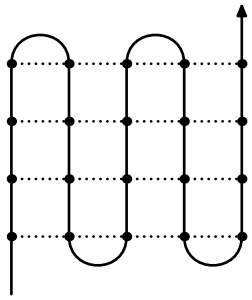


Figure 5: Protein β-sheet.

4 RCGs for protein β-sheets

What, then, can intersection be used for? Here we explore the possibility of using it for protein β-sheets. Like RNAs, proteins can be thought of as strings over an alphabet of bases, only the alphabet has 20 elements (amino acids) and the relationship between them is more complex than the complementary pairs of RNAs, and bases come together in *self-contacts* to form a folded structure. In one structural element, the β-sheet, multiple *strands* fold into a pattern like the one shown in Figure 5.

4.1 A multicomponent TAG analysis

A previous analysis (Abe and Mamitsuka, 1997) using a grammar formalism loosely related to set-local multicomponent TAG (Weir, 1988) uses *synchronous parallelism* (Rambow and Satta, 1999)—parallel processes in a single branch of a derivation—to model β-sheets. An equivalent multicomponent TAG is shown in Figure 6. This method has several strengths, which we will point out below, but two drawbacks. First, the number of strands generated is proportional to the number of components required; therefore the parsing complexity of a grammar that can generate k strands will be exponential in k. Furthermore, every grammar must impose some upper bound on the number of strands; no single grammar can generate all sheets.

A second problem is that this analysis is susceptible to a kind of spurious ambiguity in which a single structure can be derived in multiple ways. For example, consider Figure 7. In order to generate the β-sheet (a), we need trees like (b) and (c). But either of these trees can be used by itself to generate the β-sheet (d). The grammar must make room for the maximum number of strands, but when it does not use all of it, ambiguity can arise. It should be possible to carefully write the grammar to avoid much of this ambiguity, but we have not been able to eliminate all of it even for the single-component TAG case.

4.2 An RCG analysis

RCG, like many formalisms, has both synchronous parallelism (multiple arguments to a predicate) and independent parallelism (multiple predicate calls in a right-hand side). As mentioned above, it also has weak parallelism (multiple occurrences of a variable in a right-hand side), which can be coupled with either of the other two types of parallelism. We show below how these mechanisms can be used together to create an alternative model of β-sheets.

We start with some building blocks:

$$\mathbf{Anti}(a_1 X, Y a_2) \text{ :– } \mathbf{Anti}(X, Y). \qquad a_i \in \Sigma$$
$$\mathbf{Anti}(\epsilon, \epsilon).$$
$$\mathbf{Par}(a_1 X, a_2 Y) \text{ :– } \mathbf{Par}(X, Y). \qquad a_i \in \Sigma$$
$$\mathbf{Par}(\epsilon, \epsilon).$$
$$\mathbf{Adj}(X, Y) \text{ :– } \mathbf{Ant}(X, Y).$$
$$\mathbf{Adj}(X, Y) \text{ :– } \mathbf{Par}(X, Y).$$

The predicates **Anti** and **Par** generate pairs of antiparallel and parallel strands, respectively. This is an instance of synchronous parallelism, but only for pairs of strands, not all the strands together as in the multicomponent TAG analysis. Irregularities as in Figure 7a are also possible, but not shown here.

Then we can use intersection to combine them into a

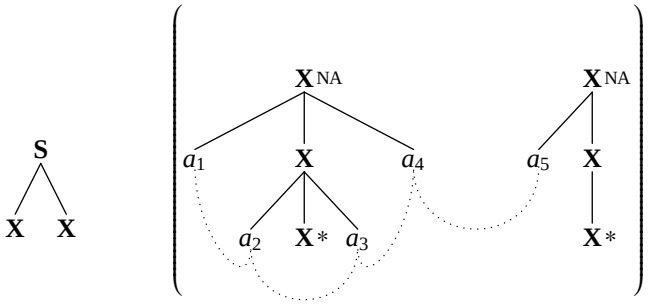


Figure 6: Set-local multicomponent TAG analysis of protein β-sheet with five alternating strands.

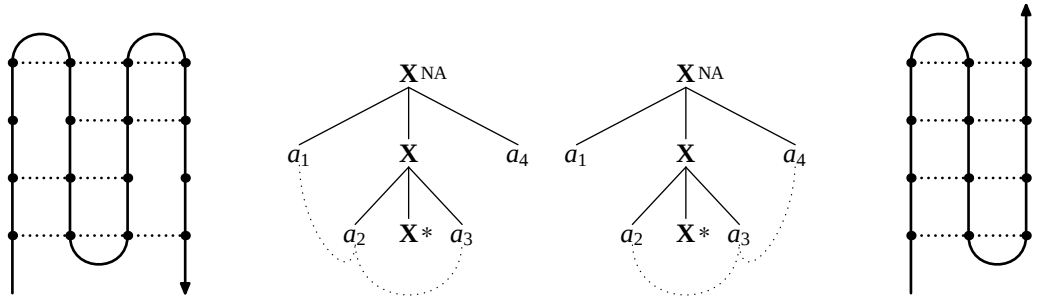


Figure 7: Illustration of spurious ambiguity.

sheet:

$$\textbf{Beta}(AB) \text{ :– } \textbf{B}(A, B).$$
$$\textbf{B}(ABY, B') \text{ :– } \textbf{B}(A, B), \textbf{Adj}(B, B').$$
$$\textbf{B}(BY, B') \text{ :– } \textbf{Adj}(B, B').$$

The first argument to **B** is a β-sheet minus the last strand, and the second argument is the last strand. The second production forms a larger β-sheet out of a smaller one by appending a new last strand and joining it to the previous last strand using **Adj**. This production has $O(n^5)$ possible instantiations (because it takes six indices to specify the variables on the left-hand side, but the arguments of **B** are always adjacent, eliminating one index), and therefore the parsing complexity of this grammar is also $O(n^5)$. Crucially, this complexity bound is not dependent on the number of strands, because each series of contacts is generated independently, not synchronously as in the multicomponent TAG analysis.

Weak parallelism is being used to ensure that each strand is consistent—that is, no two parts of the derivation that generate the same strand will disagree about the contents of the strand (including its length). Unlike with Brown and Wilson's analysis and Boullier's analysis, this is information that is really contained in the substring itself, so this is a legitimate use of weak parallelism.

Finally, even independent parallelism allows parallel subderivations to control each other via their root nonterminal (predicate) symbols, as illustrated in the following example. A β-sheet can be rolled into a cylinder to form a *β-barrel*. We can generate these as well, but we must keep track of the direction of each strand so as not to generate any Möbius strips:

$$\textbf{Barrel}(ABC) \text{ :– } \textbf{B}(A, B, C), \textbf{Par}(A, C).$$
$$\textbf{Barrel}(ABC) \text{ :– } \textbf{B}'(A, B, C), \textbf{Anti}(A, C).$$
$$\textbf{B}(A, BCY, C') \text{ :– } \textbf{B}'(A, B, C), \textbf{Anti}(C, C').$$
$$\textbf{B}(A, BCY, C') \text{ :– } \textbf{B}(A, B, C), \textbf{Par}(C, C').$$
$$\textbf{B}(A, Y, A') \text{ :– } \textbf{Par}(A, A').$$
$$\textbf{B}'(A, BCY, C') \text{ :– } \textbf{B}(A, B, C), \textbf{Anti}(C, C').$$
$$\textbf{B}'(A, BCY, C') \text{ :– } \textbf{B}'(A, B, C), \textbf{Par}(C, C').$$
$$\textbf{B}'(A, Y, A') \text{ :– } \textbf{Anti}(A, A').$$

Here **B** has three arguments: the first strand, the middle part, and the last strand; there is an additional predicate symbol **B**′ which is the same as **B**, except that **B**′ is for sheets with antiparallel first and last strands, whereas **B** is restricted here to sheets with parallel first and last strands. The first clause joins the first and last strands to form a barrel; it uses the information in the **B** vs. **B**′ distinction to join the strands so that no Möbius strips will be generated.

4.3 The importance of synchronous parallelism

The strands of β-sheets do not always appear in linear order; they can be permuted as in Figure 8. We can model such permutations by increasing the degree of synchronous parallelism (that is, the number of arguments to **B**), and therefore increasing parsing complexity. By contrast, since multicomponent TAG already uses synchronous parallelism to generate all the strands together, it allows permutations of strands at no extra cost.

Suppose we envision a sheet being built up one strand at a time, each successive strand being added to either side of the sheet:

$$\begin{aligned}
\textbf{Beta}(ABCD) &\text{ :– } \textbf{B}(A,B,C,D).\\
\textbf{B}(ABC,D,Y,B') &\text{ :– } \textbf{B}(A,B,C,D), \textbf{Adj}(B,B').\\
\textbf{B}(A,B,CDY,B') &\text{ :– } \textbf{B}(A,B,C,D), \textbf{Adj}(D,B').\\
\textbf{B}(\epsilon,B,Y,B') &\text{ :– } \textbf{Adj}(B,B').
\end{aligned}$$

Figure 8a shows an example sheet that can be generated by this grammar but not the previous ones. In this grammar, the second and fourth arguments to **B** are the leftmost and rightmost strands (*not* respectively) in the folded structure. The second clause adds a new strand on one side, and the third clause adds a new strand on the other. Both clauses have $O(n^7)$ possible instantiations if we take into account that the four arguments to **B** will always be adjacent.

Suppose we always build up a sheet out of two smaller sheets, as in Figure 9. Figure 8b shows an example sheet that can be generated by this grammar but not the previous ones. In this grammar, the second and fourth arguments are again the leftmost and rightmost strands (*not* respectively) in the folded structure. The second and third clauses join two β-sheets together in two different ways; there are conceivably four ways to join them together, but using only these two avoids spurious ambiguity. Both clauses have $O(n^{12})$ possible instantiations if we take into account that the five arguments to **B** will always be adjacent.

Figure 8c shows the only permutation of four strands that the above grammar cannot generate. This does not seem problematic, since, at least for sheets formed out of two hairpin motifs, this permutation was not known as of 1991 to occur in nature (Branden and Tooze, 1999, p. 31).

Another way in which synchronous parallelism might be important can be seen by comparing with some results for RNA structure prediction. Akutsu (2000) has shown that structure prediction for a similar class of RNA folds, "generalized pseudoknots," is NP-hard. The proof reduces another NP-hard problem (longest common subsequence) to RNA structure prediction using the assumption that in RNA structures, a single base may not participate in multiple pairings. The RCG approach illustrated above cannot enforce this assumption (just as Brown and Wilson's could not), because weak parallelism does not allow that level of control. By contrast, the multicomponent TAG analysis could enforce it easily. But for proteins this assumption does not hold, so this is not problematic for our grammars.

However, the weights we assign to productions must also respect the independence of intersected derivations. For example, the energy of a contact between two strands must not depend on other strands or contacts with them. Another NP-hardness result for RNA structure prediction (Lyngsø and Pedersen, 2000) relies crucially on such a dependency: it assumes that the energy of a base pairing (i, j) can be affected by another base pairing $(j-1, i')$ even if i and i' come from different "strands," or by $(j', i+1)$ even if j and j' come from different "strands." We leave for future work the question of how important dependencies of this sort are for β-sheets, and whether a limited amount of synchronous parallelism would suffice to approximate them.

5 Conclusion

The fundamental difficulty with the first two applications we have examined is a confusion between weak and strong generative capacity: it is misleading to speak of the "pseudoknot language" or the "scrambling language," even as abstractions, because what really matters in these two phenomena are the structures assigned to the strings rather than the strings themselves. This danger is heightened when dealing with intersected languages because intersection provides control over strings and only indirectly over structural descriptions. We have given two examples of applications which overestimate the power of this *weak parallelism* and illustrated how to use weak parallelism in concert with synchronous and independent parallelism in an RCG.

Acknowledgements

This research was supported in part by NSF ITR grant EIA-02-05456. I would like to thank Julia Hockenmaier, Laura Kallmeyer, Aravind Joshi, and the anonymous reviewers for their valuable help. *S. D. G.*

References

Naoki Abe and Hiroshi Mamitsuka. 1997. Predicting protein secondary structures based on stochastic tree grammars. *Machine Learning*, 29:275–301.

Tatuya Akutsu. 2000. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104:45–62.

Tilman Becker, Owen Rambow, and Michael Niv. 1992. The derivational generative power of formal systems,

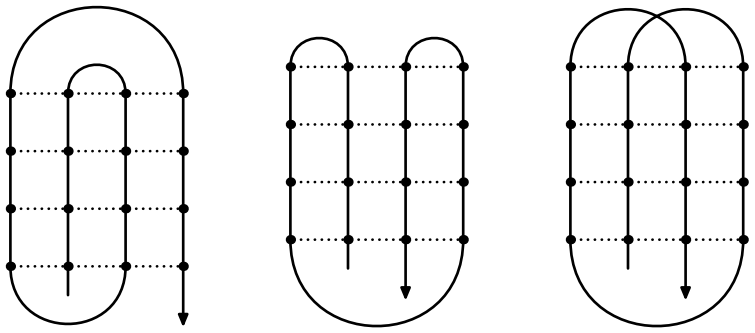


Figure 8: Permuted β-sheets.

$$\begin{aligned}
\mathbf{Beta}(ABCDE) &:\!- \mathbf{B}(A,B,C,D,E).\\
\mathbf{B}(ABC,D,EYA',B',C'D'E') &:\!- \mathbf{B}(A,B,C,D,E),\mathbf{B}(A',B',C',D',E'),\mathbf{Adj}(B,D').\\
\mathbf{B}(A,B,CDEYA',B',C',D',E') &:\!- \mathbf{B}(A,B,C,D,E),\mathbf{B}(A',B',C',D',E'),\mathbf{Adj}(D,D').\\
\mathbf{B}(\epsilon,B,C,D,\epsilon) &:\!- \mathbf{Adj}(B,D).
\end{aligned}$$

Figure 9: $O(n^{12})$-time RCG for β-sheets.

or, Scrambling is beyond LCFRS. Technical Report IRCS-92-38, Institute for Research in Cognitive Science, University of Pennsylvania. Presented at MOL3.

Eberhard Bertsch and Mark-Jan Nederhof. 2001. On the complexity of some extensions of RCG parsing. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT 2001)*, pages 66–77, Beijing.

Pierre Boullier. 1999. Chinese numbers, MIX, scrambling, and range concatenation grammars. In *Proceedings of the Ninth Conference of the European chapter of the Association for Computational Linguistics (EACL '99)*, pages 53–60.

Pierre Boullier. 2000. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 53–64, Trento.

Carl-Ivar Branden and John Tooze. 1999. *Introduction to Protein Structure*. Garland Publishing, Inc., New York. 2nd ed.

Michael Brown and Charles Wilson. 1996. RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. In *Proceedings of the Pacific Symposium on Biocomputing 1996*.

Noam Chomsky. 1963. Formal properties of grammars. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology*, volume 2, pages 323–418. Wiley, New York.

Annius Victor Groenink. 1997. *Surface without Structure: Word order and tractability issues in natural language analysis*. Ph.D. thesis, University of Utrecht.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, MA.

Rune B. Lyngsø and Christian N. S. Pedersen. 2000. RNA pseudoknot prediction in energy-based models. *J. Computational Biology*, 7(3/4):409–427.

Owen Rambow and Giorgio Satta. 1999. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223:887–120.

Elena Rivas and Sean R. Eddy. 2000. The language of RNA: a formal grammar that includes pseudoknots. *Bioinformatics*, 16(4):334–340.

David B. Searls. 1992. The linguistics of DNA. *American Scientist*, 80:579–591.

Yasuo Uemura, Aki Hasegawa, Satoshi Kobayashi, and Takashi Yokomori. 1999. Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, 210:277–303.

David J. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, Univ. of Pennsylvania.

Epsilon-Free Grammars and Lexicalized Grammars that Generate the Class of the Mildly Context-Sensitive Languages

Akio FUJIYOSHI
Department of Computer and Information Sciences, Ibaraki University
4-12-1 Nakanarusawa, Hitachi, Ibaraki, 316-8511, Japan
fujiyoshi@cis.ibaraki.ac.jp

Abstract

This study focuses on the class of string languages generated by TAGs. It examines whether the class of string languages can be generated by some epsilon-free grammars and by some lexicalized grammars. Utilizing spine grammars, this problem is solved positively. This result for spine grammars can be translated into the result for TAGs.

1 Introduction

The class of grammars called mildly context-sensitive grammars has been investigated very actively. Since it was shown that tree-adjoining grammars (TAG) (Joshi et al., 1975; Joshi and Schabes, 1996; Abeillé and Rambow, 2000), combinatory categorial grammars (CCG), linear indexed grammars (LIG), and head grammars (HG) are weakly equivalent (Vijay-Shanker and Weir, 1994), the class of string languages generated by these mildly context-sensitive grammars has been thought to be very important in the theory of formal grammars and languages. This study is strongly motivated by the work of K. Vijay-Shanker and D. J. Wier (1994) and focuses on the class of string languages generated by TAGs.

In this paper, it is examined whether the class of string languages generated by TAGs can be generated by some epsilon-free grammars and, moreover, by some lexicalized grammars. An epsilon-free grammar is a grammar with a restriction that requires no use of epsilon-rules, that is, rules defined with the empty string. Because the definitions of the four formalisms presented in the paper (Vijay-Shanker and Weir, 1994) allow the use of epsilon-rules, and all of the examples use epsilon-rules, it is natural to consider the generation problem by epsilon-free grammars. Since the notion of lexicalization is very important in the study of TAGs (Joshi and Schabes, 1996), the generation problem by lexicalized grammars is also considered.

To solve these problems, spine grammars (Fujiyoshi and Kasai, 2000) are utilized, and it is shown that for every string language generated by a TAG, not only an epsilon-free spine grammar but also a lexicalized spine grammar that genarates it can be constructed. Spine grammars are a restricted version of context-free tree grammars (CFTGs), and it was shown that spine grammars are weakly equivalent to TAGs and equivalent to linear, non-deleting, monadic CFTGs. Because considerably simple normal forms of spine grammars are known, they are useful to study the formal properties of the class of string languages generated by TAGs.

Since both TAGs and spine grammars are tree generating formalisms, they are closely related. From any epsilon-free or lexicalized spine grammar constructed in this paper, a weakly equivalent TAG is effectively obtained without utilizing epsilon-rules or breaking lexicality, the results of this paper also hold for TAGs. Though TAGs and spine grammars are weakly equivalent, the tree languages generated by TAGs are properly included in those by spine grammars. This difference occurs due to the restriction on TAGs that requires the label of the foot node to be identical to the label of the root node in an auxiliary tree. In addition, restrictions on rules of spine grammars are more lenient in some ways. Because rules of spine grammars may be non-linear, non-orderpreserving, and deleting, during derivations the copies of subtrees may be made, the order of subtrees may be changed, and subtrees may be deleted. At this point, spine grammars are different from other characterizations of TAGs (Möennich, 1997; Möennich, 1998).

2 Preliminaries

In this section, some terms, definitions and former results which will be used in the rest of this paper are introduced.

Let $\mathcal{N}$ be the set of all natural numbers, and let $\mathcal{N}_+$ be

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 16-23.

the set of all positive integers. The concatenation operator is denoted by ' $\cdot$ '. For an alphabet Σ, the set of strings over Σ is denoted by Σ^*, and the empty string is denoted by λ.

2.1 Ranked Alphabets, Trees and Substitution

A *ranked alphabet* is a finite set of symbols in which each symbol is associated with a natural number, called the *rank* of a symbol. Let Σ be a ranked alphabet. For $a \in \Sigma$, the rank of a is denoted by $r(a)$. For $n \geq 0$, it is defined that $\Sigma_n = \{a \in \Sigma \mid r(a) = n\}$.

A set D is a *tree domain* if D is a nonempty finite subset of $(\mathcal{N}_+)^*$ satisfying the following conditions:

- For any $d \in D$, if $d', d'' \in (\mathcal{N}_+)^*$ and $d = d' \cdot d''$, then $d' \in D$.
- For any $d \in D$ and $i, j \in \mathcal{N}_+$, if $i \leq j$ and $d \cdot j \in D$, then $d \cdot i \in D$.

Let D be a tree domain and $d \in D$. Elements in D are called *nodes*. A node d' is a *child of* d if there exists $i \in \mathcal{N}_+$ such that $d' = d \cdot i$. A node is called a *leaf* if it has no child. The node λ is called the *root*. A node that is neither a leaf nor the root is called an *internal node*. The *path from the root to* d is the set of nodes $\{d' \in D \mid d' \text{ is a prefix of } d\}$.

Let Σ be a ranked alphabet. A *tree* over Σ is a function $\alpha : D \to \Sigma$ where D is a tree domain. The set of trees over Σ is denoted by T_Σ. The domain of a tree α is denoted by D_α. For $d \in D_\alpha$, $\alpha(d)$ is called the *label* of d. The *subtree* of α *at* d is $\alpha/d = \{(d', a) \in (\mathcal{N}_+)^* \times \Sigma \mid (d \cdot d', a) \in \alpha\}$.

The expression of a tree over Σ is defined to be a string over elements of Σ, parentheses and commas. For $\alpha \in T_\Sigma$, if $\alpha(\lambda) = b$, $\max\{i \in \mathcal{N}_+ \mid i \in D_\alpha\} = n$ and for each $1 \leq i \leq n$, the expression of α/i is α_i, then the expression of α is $b(\alpha_1, \alpha_2, \ldots, \alpha_n)$. Note that n is the number of the children of the root. For $b \in \Sigma_0$, trees are written as b instead of $b()$.When the expression of α is $b(\alpha_1, \alpha_2, \ldots, \alpha_n)$, it is written that $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_n)$, i.e., each tree is identified with its expression.

It is defined that ε is the special symbol that may be contained in Σ_0. The *yield* of a tree is a function from T_Σ into Σ^* defined as follows. For $\alpha \in T_\Sigma$, (1) if $\alpha = a \in (\Sigma_0 - \{\varepsilon\})$, yield$(\alpha) = a$, (1') if $\alpha = \varepsilon$, yield$(\alpha) = \lambda$ and (2) if $\alpha = a(\alpha_1, \alpha_2, \ldots, \alpha_n)$ for some $a \in \Sigma_n$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \in T_\Sigma$, yield$(\alpha)$ = yield$(\alpha_1) \cdot$ yield$(\alpha_2) \cdot \cdots \cdot$ yield(α_n).

Let Σ be a ranked alphabet, and let I be a set that is disjoint from Σ. $T_\Sigma(I)$ is defined to be $T_{\Sigma \cup I}$ where $\Sigma \cup I$ is the ranked alphabet obtained from Σ by adding all elements in I as symbols of rank 0.

Let $X = \{x_1, x_2, \ldots\}$ be the fixed countable set of variables. It is defined that $X_0 = \emptyset$ and for $n \geq 1$, $X_n = \{x_1, x_2, \ldots, x_n\}$. x_1 is situationally denoted by x.

Let $\alpha, \beta \in T_\Sigma$ and $d \in D_\alpha$. It is defined that $\alpha\langle d \leftarrow \beta\rangle = \{(d', a) \mid (d', a) \in \alpha$ and d is not a prefix of $d'\} \cup \{(d \cdot d'', b) \mid (d'', b) \in \beta\}$, i.e., the tree $\alpha\langle d \leftarrow \beta\rangle$ is the result of replacing α/d by β.

Let $\alpha \in T_\Sigma(X_n)$, and let $\beta_1, \beta_2, \ldots, \beta_n \in T_\Sigma(X)$. The notion of substitution is defined. The result of substituting each β_i for nodes labeled by variable x_i in α, denoted by $\alpha[\beta_1, \beta_2, \ldots, \beta_n]$, is defined as follows.

- If $\alpha = a \in \Sigma_0$, then $a[\beta_1, \beta_2, \ldots, \beta_n] = a$.
- If $\alpha = x_i \in X_n$, then $x_i[\beta_1, \beta_2, \ldots, \beta_n] = \beta_i$.
- If $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_k)$, $b \in \Sigma_k$ and $k \geq 1$, then $\alpha[\beta_1, \beta_2, \ldots, \beta_n] =$ $b(\alpha_1[\beta_1, \beta_2, \ldots, \beta_n], \ \ldots \ , \alpha_k[\beta_1, \beta_2, \ldots, \beta_n])$.

2.2 Context-Free Tree Grammars

The context-free tree grammars (CFTGs) were introduced by W. C. Rounds (1970) as tree generating systems. The definition of CFTGs is a direct generalization of context-free grammars (CFGs).

Definition 2.1 A *context-free tree grammar* (CFTG) is a four-tuple $\mathcal{G} = (N, \Sigma, P, S)$, where:

- N and Σ are disjoint ranked alphabets of *nonterminals* and *terminals*, respectively.
- P is a finite set of *rules* of the form

  $$A(x_1, x_2, \ldots, x_n) \to \alpha$$

  with $n \geq 0$, $A \in N_n$, and $\alpha \in T_{N \cup \Sigma}(X_n)$. For $A \in N_0$, rules are written as $A \to \alpha$ instead of $A() \to \alpha$.
- S, the *initial nonterminal*, is a distinguished symbol in N_0.

For a CFTG $\mathcal{G}$, the *one-step derivation* $\underset{\mathcal{G}}{\Rightarrow}$ is the relation on $T_{N \cup \Sigma} \times T_{N \cup \Sigma}$ such that for a tree $\alpha \in T_{N \cup \Sigma}$ and a node $d \in D_\alpha$, if $\alpha/d = A(\alpha_1, \alpha_2, \ldots, \alpha_n)$, $A \in N_n$, $\alpha_1, \alpha_2, \ldots, \alpha_n \in T_{N \cup \Sigma}$ and $A(x_1, x_2, \ldots, x_n) \to \beta$ is in P, then $\alpha \underset{\mathcal{G}}{\Rightarrow} \alpha\langle d \leftarrow \beta[\alpha_1, \alpha_2, \ldots, \alpha_n]\rangle$.

An (n-*step*) *derivation* is a finite sequence of trees $\alpha_0, \alpha_1, \ldots, \alpha_n \in T_{N \cup \Sigma}$ such that $n \geq 0$ and $\alpha_0 \underset{\mathcal{G}}{\Rightarrow} \alpha_1 \underset{\mathcal{G}}{\Rightarrow} \cdots \underset{\mathcal{G}}{\Rightarrow} \alpha_n$. When there exists a derivation $\alpha_0, \alpha_1, \ldots, \alpha_n$, it is writen that $\alpha_0 \overset{n}{\underset{\mathcal{G}}{\Rightarrow}} \alpha_n$ or $\alpha_0 \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha_n$.

The *tree language generated by* $\mathcal{G}$ is the set $L(\mathcal{G}) = \{\alpha \in T_\Sigma \mid S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\}$. The *string language generated by* $\mathcal{G}$ is $L_S(\mathcal{G}) = \{\text{yield}(\alpha) \mid \alpha \in L(\mathcal{G})\}$. Note that $L_S(\mathcal{G}) \subseteq (\Sigma_0 - \{\varepsilon\})^*$.

Let $\mathcal{G}$ and $\mathcal{G}'$ be CFTGs. $\mathcal{G}$ and $\mathcal{G}'$ are *equivalent* if $L(\mathcal{G}) = L(\mathcal{G}')$. $\mathcal{G}$ and $\mathcal{G}'$ are *weakly equivalent* if $L_S(\mathcal{G}) = L_S(\mathcal{G}')$.

2.3 Spine Grammars

Spine grammars are CFTGs with a restriction called spinal-formed. To define this restriction, each nonterminal is additionally associated with a natural number.

Definition 2.2 A *head-pointing ranked alphabet* is a ranked alphabet in which each symbol is additionally associated with a natural number, called the *head* of a symbol, and the head of a symbol is satisfying the following conditions:

- If the rank of the symbol is 0, then the head of the symbol is also 0.
- If the rank of the symbol is greater than 0, then the head of the symbol is greater than 0 and less or equal to the rank of the symbol.

Let N be a head-pointing ranked alphabet. For $A \in N$, the head of A is denoted by $h(A)$.

Definition 2.3 Let $\mathcal{G} = (N, \Sigma, P, S)$ be a CFTG where N is a head-pointing ranked alphabet. For $n \geq 1$, a rule $A(x_1, x_2, \ldots, x_n) \rightarrow \alpha$ in P is *spinal-formed* if it satisfies the following conditions:

- There is exactly one leaf in α that is labeled by $x_{h(A)}$. The path from the root to the leaf is called the *spine* of α.
- For a node $d \in D_\alpha$, if d is on the spine and $\alpha(d) = B \in N$ with $r(B) \geq 1$, then $d \cdot h(B)$ is a node on the spine.
- Every node labeled by a variable in $X_n - \{x_{h(A)}\}$ is a child of a node on the spine.

The intuition of this restriction is given as follows. Let α be the right-hand side of a spinal-formed rule, and let d be a node on the spine of α. Suppose that $\alpha/d = B(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $B \in N_n$. Suppose also that the rule $B(x_1, x_2, \ldots, x_n) \rightarrow \beta$ is applied to d. Then, the tree $\alpha\langle d \leftarrow \beta[\alpha_1, \alpha_2, \ldots, \alpha_n]\rangle$ also satisfies the conditions of the right-hand side of a spinal-formed rule, i.e., the spines of α and β are combined into the new well-formed spine. Note that every node labeled by a variable in $X_n - \{x_{h(A)}\}$ is still a child of a node on the new spine.

A CFTG $\mathcal{G} = (N, \Sigma, P, S)$ is *spinal-formed* if every rule $A(x_1, x_2, \ldots, x_n) \rightarrow \alpha$ in P with $n \geq 1$ is spinal-formed. To shorten our terminology, it is said '*spine grammars*' instead of 'spinal-formed CFTGs'.

Example 2.4 Examples of spinal-formed and non-spinal-formed rules are shown. Let $\Sigma = \{a, b, c\}$ where the ranks of a, b, c are $0, 1, 3$, respectively. Let $N = \{A, B, C, D, E\}$ where the ranks of A, B, C, D, E are $4, 2, 5, 1, 0$, respectively, and the head of A, B, C are $3, 1, 5$, respectively. Note that $r(D) = 1$ and $r(E) = 0$ imply that $h(D) = 1$ and $h(E) = 0$. See Figure 1. The rule $A(x_1, x_2, x_3, x_4) \rightarrow \alpha_1$ is spinal-formed though x_2 occurs twice and x_4 does not occur in α_1. The rule $A(x_1, x_2, x_3, x_4) \rightarrow \alpha_2$ is not spinal-formed because the third child of the node labeled by A is not on the spine despite $h(A) = 3$. The rule $A(x_1, x_2, x_3, x_4) \rightarrow \alpha_3$ is not spinal-formed because there are two nodes labeled by $x_{h(A)}$. The rule $A(x_1, x_2, x_3, x_4) \rightarrow \alpha_4$ is not spinal-formed because the node labeled by x_4 is not a child of a node on the spine.

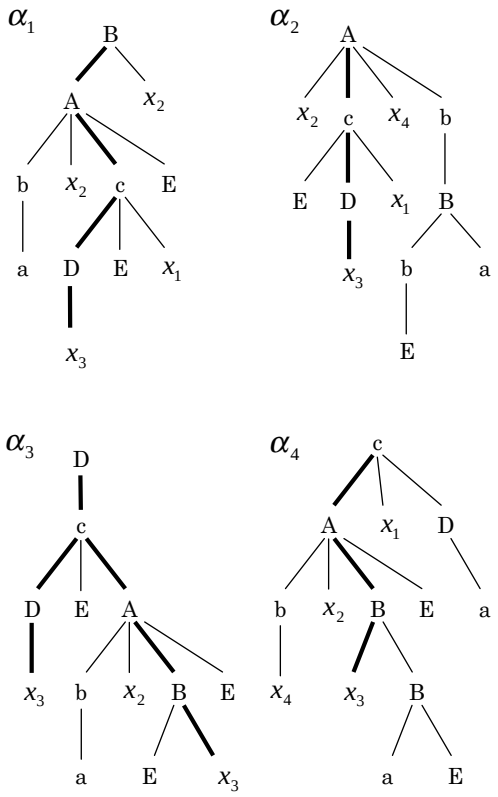


Figure 1: Right-hand sides of rules

For spine grammars, the following results are known.

Theorem 2.5 (Fujiyoshi and Kasai, 2000) *The class of string languages generated by spine grammars coincides with the class of string languages generated by TAGs.*

Theorem 2.6 (Fujiyoshi and Kasai, 2000) *For any spine grammar, there exists a equivalent spine grammar $\mathcal{G} = (N, \Sigma, P, S)$ that satisfies the following conditions:*

- *For all $A \in N$, the rank of A is either 0 or 1.*
- *For each $A \in N_0$, if $A \rightarrow \alpha$ is in P, then either $\alpha = a$ with $a \in \Sigma_0$ or $\alpha = B(C)$ with $B \in N_1$ and $C \in N_0$. See (1) and (2) in Figure 2.*
- *For each $A \in N_1$, if $A(x) \rightarrow \alpha$ is in P, then either $\alpha = B_1(B_2(\cdots(B_m(x))\cdots))$ with $m \geq 0$ and*

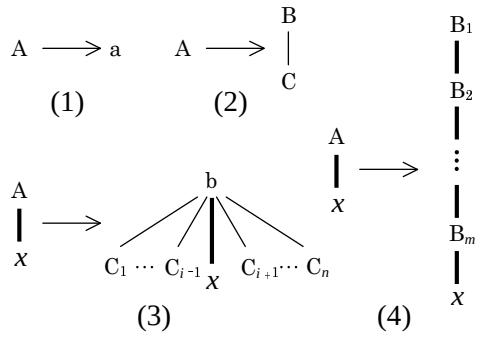


Figure 2: Rules in normal form

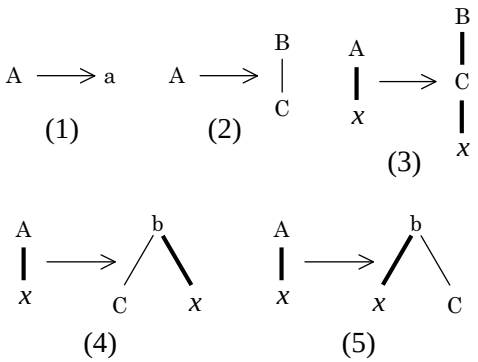


Figure 3: Rules in strong normal form

$B_1, B_2, \ldots, B_m \in N_1$ *or* $\alpha = b(C_1, C_2, \ldots, C_n)$ *with* $n \geq 1$, $b \in \Sigma_n$, *and* $C_1, C_2, \ldots, C_n$ *such that all are in* N_0 *but* $C_i = x$ *for exactly one* $i \in \{1, \ldots, n\}$. *See (3) and (4) in Figure 2.*

A spine grammar satisfies the condition of Theorem 2.6 is said to be in *normal form*. Note that for a spine grammar in normal form, the heads assigned for each nonterminal are not essential anymore because $h(A) = r(A)$ for all $A \in N$.

Theorem 2.7 (Fujiyoshi and Kasai, 2000) *For any spine grammar, there exists a weakly equivalent spine grammar* $\mathcal{G} = (N, \Sigma, P, S)$ *that satisfies the following conditions:*

- *For all* $A \in N$, *the rank of* A *is either 0 or 1.*
- *For all* $a \in \Sigma$, *the rank of* a *is either 0 or 2.*
- *For each* $A \in N_0$, *if* $A \to \alpha$ *is in* P, *then either* $\alpha = a$ *with* $a \in \Sigma_0$ *or* $\alpha = B(C)$ *with* $B \in N_1$ *and* $C \in N_0$. *See (1) and (2) in Figure 3.*
- *For each* $A \in N_1$, *if* $A(x) \to \alpha$ *is in* P, *then* α *is one of the following forms:*

  $\alpha = B(C(x))$ *with* $B, C \in N_1$,
  $\alpha = b(C, x)$ *with* $b \in \Sigma_2$ *and* $C \in N_0$, *or*
  $\alpha = b(x, C)$ *with* $b \in \Sigma_2$ *and* $C \in N_0$.

  See (3),(4), and (5) in Figure 3.

A spine grammar satisfies the condition of Theorem 2.7 is said to be in *strong normal form*.

3 The Construction of Epsilon-Free Spine Grammars

According to our definition of spine grammars, they are allowed to generate trees with leaves labeled by the special symbol ε, which is treated as the empty string while taking the yields of trees. In this section, it is shown that for any spine grammar, there exists a weakly equivalent epsilon-free spine grammar. Because any epsilon-free spine grammar cannot generate a tree with its leaves labeled by ε, it is clear that for a spine grammar with epsilon-rules, there generally doesn't exist an equivalent epsilon-free spine grammar.

Definition 3.1 A spine grammar $\mathcal{G} = (N, \Sigma, P, S)$ is *epsilon-free* if for any rule $A(x_1, x_2, \ldots, x_n) \to \alpha$ in P, α has no node labeled by the symbol ε.

Theorem 3.2 *For any spine grammar* $\mathcal{G} = (N, \Sigma, P, S)$, *if* $\lambda \notin L_S(\mathcal{G})$, *then we can construct a weakly equivalent epsilon-free spine grammar* $\mathcal{G}'$. *If* $\lambda \in L_S(\mathcal{G})$, *then we can construct a weakly equivalent spine grammar* $\mathcal{G}'$ *whose epsilon-rule is only* $S \to \varepsilon$.

Proof. Since it is enough to show the existence of a weakly equivalent grammar, without loss of generality, we may assume that $\mathcal{G}$ is in strong normal form. We may also assume that the initial nonterminal S doesn't appear in the right-hand side of any rule in P.

We first construct subsets of nonterminals E_0 and E_1 as follows. For initial values, we set $E_0 = \{A \in N_0 | A \to \varepsilon \in P\}$ and $E_1 = \emptyset$. We repeat the following operations to E_0 and E_1 until no more operations are possible:

- If $A \to B(C)$ with $B \in E_1$ and $C \in E_0$ is in P, then add $A \in N_0$ to E_0.
- If $A(x) \to b(C, x)$ with $C \in E_0$ is in P, then add $A \in N_1$ to E_1.
- If $A(x) \to b(x, C)$ with $C \in E_0$ is in P, then add $A \in N_1$ to E_1.
- If $A(x) \to B(C(x))$ with $B, C \in E_1$ is in P, then add $A \in N_1$ to E_1.

In the result, E_0 satisfies the following.

$$E_0 = \{A \in N_0 | \exists \alpha \in T_\Sigma, A \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha, \text{yield}(\alpha) = \lambda\}$$

We construct $\mathcal{G}' = (N', \Sigma', P', S)$ as follows. The set of nonterminals is $N' = N'_0 \cup N'_1$ such that $N'_0 = N_0 \cup \{\overline{A} | A \in N_1\}$ and $N'_1 = N_1$. The set of terminal

is $\Sigma' = \Sigma \cup \{c\}$, where c is a new symbol of rank 1. The set of rules P' is the smallest set satisfying following conditions:

- P' contains all rules in P except rules of the form $A \to \varepsilon$.
- If $S \in E_0$, then $S \to \varepsilon$ is in P'.
- If $A \to B(C)$ is in P and $C \in E_0$, then $A \to \overline{B}$ is in P'.
- If $A(x) \to B(C(x))$ is in P, then $\overline{A} \to B(\overline{C})$ is in P'.
- If $A(x) \to b(C, x)$ or $A(x) \to b(x, C)$ is in P and $C \in E_0$, then $A(x) \to c(x)$ is in P'.
- If $A(x) \to b(C, x)$ or $A(x) \to b(x, C)$ is in P, then $\overline{A} \to c(C)$ is in P'.

To show $L_S(\mathcal{G}') = L_S(\mathcal{G})$, we prove the following (i), (ii), and (iii) hold by induction on the length of derivations:

(i) For $A \in N_0$, $A \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$ and $\alpha' \in T_\Sigma$ if and only if $A \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha$ for some $\alpha \in T_\Sigma$ such that yield$(\alpha) =$ yield$(\alpha') \neq \lambda$.

(ii) For $A \in N_1$, $A(x) \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$ and $\alpha' \in T_\Sigma(X_1)$ if and only $A(x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha$ for some $\alpha \in T_\Sigma(X_1)$ such that yield$(\alpha) =$ yield(α').

(iii) For $\overline{A} \in N_0' - N_0$, $\overline{A} \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$ and $\alpha' \in T_\Sigma$ if and only if $A(x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha$ for some $\alpha \in T_\Sigma(X_1)$ such that yield$(\alpha[\varepsilon]) =$ yield$(\alpha') \neq \lambda$.

We start with "only if" part. For 0-step derivations, (i), (ii), and (iii) clearly hold since there doesn't exists $\alpha' \in T_\Sigma$ nor $\alpha' \in T_\Sigma(X_1)$ for each statement.

We consider the cases for 1-step derivations.

[Proof of (i)] If $A \underset{\mathcal{G}'}{\Rightarrow} \alpha'$ and $\alpha' \in T_\Sigma$, then $\alpha' = a$ for some $a \in \Sigma_0$, and the rule $A \to a$ in P' has been used. Therefore, $A \to a$ is in P, and $A \underset{\mathcal{G}}{\Rightarrow} a$.

[Proof of (ii)] If $A(x) \underset{\mathcal{G}'}{\Rightarrow} \alpha'$ and $\alpha' \in T_\Sigma(X_1)$, then $\alpha' = c(x)$, and the rule $A(x) \to c(x)$ in P' has been used. By the definition of P', $A(x) \to b(C, x)$ or $A(x) \to b(x, C)$ is in P for some $C \in E_0$. There exists $\gamma \in T_\Sigma$ such that $C \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \gamma$ and yield$(\gamma) = \lambda$. Therefore, $A(x) \underset{\mathcal{G}}{\Rightarrow} b(C, x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} b(\gamma, x)$ or $A(x) \underset{\mathcal{G}}{\Rightarrow} b(x, C) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} b(x, \gamma)$, and yield$(b(\gamma, x)) =$ yield$(b(x, \gamma)) =$ yield$(c(x))$.

[Proof of (iii)] There doesn't exists $\alpha' \in T_\Sigma$ such that $\overline{A} \underset{\mathcal{G}'}{\Rightarrow} \alpha'$.

For $k \geq 2$, assume that (i), (ii), and (iii) holds for any derivation of length less than k.

[Proof of (i)] If $A \overset{k}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$, then the rule used at the first step is one of the follwoing form: (1) $A \to B(C)$ or (2) $A \to \overline{B}$. In the case (1), $A \underset{\mathcal{G}'}{\Rightarrow} B(C) \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \beta'[\gamma'] = \alpha'$ for some $\beta' \in T_\Sigma(X_1)$ and $\gamma' \in T_\Sigma$ such that $B(x) \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \beta'$ and $C \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \gamma'$. By the induction hypothesis of (ii), there exists $\beta \in T_\Sigma(X_1)$ such that $B(x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \beta$ and yield$(\beta) =$ yield(β'). By the induction hypothesis of (i), there exists $\gamma \in T_\Sigma$ such that $C \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \gamma$, and yield$(\gamma) =$ yield(γ'). By the definition of P', $A \to B(C)$ is in P. Therefore, $A \underset{\mathcal{G}}{\Rightarrow} B(C) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \beta[\gamma]$, and yield$(\beta[\gamma]) =$ yield$(\beta'[\gamma'])$. In the case (2), $A \underset{\mathcal{G}'}{\Rightarrow} \overline{B} \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$. By the definition of P', $A \to B(C)$ is in P for some $C \in E_0$. There exists $\gamma \in T_\Sigma$ such that $C \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \gamma$ and yield$(\gamma) = \lambda$. By the induction hypothesis of (iii), there exists $\beta \in T_\Sigma(X_1)$ such that $B(x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \beta$ and yield$(\beta[\varepsilon]) =$ yield(α'). Therefore, $A \underset{\mathcal{G}}{\Rightarrow} B(C) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \beta[\gamma]$, and yield$(\beta[\gamma]) =$ yield(α').

[Proof of (ii)] If $A(x) \overset{k}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$, then the rule used at the first step is one of the follwoing form: (1) $A(x) \to B(C(x))$, (2) $A(x) \to b(C, x)$ or (3) $A(x) \to b(x, C)$. Becasue these rule are in P, the proofs are direct from the induction hypothesis like the proof of the case (1) of (i).

[Proof of (iii)] If $\overline{A} \overset{k}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha'$, then the rule used at the first step is one of the follwoing form: (1) $\overline{A} \to B(\overline{C})$ or (2) $\overline{A} \to c(C)$. In the case (1), $\overline{A} \underset{\mathcal{G}'}{\Rightarrow} B(\overline{C}) \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \beta'[\gamma'] = \alpha'$ for some $\beta' \in T_\Sigma(X_1)$ and $\gamma' \in T_\Sigma$ such that $B(x) \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \beta'$ and $\overline{C} \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \gamma'$. By the induction hypothesis of (ii), there exists $\beta \in T_\Sigma(X_1)$ such that $B(x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \beta$ and yield$(\beta) =$ yield(β'). By the induction hypothesis of (iii), there exists $\gamma \in T_\Sigma(X_1)$ such that $C(x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \gamma$ and yield$(\gamma[\varepsilon]) =$ yield(γ'). By the definition of P', $A(x) \to B(C(x))$ is in P. Therefore, $A(x) \underset{\mathcal{G}}{\Rightarrow} B(C(x)) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \beta[\gamma]$, and yield$(\beta[\gamma[\varepsilon]]) =$ yield$(\beta'[\gamma'])$. In the case (2), $\overline{A} \underset{\mathcal{G}'}{\Rightarrow} c(C) \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} c(\gamma') = \alpha'$ for some $\gamma' \in T_\Sigma$ such that $C \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \gamma'$. By the induction hypothesis of (i), there exists $\gamma \in T_\Sigma$ such that $C \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \gamma$ and yield$(\gamma) =$ yield(γ'). By the definition of P', $A(x) \to b(C, x)$ or $A(x) \to b(x, C)$ is in P. Without loss of generality, we may assume that $A(x) \to b(C, x)$ is in P. Therefore, $A(x) \underset{\mathcal{G}}{\Rightarrow} b(C, x) \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} b(\gamma, x)$, and yield$(b(\gamma, x)[\varepsilon]) =$ yield$(c(\gamma'))$.

The "if" part is similarly proved as follows. For 0-step derivations, (i), (ii), and (iii) clearly hold since there doesn't exists $\alpha \in T_\Sigma$ nor $\alpha \in T_\Sigma(X_1)$ for each statement.

The cases for 1-step derivations are proved.

[Proof of (i)] If $A \underset{\mathcal{G}}{\Rightarrow} \alpha$ and $\alpha \in T_\Sigma$, then $\alpha = a$ for some $a \in \Sigma_0$, and the rule $A \to a$ in P has been used. Therefore, $A \to a$ is in P', and $A \underset{\mathcal{G}'}{\Rightarrow} a$.

[Proof of (ii) and (iii)] There doesn't exists $\alpha \in T_\Sigma$ such that $A \underset{\mathcal{G}}{\Rightarrow} \alpha$.

For $k \geq 2$, assume that (i), (ii), and (iii) holds for any derivation of length less than k.

[Proof of (i)] If $A \underset{\mathcal{G}}{\overset{k}{\Rightarrow}} \alpha$, then the rule used at the first step must be of the form $A \to B(C)$. Thus, $A \underset{\mathcal{G}}{\Rightarrow} B(C) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \beta[\gamma] = \alpha$ for some $\beta \in T_\Sigma(X_1)$ and $\gamma \in T_\Sigma$ such that $B(x) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \beta$ and $C \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \gamma$. Here, we have to think of the two cases: (1) yield$(\gamma) \neq \lambda$ and (2) yield$(\gamma) = \lambda$. In the case (1), by the induction hypothesis of (ii), there exists $\beta' \in T_\Sigma(X_1)$ such that $B(x) \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \beta'$ and yield$(\beta') =$ yield(β), and by the induction hypothesis of (i), there exists $\gamma' \in T_\Sigma$ such that $C \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \gamma'$ and yield$(\gamma') =$ yield(γ). By the definition of P', $A \to B(C)$ is in P. Therefore, $A \underset{\mathcal{G}'}{\Rightarrow} B(C) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \beta'[\gamma']$, and yield$(\beta'[\gamma']) =$ yield$(\beta[\gamma])$. In the case (2), $C \in E_0$. Thus, $A \to \overline{B}$ is in P'. By the induction hypothesis of (iii), there exists $\beta' \in T_\Sigma(X_1)$ such that $\overline{B} \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \beta'$ and yield$(\beta') =$ yield$(\beta[\varepsilon])$. Therefore, $A \underset{\mathcal{G}'}{\Rightarrow} \overline{B} \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \beta'$, and yield$(\beta') =$ yield$(\beta[\gamma])$.

[Proof of (ii)] If $A(x) \underset{\mathcal{G}}{\overset{k}{\Rightarrow}} \alpha$, then the rule used at the first step is one of the follwoing form: (1) $A(x) \to B(C(x))$, (2) $A(x) \to b(C, x)$ or (3) $A(x) \to b(x, C)$. The proof of the case (1) is direct from the induction hypothesis. In the case (2), $A(x) \underset{\mathcal{G}}{\Rightarrow} b(C, x) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} b(\gamma, x) = \alpha$ for some $\gamma \in T_\Sigma$ such that $C \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \gamma$. Here, we have to think of the two cases: (a) yield$(\gamma) \neq \lambda$ and (b) yield$(\gamma) = \lambda$. (a) If yield$(\gamma) \neq \lambda$, then by the induction hypothesis of (i), there exists $\gamma' \in T_\Sigma$ such that $C \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \gamma'$, and yield$(\gamma') =$ yield(γ). By the definition of P', $A(x) \to b(C, x)$ is in P'. Therefore, $A(x) \underset{\mathcal{G}'}{\Rightarrow} b(C, x) \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} b(\gamma', x)$, and yield$(b(\gamma', x)) =$ yield$(b(\gamma, x))$. (b) If yield$(\gamma) = \lambda$, then $C \in E_0$, and $A(x) \to c(x)$ is in P'. Therefore, $A(x) \underset{\mathcal{G}'}{\Rightarrow} c(x)$, and yield$(c(x)) =$ yield$(b(\gamma, x))$. The proof of the case (3) is similar to that of the case (2).

[Proof of (iii)] If $A(x) \underset{\mathcal{G}}{\overset{k}{\Rightarrow}} \alpha$, then the rule used at the first step is one of the follwoing form: (1) $A(x) \to B(C(x))$, (2) $A(x) \to b(C, x)$ or (3) $A(x) \to b(x, C)$. In the case (1), $A(x) \underset{\mathcal{G}}{\Rightarrow} B(C(x)) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \beta[\gamma] = \alpha$ for some $\beta, \gamma \in T_\Sigma(X_1)$ such that $B(x) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \beta$ and $C(x) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \gamma$. By the definition of P', $\overline{A} \to B(\overline{C})$ is in P'. By the induction hypothesis of (ii), there exists $\beta' \in T_\Sigma(X_1)$ such that $B(x) \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \beta'$ and yield$(\beta') =$ yield(β). By the induction hypothesis of (iii), there exists $\gamma' \in T_\Sigma$ such that $\overline{C} \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \gamma'$ and yield$(\gamma') =$ yield$(\gamma[\varepsilon])$. Therefore, $\overline{A} \underset{\mathcal{G}'}{\Rightarrow} B(\overline{C}) \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \beta'[\gamma']$ and yield$(\beta'[\gamma']) =$ yield$(\beta[\gamma[\varepsilon]])$. In the case (2), $A(x) \underset{\mathcal{G}}{\Rightarrow} b(C, x) \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} b(\gamma, x) = \alpha$ for some $\gamma \in T_\Sigma$ such that $C \underset{\mathcal{G}}{\overset{*}{\Rightarrow}} \gamma$ and yield$(\gamma) \neq \lambda$. By the definition of P', $\overline{A} \to c(C)$ is in P'. By the induction hypothesis of (i), there exists $\gamma' \in T_\Sigma$ such that $C \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} \gamma'$ and yield$(\gamma') =$ yield(γ). Therefore, $\overline{A} \underset{\mathcal{G}'}{\Rightarrow} c(C) \underset{\mathcal{G}'}{\overset{*}{\Rightarrow}} c(\gamma')$, and yield$(c(\gamma')) =$ yield$(b(\gamma, x)[\varepsilon])$. The proof of the case (3) is similar to that of the case (2).

By (i), we have the result $L_S(\mathcal{G}') = L_S(\mathcal{G})$. □

4 Lexicalization of Spine Grammars

In this section, lexicalization of spine grammars is discussed. First, it is seen that there exists a tree language generated by a spine grammar that no lexicalized spine grammar can generate. Next, it is shown that for any spine grammar, there exists a weakly equivalent lexicalized spine grammar. In the construction of a lexicalized spine grammar, the famous technique to construct a context-free grammar (CFG) in Greibach normal form (Hopcroft and Ullman, 1979) is employed. The technique can be adapted to spine grammars because paths of derivation trees of spine grammars can be similarly treated as derivation strings of CFGs.

Definition 4.1 A spine grammar $\mathcal{G} = (N, \Sigma, P, S)$ is *lexicalized* if it is epsilon-free and for any rule $A(x_1, x_2, \ldots, x_n) \to \alpha$ in P, α has exactly one leaf labeled by a terminal and the other leaves are labeled by a nonterminal or a variable.

The following example is a spine grammar that no lexicalized spine grammar is equivalent.

Example 4.2 Let us consider the spine grammar $\mathcal{G} = (N, \Sigma, P, S)$ such that $\Sigma = \{a, b\}$ with $r(a) = 0$ and $r(b) = 1$, $N = \{S\}$ with $r(S) = 0$, and P consists of $S \to a$ and $S \to b(S)$. The tree language generated by $\mathcal{G}$ is $L(\mathcal{G}) = \{a, b(a), b(b(a)), b(b(b(a))), \ldots\}$. Suppose that $L(\mathcal{G})$ is generated by a lexicalized spine grammar $\mathcal{G}'$. Because $L_S(\mathcal{G}) = \{a\}$, all trees in $L(\mathcal{G})$ have to be derived in one step, and the set of rules of $\mathcal{G}'$ has to be $\{S \to a, S \to b(a), S \to b(b(a)), S \to b(b(b(a))), \ldots\}$. However, the number of rules of $\mathcal{G}'$ has to be finite. Therefore, $L(\mathcal{G})$ can not be generated by any lexicalized spine grammar.

To prove the main theorem, the following lemmas are needed.

Lemma 4.3 *For any spine grammar $\mathcal{G}$, we can construct a weakly equivalent spine grammar in normal form $\mathcal{G}' = (N, \Sigma, P, S)$ that doesn't have a rule of the form $A(x) \to b(x)$ with $A \in N_1$ and $b \in \Sigma_1$.*

Proof. Without loss of generality, we may assume that $\mathcal{G}$ is in normal form. For each rule of the form $A(x) \to b(x)$, delete it and add the rule $A(x) \to x$. Then, a weakly equivalent grammar is constructed. □

Lemma 4.4 *For any spine grammar $\mathcal{G}$, we can construct an equivalent spine grammar in normal form $\mathcal{G}' =$*

(N, Σ, P, S) *that doesn't have a rule of the form* $A(x) \to x$ *with* $A \in N_1$.

Proof. Omitted. □

The following lemmas guarantees that the technique to construct a CFG in Greibach normal form (Hopcroft and Ullman, 1979) can be adapted to spine grammars.

Lemma 4.5 *Define an A-rule to be a rule with a nonterminal A on the left-hand side. Let* $\mathcal{G} = (N, \Sigma, P, S)$ *be a spine grammar such that* $r(A) \leq 1$ *for all* $A \in N$.

For $A \in N_0$, *let* $A \to \alpha$ *be a rule in* P *such that* $\alpha(d) = B$ *for some* $d \in D_\alpha$ *and* $B \in N_0$. *Let* $\{B \to \beta_1, B \to \beta_2, \ldots, B \to \beta_r\}$ *be the set of all B-rules. Let* $\mathcal{G}' = (N, \Sigma, P', S)$ *be obtaind from* $\mathcal{G}$ *by deleting the rule* $A \to \alpha$ *from* P *and adding the rules* $A \to \alpha\langle d \leftarrow \beta_i\rangle$ *for all* $1 \leq i \leq r$. *Then* $L(\mathcal{G}') = L(\mathcal{G})$.

For $A \in N_1$, *let* $A(x) \to \alpha$ *be a rule in* P *such that* $\alpha(d) = B$ *for some* $d \in D_\alpha$ *and* $B \in N_1$. *Let* $\{B(x) \to \beta_1, B(x) \to \beta_2, \ldots, B(x) \to \beta_r\}$ *be the set of all B-rules. Let* $\mathcal{G}'' = (N, \Sigma, P'', S)$ *be obtaind from* $\mathcal{G}$ *by deleting the rule* $A(x) \to \alpha$ *from* P *and adding the rules* $A(x) \to \alpha\langle d \leftarrow \beta_i[\alpha/d \cdot 1]\rangle$ *for all* $1 \leq i \leq r$. *Then* $L(\mathcal{G}'') = L(\mathcal{G})$.

Proof. Omitted. □

Lemma 4.6 *Let* $\mathcal{G} = (N, \Sigma, P, S)$ *be spine grammar such that* $r(A) \leq 1$ *for all* $A \in N$.

For $A \in N_0$, *let* $A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_r$ *be the set of A-rules such that for all* $1 \leq i \leq r$, *there exists a leaf node* $d_i \in D\alpha_i$ *labeled by A. Let* $A \to \beta_1, A \to \beta_2, \ldots, A \to \beta_s$ *be the remaining A-rules. Let* $\mathcal{G}' = (N \cup \{Z\}, \Sigma, P', S)$ *be the spine grammar formed by adding a new nonterminal* Z *to* N_1 *and replacing all the A-rules by the rules:*

$$1) \quad \left.\begin{array}{l} A \to \beta_i \\ A \to Z(\beta_i) \end{array}\right\} 1 \leq i \leq s$$

$$2) \quad \left.\begin{array}{l} Z(x) \to \alpha_i\langle d_i \leftarrow x\rangle \\ Z(x) \to Z(\alpha_i\langle d_i \leftarrow x\rangle) \end{array}\right\} 1 \leq i \leq r$$

Then $L(\mathcal{G}') = L(\mathcal{G})$.

For $A \in N_1$, *let* $A(x) \to A(\alpha_1), A(x) \to A(\alpha_2), \ldots, A(x) \to A(\alpha_r)$ *be the set of A-rules such that A is the label of the root node of the right-hand side. Let* $A(x) \to \beta_1$, $A(x) \to \beta_2, \ldots, A(x) \to \beta_s$ *be the remaining A-rules. Let* $\mathcal{G}'' = (N \cup \{Z\}, \Sigma, P'', S)$ *be the spine grammar formed by adding a new nonterminal* Z *to* N_1 *and replacing all the A-rules by the rules:*

$$1) \quad \left.\begin{array}{l} A(x) \to \beta_i \\ A(x) \to \beta_i[Z(x)] \end{array}\right\} 1 \leq i \leq s$$

$$2) \quad \left.\begin{array}{l} Z(x) \to \alpha_i \\ Z(x) \to \alpha_i[Z(x)] \end{array}\right\} 1 \leq i \leq r$$

Then $L(\mathcal{G}'') = L(\mathcal{G})$.

Proof. Omitted. □

Theorem 4.7 *For any spine grammar* $\mathcal{G} = (N, \Sigma, P, S)$, *we can construct a weakly equivalent lexicalized spine grammar* $\mathcal{G}'$.

Proof. Since it is enough to show the existence of a weakly equivalent grammar, without loss of generality, we may assume that $\mathcal{G}$ is epsilon-free and $\mathcal{G}$ is in normal form without a rule of the form $A(x) \to b(x)$ with $A \in N_1$ and $b \in \Sigma_1$. We may also assume that $\mathcal{G}$ is in normal form without a rule of the form $A(x) \to x$ with $A \in N_1$.

Each rule in P is one of the following form:

Type 1 $A \to a$ with $A \in N_0$ and $a \in \Sigma_0$,

Type 2 $A \to B(C)$ with $A \in N_0$, $B \in N_1$, and $C \in N_0$,

Type 3 $A(x) \to b(C_1, C_2, \ldots, C_n)$ with $A \in N_1$, $n \geq 2$ $b \in \Sigma_n$, and $C_1, C_2, \ldots, C_n$ such that all are in N_0 but $C_i = x$ for exactly one $i \in \{1, \ldots, n\}$, or

Type 4 $A(x) \to B_1(B_2(\cdots(B_m(x))\cdots))$ with $A \in N_1$, $m \geq 1$, and $B_1, B_2, \ldots, B_m \in N_1$.

Because of the assumption above, $m \geq 1$ and $n \geq 2$.

First, by the technique to construct a CFG in Greibach normal form, we replace all type 1 and type 2 rules with rules of the form $A \to B_1(B_2(\cdots(B_m(a))\cdots))$ with $m \geq 0$, $a \in \Sigma_0$, and $B_1, \ldots, B_m \in N_1$ and new type 4 rules with a new nonterminal on the left-hand side. See (1) in Figure 4.

Secondly, we consider type 4 rules of the form $A(x) \to B(x)$. By the standard technique of formal language theory, those rules can be replaced by other type 4 rules with at least 2 nonterminals on the right-hand side.

Thirdly, by the technique to construct a CFG in Greibach normal form, we replace all type 1 and type 2 rules with $A(x) \to b(\gamma_1, \gamma_2, \ldots, \gamma_n)$ with $\gamma_1, \gamma_2, \ldots, \gamma_n$ such that all are in N_0 but $\gamma_i \in T_{N_1}(X_1)$ for exactly one $i \in \{1, \ldots, n\}$. See (2) in Figure 4.

Lastly, the remaining non-lexicalized rules are only of the form $A(x) \to b(\gamma_1, \gamma_2, \ldots, \gamma_n)$. Because $n \geq 2$, the right-hand side has a node labeled by a nonterminal in N_0. This node can be replaced by the right-hand side of the rules of the form $C_i \to D_1(D_2(\cdots(D_m(a))\cdots))$. See (3) in Figure 4.

A weakly equivalent lexicalized spine grammar $\mathcal{G}'$ is constructed. □

If $\mathcal{G}$ is epsilon-free and dosen't have a rule of the form $A(x) \to b(x)$ with $A \in N_1$ and $b \in \Sigma_1$, then the equivalence of the construted grammar is preserved. The following corollary is immediate.

Corollary 4.8 *For any epsilon-free spine grammar* $\mathcal{G} = (N, \Sigma, P, S)$ *such that* $\Sigma_1 = \emptyset$, *we can construct an equivalent lexicalized spine grammar* $\mathcal{G}'$.

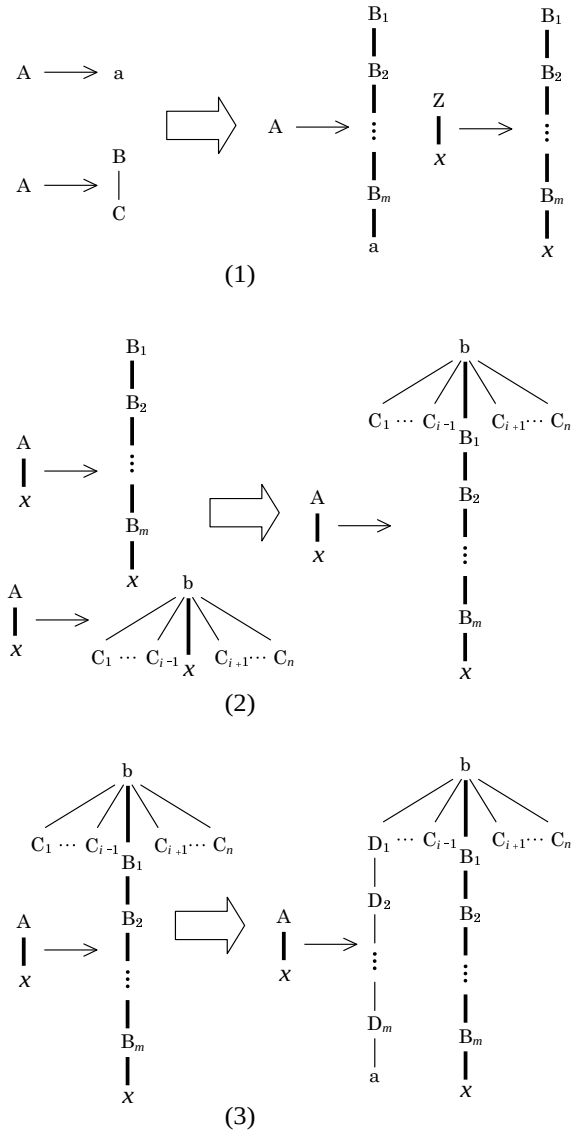


Figure 4: The explanation for the proof

5 The Results for TAGs

From a spine grammar, a weakly equivalent TAG is obtained easily. Recall the definition of TAGs in the paper (Joshi and Schabes, 1996). Let $\mathcal{G} = (N, \Sigma, P, S)$ be a spine grammar in strong normal form. A weakly equivalent TAG $\mathcal{G}' = (\Sigma_0, N \cup (\Sigma - \Sigma_0), I, A, S)$ can be constructed as follows. The set of initial trees is the smallest set satisfying following conditions:

- The tree $S\downarrow$ is in I.
- If $A \rightarrow a$ with $A \in N_0$ and $a \in \Sigma_0$ is in P, then $A_{NA}(a)$ is in I.
- If $A \rightarrow B(C)$ with $A \in N_0$, $B \in N_1$, and $C \in N_0$ is in P, then $A_{NA}(B_{OA}(C\downarrow))$ is in I.

The set of auxiliary trees is the smallest set satisfying following conditions:

- If $A(x) \rightarrow B(C(x))$ with $A \in N_1$ and $B, C \in N_1$ is in P, then $A_{NA}(B_{OA}(C_{OA}(A_{NA}*)))$ is in A.
- If $A(x) \rightarrow b(C, x)$ with $A \in N_1$, $b \in \Sigma_2$ and $C \in N_0$ is in P, then $A_{NA}(b_{NA}(C\downarrow, A_{NA}*))$ is in A.
- If $A(x) \rightarrow b(x, C)$ with $A \in N_1$, $b \in \Sigma_2$ and $C \in N_0$ is in P, then $A_{NA}(b_{NA}(A_{NA}*, C\downarrow))$ is in A.

The way to construct a weakly equivalent TAG from a spine grammar in strong normal form was shown. By a similar way, a weakly equivalent TAG is effectively obtained from any epsilon-free or lexicalized spine grammar constructed in this paper without utilizing epsilon-rules or breaking lexicality. Therefore, the results for spine grammars also hold for TAGs.

Corollary 5.1 *For any TAG, we can construct a weakly equivalent epsilon-free TAG.*

Corollary 5.2 *For any TAG, we can construct a weakly equivalent lexicalized TAG.*

References

Anne Abeillé and Owen Rambow, editors. 2000. *Tree adjoining grammars: formalisms, linguistic analysis and processing*. CSLI Publications, Stanford, California.

Akio Fujiyoshi and Takumi Kasai. 2000. Spinal-formed context-free tree grammars. *Theory of Computing Systems*, 33(1):59–83.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, Reading, Massachusetts.

Aravind K. Joshi and Yves Schabes, 1996. *Handbook of Formal Languages*, volume 3, chapter Tree-adjoining grammars, pages 69–124. Springer, Berlin.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *J. Computer & System Sciences*, 10(1):136–163.

Uwe Möennich. 1997. Adjunction as substitution: an algebraic formulation of regular, context-free and tree adjoining languages. In G. V. Morrill G-J. Kruijff and R. T. Oehrle, editors, *Formal Grammars 1997: Proceedings of the Conference*, Aix-en-Provence, pages 169–178.

Uwe Möennich. 1998. TAGs M-constructed. In *TAG+ 4th Workshop*, Philadelphia.

William C. Rounds. 1970. Mapping and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287.

K. Vijay-Shanker and David J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546.

N-Best Hidden Markov Model Supertagging to Improve Typing on an Ambiguous Keyboard

Saša Hasan[*] and **Karin Harbusch**
Computer Science Department
University of Koblenz-Landau
{hasan|harbusch}@informatik.uni-koblenz.de

Abstract

Ambiguous keyboards, i.e. several letters host on the same key (cf. telephone keyboard), produce candidate lists with words that match the entered code. These lists have to be disambiguated by the user for the intended word. Consequently, the primary goal is to order these candidates in a way that the most appropriate words are placed on top of the suggestion list for minimal selection costs or to postpone the attention to suggestions to a final editing step. This paper reports on promising results for this goal by inspecting the whole sentence on the basis of supertagging and lightweight dependency analysis.

1 Introduction

Watch-sized devices which lack space for a full keyboard, i.e. uniquely addressable keys, on the one hand, and on the other, Asian language typing or devices for speech- and motor-impaired people where not all letters can be addressed directly because so many buttons are not manageable in reasonable time make use of so-called *ambiguous* or *reduced* or *cluster* keyboards (for one of the earliest systems see (Witten, 1982)) — even only realized virtually on a screen (which does not make any difference for the following considerations). Typing with ambiguous keyboards, i.e. keyboards where several letters, symbols or numbers, respectively, host on one and the same button (cf. telephone keyboards), basically can be performed in two different manners. *Multi-tapping* as basic encoding method for short message sending (SMS) on cellular phones uniquely addresses a symbol by a predefined number of button hits in a row. Obviously, this method is cumbersome and time consuming.

As a consequence of an observation by (Witten, 1982), namely that in a dictionary with 24500 words only 8% are ambiguous if the respective button on a phone keyboard is pressed only once, predictive methods have emerged on the market. *Predictive* text entry devices (e.g., the product T9 by Tegic Communications for SMS typing (Kushler, 1998)) have been developed that reduce the number of keystrokes needed for entering a word by proposing possible candidates matching the current input. Moreover, the possible candidates for *completion* all match the already entered prefix of the word. By selecting one of the available suggestions (irrespective whether assuming prediction or completion mode here), the number of keypresses decreases but the overall time to enter the word is not necessarily reduced due to the cognitive load that emerges while scanning the suggested candidate list (see, e.g., (Horstmann Koester and Levine, 1994)). Consequently, the primary goal is to order these candidates in a way that the most appropriate words are placed on top of the suggestion list for minimal selection costs or to postpone the attention to suggestion lists to a final editing step. This paper reports on promising results for this goal by inspecting the whole sentence on the basis of supertagging and lightweight dependency analysis (LDA).

The paper is organized as follows. Section 2 gives a short overview on related work in the area of predictive typing. Section 3 presents the sentence-wise approach based on supertagging and LDA. The experiments and results achieved with these methods are summarized in Section 4. Section 5, finally, summarizes our approach and the conclusions we reached.

2 State of the art in predictive typing

Only for completeness, we mention *letter-wise* predictive systems here (see, e.g., LetterWise by (MacKenzie et al., 2001)) or the Reactive Keyboard by (Darragh and Witten, 1992). For *word-wise* systems, the easiest way to achieve appropriate suggestion lists is to sort the list according to word frequencies obtained from large corpora (see the

[*]The author is now affiliated with the Chair of Computer Science VI, RWTH Aachen University, Germany.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 24-31.

proceedings of the EACL-workshop on language modeling for text entry methods (Harbusch et al., 2003) for an overview of recent n-gram systems). Tanaka et al. (2002) propose an adaptive language model utilizing prediction by partial match (PPM (Cleary and Witten, 1984), which actually originates from the information theory domain and deals with the problem of improving the compression rates of arithmetic coding) that lowers the entropy of a language model by maintaining a list of already seen contexts and its corresponding successors. In (Matiasek et al., 2002), a system based on word n-grams with additional part-of-speech information is outlined. Fazly and Hirst (2003) also impose part-of-speech information on prediction. Surprisingly, additional part-of-speech information hardly improves the prediction lists. Thus, other information sources have to be investigated.

The only approach that goes beyond a word-wise step-by-step disambiguation we are aware of is reported in (Rau and Skiena, 1996). Instead of permanently changing between two modes, i.e. a phase where a word is typed and a phase where it is disambiguated in a list of current suggestions, the user can solely concentrate on the process of text entry in a *sentence-wise* approach. Here, a telephone keypad that distributes the 26 letters and the blank character (word delimiter) on 9 keys serves as ambiguous keyboard (i.e. 3 letters are placed on one key at a time). The end of the sentence is marked unambiguously using the "#" key. Sentence disambiguation applies the Viterbi algorithm and involves word bigram probabilities and part-of-speech information extracted from the Brown Corpus. The results obtained by simulating the typing of various text samples with this framework look promising. For various domains, the percentage of correct words ranges from 92.86 to 97.67%. This is due to the relatively high number of keys and low number of ambiguous words, respectively.

A possible way of entering commonly used expressions such as "how are you" or "could you please open the door" fast is the use of *sentence compansion* (see e.g. in (Copestake, 1997; McCoy et al., 1998)). So, from the input "open door", the system would generate "could you please open the door". This *cogeneration* approach needs three knowledge sources, namely a grammar and a lexicon, statistical information about collocations, i.e. syntagmatically related words, and a set of templates. A thinkable drawback for the user might be that the expanded sentences sound monotonous by and by. In contrast, flexibility and individuality are valuable features of direct and unrestricted text entry systems. Thus, the motivation of typing on a sentence level is reasonable. The idea is to make use of the syntactic relations that exist in the sentence the user wants to express and exploit them to present more accurate candidate lists that allow for faster selection by moving likely matches to the top.

a g j l m q r w z ä	c f h k o s t u v x y ü ß	b d e i n p ö -
Button 1	Button 2	Button 3

Figure 1: The layout of the letter keys for German.

3 Sentence-wise predictive typing based on Supertagging

In this paper, we report on results for a sentence-wise text entry system with a highly reduced ambiguous keyboard containing only three letter keys and one command key (the setting results from needs of our disabled test subjects). Nevertheless, the presented system called UKO-II (Harbusch and Kühn, 2003) is adaptive with respect to the number of keys, i.e. the system can be tailored to any number of keys where the symbol distribution is matter of definition.

The distribution of the letters on the keys that is used in this work is language-specific by applying a genetic algorithm which optimizes the candidate lists' length and overall selection costs for a given lexicon. For German and English, the dictionaries are based on the CELEX lexical database (Baayen et al., 1995). The current keyboard layout of the letter keys for the German language is depicted in Figure 1. In contrast to the approach in (Rau and Skiena, 1996), the word delimiter (space) is coded unambiguously by entering the command mode.[1] So for example, in order to enter *guten Morgen* ("*good morning*") the user types the code sequence 1 2 2 3 3 ␣ 1 2 1 1 3 3. For the first code, there exist 48 possible words (*guten, außen, wohin, ...*), for the second, there are 30 entries (*wollen, morgen, Morgen, ...*). This small example already allows for a total of 1440 sentence hypotheses.

3.1 N-best Supertagger

The entire technique that is chosen to achieve our goal is based on *supertagging*, a procedure that associates so-called supertags, i.e. elementary trees in the grammar formalism of Lexicalized Tree-Adjoining Grammar (see, e.g., (Joshi and Schabes, 1997)) that code local dependencies, with the corresponding words of the sentence (see, e.g., (Bangalore and Joshi, 1999)). The core of the presented system is an n-best supertagger that is based on a second-order Hidden Markov Model (see (Bäcker and Harbusch, 2002)) and is able to find the n best sentence hypotheses for a sequence of coded words.

Let $t_1^N = t_1 t_2 \cdots t_N$ be a sequence of supertags for a sentence $w_1^N = w_1 w_2 \cdots w_N$. We are interested in the

[1] In command mode, the mapping of the letter keys is changed to commands like *delete last key* or *space*. Thus, the command button functions as a meta key and allows for hierarchical menu structures which are not further discussed here.

most probable tag sequence $\hat{t}_1^N$ which is defined by

$$\hat{t}_1^N = \text{argmax}_{t_1^N} P(t_1^N|w_1^N). \quad (1)$$

According to Bayes' law and additional assumptions that the words are independent of each other, the probability of a supertag sequence given a sentence, $P(t_1^N|w_1^N)$, can be rewritten as

$$P(t_1^N|w_1^N) \approx \prod_{i=1}^{N} P(t_i|t_{i-2}t_{i-1})P(w_i|t_i), \quad (2)$$

where *maximum likelihood estimation* (MLE) is used for the probabilities by relative frequencies derived from an annotated training set of supertagged sentences. For unknown events, Good-Turing discounting in combination with Katz's back-off is applied.

Usually, a dynamic programming technique (i.e. the Viterbi algorithm) finds the best supertag sequence of a sentence (cf. Equation 1) for a given HMM efficiently by storing the best local sequence probability and a backpointer to the predecessor state for each word position in the sentence. In order to find the n best paths through the HMM trellis, we have to allow the backpointer table to hold not only the best predecessor, but the n best predecessor states sorted by the corresponding log probability. Since we deal with trigrams in Equation 2, the states of the HMM have to be coded as supertag pairs, thus $P(t_i|t_{i-2}t_{i-1}) = P(\langle t_{i-1}, t_i\rangle|\langle t_{i-2}, t_{i-1}\rangle)$. This leads to the following recurrence formula for state probabilities at position k in the sentence $w_1^N, 1 \leq k \leq N$:

$$\delta_k(\langle t_{i-1}, t_i\rangle) = \max_{\langle t_{i-2}, t_{i-1}\rangle} \big[\delta_{k-1}(\langle t_{i-2}, t_{i-1}\rangle) \cdot P(\langle t_{i-1}, t_i\rangle|\langle t_{i-2}, t_{i-1}\rangle)\big] \cdot P(w_k|\langle t_{i-1}, t_i\rangle) \quad (3)$$

The values in the δ-table are used to build an additional table which yields the n best local hypothesis scores

$$\phi_k(s_j, s_i) = \delta_{k-1}(s_i)P(s_j|s_i)P(w_k|s_j) \quad (4)$$

for states $s_i = \langle t_{i-2}, t_{i-1}\rangle$ and $s_j = \langle t_{i-1}, t_i\rangle$. For each s_j, the number of predecessors s_i can be limited to n. The corresponding backpointers are stored in a table $\psi_k(s_j, m) = s_i$ where $m = 1$ denotes the best and $m = n$ the n^{th} predecessing state.

Now, after this forward-trellis step, a backward-tree search is applied in order to find the n most promising supertag sequences which are used to adjust the candidate lists and move likely matches to the top. The evaluation function $f(\langle t_{i-1}, t_i\rangle)$ that associates the current path cost with a state $\langle t_{i-1}, t_i\rangle$ can directly use the log probabilities from the forward-trellis step as a heuristic $h(\langle t_{i-1}, t_i\rangle)$. This approach leads to *greedy search*. An important note is that the heuristic is optimal since it actually returns the *exact* path costs to the goal. By also incorporating the backward partial path costs $g(\langle t_{i-1}, t_i\rangle)$ of the search process, i.e. $f = g + h$, we arrive at *A* search*. The resulting system is able to generate the n best supertag hypotheses for a given sentence. For a more detailed presentation of the system, see (Hasan, 2003).

3.2 Incorporating ambiguous codes

The starting point is an ambiguously coded word sequence typed with a reduced keyboard as introduced in the beginning of this section. Every code generates a list of words and every word has several supertags associated with it. A supertagger is used to find the most likely supertag sequence for the sentence and on the basis of this information, the candidate list becomes reordered such that the most likely words (which are the lexical anchors of the supertags) appear at the top. Due to the ambiguous coding, the number of supertags for a code (which corresponds to the supertags of all word expansions of a code) is so large that the best supertag sequence is not sufficient to improve the results significantly. Therefore, we use the n-best tree-trellis approach from Section 3.1 in order to produce more than one hypothesis. At this point, the code sequence of each sentence is associated with a list of the n best supertag sequences found by the supertagger.

Every word usually has several supertags, since the lexical items of an LTAG are almost always associated with several elementary structures that encode the various local dependencies of each word. And since every code expands to several matching words, the result is a set of supertag sets that form a trellis (cf. detailed view in Figure 2). This trellis is the basis for the tree-trellis search that finds the n best supertag hypotheses for a given sentence. Figure 2 also shows the different expansion steps for the sentence *ich habe ein kleines Problem* ("*I have a little problem*").

After typing the words of a sentence with the ambiguous keyboard, the code sequence is expanded and the candidate list is obtained according to the CELEX lexicon. After that, the possible supertags are looked up in the trained language model, i.e. all supertags that occurred in the training corpus with its corresponding lexical anchor are primed for the n-best tree-trellis search. The hypotheses that are returned by the search are then used to reorder the candidate lists. The effect is that likely words of the trained language model will move to the top of the match lists and improve the overall accuracy of the system.

3.3 Filtering ungrammatical hypotheses

In a second step, a *lightweight dependency analysis* (Bangalore, 2000) on the list of supertag hypotheses found by the n-best search is used as an additional knowledge source in order to determine likely chunks of the sentence. The dependencies coded in the elementary trees

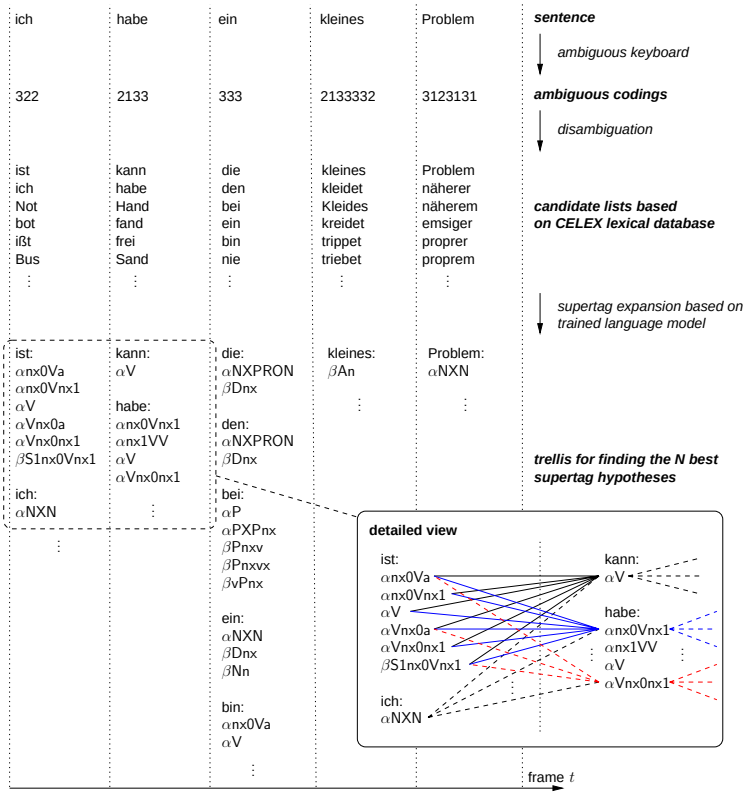


Figure 2: Coping with ambiguous words: disambiguation of coded words and the corresponding supertag expansion.

(supertags) can be used to actually derive a shallow parse of the sentence in linear time.[2] We use a *dependency coverage* criterion which determines how many dependency slots of each supertag are filled to the left and right of its lexical anchor. The hypotheses that have a maximum number of covered elements are used to adjust the final candidate lists, i.e. the supertag hypotheses that span the largest portion of the sentence and seem most "consistent" are moved to the top. This method is applied in order to discard hypotheses that have syntactic inconsistencies.

Figure 3 illustrates the rearrangements of the ambiguously typed sentence *ich habe ein kleines Problem* ("*I have a little problem*"). The three marked hypotheses all have a maximum coverage of 5, i.e. all supertags have their dependency slots filled, whereas the other hypotheses have coverages less than 5. One can see that local word probabilities would suggest *ist kann die kleines Problem* ("*is can the little problem*"). The information that is provided by the surviving hypotheses is used for additional final adjustments of the candidate lists to be presented to the user. We call this reordering process *match list boosting*, or shortly *boosting* (see Figure 3 for an example).

[2]We decided for LDA because it considers more syntactic knowledge than simple chunking techniques, while still being very efficient in comparison to full TAG parsing.

4 Evaluation

For an evaluation of the techniques presented in the previous section, the ambiguous typing of a sample text is simulated and processed with the n-best supertagger. As performance criteria, the *accuracy* and the *average rank* of the correct word are compared to the values obtained from the baseline approach using the word frequencies from the CELEX lexical database (approx. 320 000 word types). For this purpose, a lexicalized tree adjoining grammar is needed because of the lightweight dependency analysis performed in the last step of the n-best approach. The trigram HMM is directly trained on a corpus that is annotated with supertags.

For the experiments, the corpus and LTAG developed in (Bäcker and Harbusch, 2002) is used. It comprises 250 labeled sentences (approx. 2000 words), whereof 225 are used for training and 25 for testing. Since the corresponding LTAG is rather small, containing only 127 elementary trees (58 initial and 69 auxiliary trees), this directly impacts on the size of the trained HMM and the runtime of the LDA. Therefore, it was possible to run the n-best supertagger for up to 2000 hypotheses in an acceptable

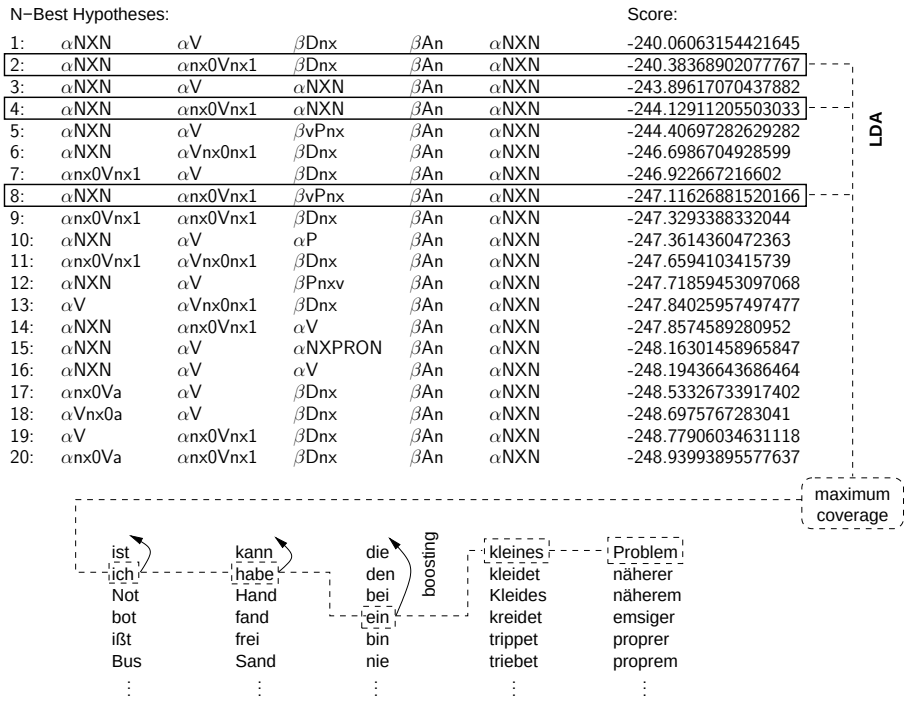


Figure 3: Example for rearranging word hypotheses according to the results of supertagging and LDA.

amount of time. On a 1.4GHz AMD Athlon, the evaluation of the reference test set needs approx. 10.58s for $n = 250$, i.e. 423ms per sentence. The adjustments of the match lists can therefore be performed in real-time for smaller values of n.

Coping with unknown words in ambiguous typing is a more complicated problem. If the word is not in the dictionary, it has to be disambiguated letter by letter for all the keys of the code. Since the primary goal was not to simulate a specific keyboard but to evaluate whole sentences with the n-best supertagging framework, the dictionary was patched by adding the unknown words with a zero-frequency and thus contained all words of the corpus.

4.1 Baseline

The baseline results are achieved with the simple unigram approach where the frequencies of the words that are stored in the lexicon order the candidate list in descending order, i.e. with highest frequency first. As evaluation criteria, the *accuracy of rank* r and the *average match position* is chosen. More formally, let

$$f_r(w|c) = \begin{cases} 1 & \text{if } w \in \text{matches}(c) \wedge \text{rank}(w) = r \\ 0 & \text{else} \end{cases} \tag{5}$$

be a binary function that returns 1 if a disambiguated target word w correctly occurs on the r^{th} position of the candidate list of its code c, which is given by $\text{matches}(c)$.

Reference test set evaluation, $\bar{r} = 3.02$					
	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$\text{acc}(r)$ [%]	50.26	28.04	5.29	7.41	1.59
$\text{cac}(r)$ [%]	50.26	78.30	83.59	91.00	92.59

Table 1: The baseline results of ambiguously typing the test corpus.

For a test corpus containing a total of N words, the accuracy of rank r for the given corpus can be computed as

$$\text{acc}(r) = \frac{\sum_w f_r(w|c)}{N}. \tag{6}$$

For a *cumulative accuracy*, i.e. where the target words appear within the first r ranks of the candidate lists, the single accuracy values are summed:

$$\text{cac}(r) = \sum_{i=1}^{r} \text{acc}(i). \tag{7}$$

The second evaluation measure is the *average rank* of words of the test corpus. It is simply computed by

$$\bar{r} = \frac{\sum_w \text{rank}(w)}{N}. \tag{8}$$

The results for the baseline are outlined in Table 1. Apparently, the unigram approach places approx. 50% of the target words on the first position of the candidate lists. 92.6% of the words appear within the first 5 ranks. The

Reference test set evaluation					(a)
Average for $n = 1, \ldots, 1000$, $\bar{r} = 2.18$					
	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$\mathrm{acc}(r)$ [%]	61.84	23.01	1.84	8.38	0.55
$\mathrm{cac}(r)$ [%]	61.84	84.85	86.69	95.07	95.62
Single best results					(b)
Overall best for $n = 592$, $\bar{r} = 2.11$					(b.1)
$\mathrm{acc}(r)$ [%]	66.67	18.52	1.59	8.47	0.53
$\mathrm{cac}(r)$ [%]	66.67	85.19	86.78	95.25	95.78
Best accuracy/time trade-off for $n = 250$, $\bar{r} = 2.16$					(b.2)
$\mathrm{acc}(r)$ [%]	61.90	22.75	2.12	8.47	0.53
$\mathrm{cac}(r)$ [%]	61.90	84.65	86.77	95.24	95.77
Trigram experiment					(c)
Average for $n = 1, \ldots, 1000$, $\bar{r} = 2.91$					
$\mathrm{acc}(r)$ [%]	60.01	19.64	4.09	6.59	2.26
$\mathrm{cac}(r)$ [%]	60.01	79.65	83.74	90.33	92.59
Upper bound experiment					(d)
Average for $n = 1, \ldots, 1000$, $\bar{r} = 2.11$					
$\mathrm{acc}(r)$ [%]	68.07	16.92	1.85	8.22	0.55
$\mathrm{cac}(r)$ [%]	68.07	84.99	86.84	95.07	95.62

Table 2: The improved results using the n-best supertagger/LDA system and additional experiments with trigrams and an upper bound.

rank expectation for the reference test set is 3, i.e. the user has to scroll two times on average before selecting the desired word.

4.2 N-best system

This section reports on the improvements obtained with the system using the n-best supertagger and additional LDA. The results show that the approach yields better rankings than the simple word-wise prediction method (baseline) and also outperforms a trigram language model. The overall results are shown in Table 2. The first part (a) shows the values computed for the reference test set, namely the average for the full evaluation runs with hypothesis sizes ranging from 1 to 1000. When comparing the values to those in Table 1, a significant improvement for the reference test set is visible. The cumulative accuracy of rank 1 raises by approx. 12%, i.e. 61.8% of the target words are now placed on top of the candidate lists. For the other ranks, the improvement is not as big as for rank 1, but there is still a significant increase. With the n-best approach, 95.6% are placed within the top 5 ranks, whereas the average rank drops down to 2.18. The overall best run of this evaluation session is given in (b.1). The maximum occurred for the hypothesis size $n = 592$, i.e. the 592 best supertag sequence hypotheses for the ambiguously coded sentences are used for adjusting the candidate lists. This result also shows that the biggest variation takes place for rank 1. The changes in cumulative accuracy for ranks ≥ 2 are very small for larger values of n. The graphs in Figure 4 give an overview on the differences between the n-best approach and the baseline and also show the slightly better performance of A* search when compared to greedy search.

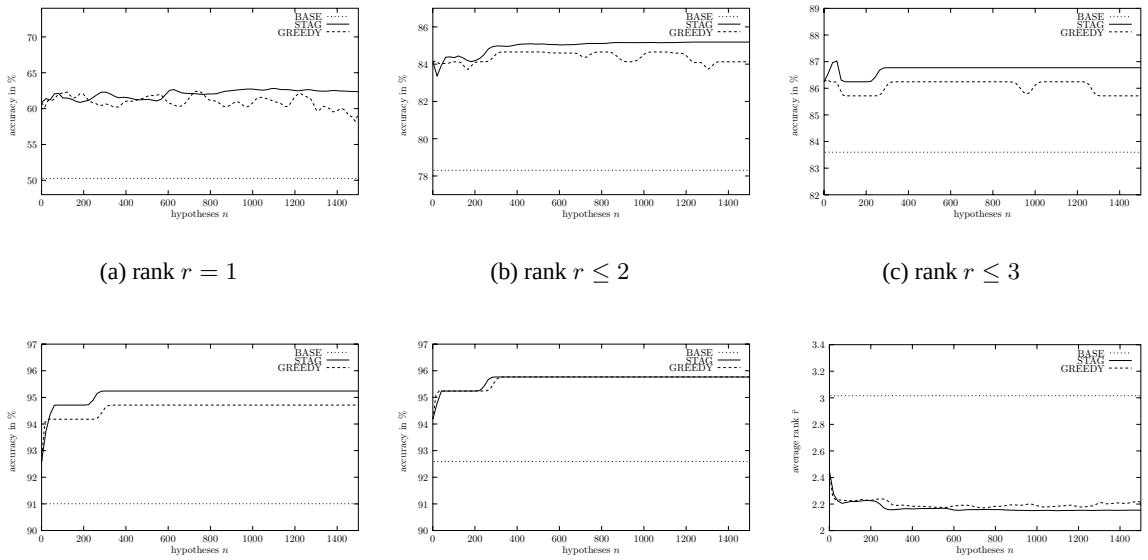
As can be seen in all graphs, enhancing the search from 1-best (Viterbi) to n-best has the largest effect for values of $n < 250$. After approx. 250 hypotheses, the results do not improve significantly, at least for higher cumulative ranks. In general, a hypothesis size of $n = 250$ (cf. Table 2 (b.2)) shows good results since the value for $\mathrm{cac}(5)$ does not increase any more for $n \geq 250$ and the computation time is quite fast.

Another method of evaluating the n-best supertagger is the possibility to look at the target words of the sentences that are typed ambiguously and use only the hypotheses that match closest for adjusting the candidate lists (cf. results in Table 2 (d)). Clearly, this procedure is illegal for an objective evaluation since we are already looking at the desired result we want to achieve, but nevertheless it gives an upper bound of what accuracy the n-best supertagger can theoretically reach by just picking the most promising hypotheses. The detailed evaluation graphs are given in Figure 5. As can be seen, the accuracy between the two approaches differs only for lower ranks (cf. (c)), while for higher ranks (cf. (d) and (e)), the graphs are nearly identical. This means that for the higher rank accuracy, the n-best supertagger already performs in an optimal way for the reference test set and it actually cannot get any better with this kind of training material. It is assumed that with a larger training corpus and thus better language model, the rankings can be further improved.

An interesting constellation is revealed in Figure 5 (a) where the trigram approach outperforms supertagging for lower hypothesis sizes considering rank 1, whereas the accuracy cannot compete for higher ranks (cf. (b)). This is possibly due to the sparseness of data, i.e. the few learned estimations lead to overproportionally many misclassifications for a small hypothesis search space. This claim has to be verified on the basis of more data.

5 Conclusion

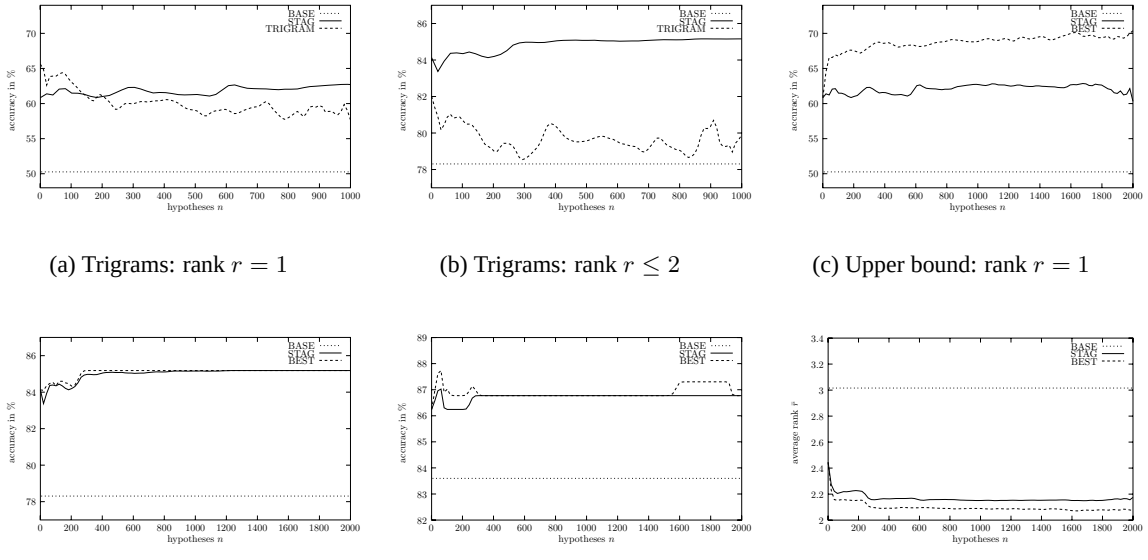
In this paper, we have presented a sentence-wise predictive typing method based on an n-best Hidden Markov Model Supertagger for typing with a four-button ambiguous keyboard. The main idea is to utilize the syntactic dependencies that exist between the words of a sentence and use this information to adjust the candidate lists such that more likely words appear at the top. Instead of being distracted by a list of proposals after every keypress, the user has to pay attention to the prediction list only at the end of the sentence. So the user can concentrate on what (s)he wants to express in the first phase, i.e. the ambiguous typing of the whole sentence, and disambiguate the target words from the candidate lists in a second phase. First evaluations show that users like this mode better than word-wise disambiguation. Further evaluations have



(a) rank $r = 1$ (b) rank $r \leq 2$ (c) rank $r \leq 3$

(d) rank $r \leq 4$ (e) rank $r \leq 5$ (f) average rank $\bar{r}$

Figure 4: Backward search: A* vs. Greedy, cumulative accuracy for ranks $r = 1, \ldots, 5$ and average match position $\bar{r}$. BASE refers to the unigram baseline, STAG/GREEDY is the supertagging+LDA approach with A* and Greedy search, respectively.



(a) Trigrams: rank $r = 1$ (b) Trigrams: rank $r \leq 2$ (c) Upper bound: rank $r = 1$

(d) Upper bound: rank $r \leq 2$ (e) Upper bound: rank $r \leq 3$ (f) Upper bound: average rank $\bar{r}$

Figure 5: Comparison of the baseline (unigram model, BASE), supertagging+LDA (STAG), a trigram approach (TRIGRAM) (Figures (a) and (b)) and an upper bound experiment (BEST) (Figures (c)–(f)).

to be carried out in the future to prove this claim.

As for future work, experiments should be carried out with an enhanced training corpus. Furthermore, a comparison has to be performed with the sentence-wise approach in (Rau and Skiena, 1996). Both systems have to deploy a 9-button keyboard, the LTAG underpinning the supertagging approach and the same lexicon. Under the current circumstances, a direct comparison is not possible.

As it is known (see, e.g., (Baayen, 1991)) that high frequency words often only differ in one letter, and thus, remain highly competitive in all syntactic approaches, we are going to add semantic features (taken from *WordNet* (Miller, 1995)) to the supertags. We expect that rearrangements in the prediction list according to semantic clusters will considerably improve the accuracy of predictions.

References

R. Harald Baayen, Richard Piepenbrock, and Léon Gulikers. 1995. The CELEX lexical database (release 2). CD-ROM, Linguistic Data Consortium, University of Pennsylvania, PA, USA.

R. Harald Baayen. 1991. A stochastic process for word frequency distributions. In *Proceedings of the 29th Meeting of the Association for Computational Linguistics*, pages 271–278, Berkeley, CA, USA.

Jens Bäcker and Karin Harbusch. 2002. Hidden Markov model-based supertagging in a user initiative dialogue system. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 269–278, Venice, Italy.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Srinivas Bangalore. 2000. A lightweight dependency analyzer for partial parsing. *Computational Linguistics*, 6(2):113–138.

John G. Cleary and Ian H. Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Ann Copestake. 1997. Augmented and alternative NLP techniques for Augmentative and Alternative Communication. In *Proceedings of the ACL workshop on Natural Language Processing for Communication Aids*, pages 37–42, Madrid, Spain. Association for Computational Linguistics.

John J. Darragh and Ian H. Witten. 1992. *The Reactive Keyboard*. Cambridge Series on Human-Computer Interaction. Cambridge University Press.

Afsaneh Fazly and Graeme Hirst. 2003. Testing the efficacy of part-of-speech information in word completion. In (Harbusch et al., 2003), pages 9–16.

Karin Harbusch and Michael Kühn. 2003. Towards an adaptive communication aid with text input from ambiguous keyboards. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL03), Conference Companion*, pages 207–210, Budapest, Hungary.

Karin Harbusch, Michael Kühn, and Harald Trost, editors. 2003. *Proceedings of the Workshop on Language Modeling for Text Entry Methods*, Budapest, Hungary. Association for Computational Linguistics.

Saša Hasan. 2003. *N-Best Hidden Markov Model Supertagging for Typing with Ambiguous Keyboards*. Diploma thesis, Computer Science Department, University of Koblenz-Landau, Koblenz, Germany.

Heidi Horstmann Koester and Simon P. Levine. 1994. Modeling the speed of text entry with a word prediction interface. *IEEE Transactions on Rehabilitation Engineering*, 2(3):177–187.

Aravind K. Joshi and Yves Schabes. 1997. Tree Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–214. Springer, Berlin, Germany.

Cliff Kushler. 1998. AAC using a reduced keyboard. In *Proceedings of the Technology and Persons with Disabilities Conference 1998*. Online document available at `http://www.csun.edu/cod/conf/1998/proceedings/csun98_140.htm`.

I. Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. 2001. LetterWise: Prefix-based disabiguation for mobile text input. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 111–120, Orlando, FL, USA.

Johannes Matiasek, Marco Baroni, and Harald Trost. 2002. FASTY — A multilingual approach to text prediction. In K. Miesenberger, J. Klaus, and W. Zagler, editors, *Computers Helping People with Special Needs — Proceedings of the 8th International Conference ICCHP 2002, Linz, Austria*, volume 2398 of *Lecture Notes in Computer Science*, pages 243–250, Berlin, Germany. Springer.

Kathleen F. McCoy, Christopher A. Pennington, and Arlene Luberoff Badman. 1998. Compansion: From research prototype to practical integration. *Natural Language Engineering*, 4(1):73–95.

George A. Miller. 1995. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41.

Harald Rau and Steven S. Skiena. 1996. Dialing for documents: An experiment in information theory. *Journal of Visual Languages and Computing*, 7:79–95.

Kumiko Tanaka-Ishii, Yusuke Inutsuka, and Masato Takeichi. 2002. Entering text with a four-button device. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING '02)*, pages 988–994, Taipei, Taiwan.

Ian H. Witten. 1982. *Principles of Computer Speech*. Academic Press, London, UK.

LTAG Analysis for Pied-Piping and Stranding of wh-Phrases

Laura Kallmeyer
UFRL
University Paris 7
2 place Jussieu
75251 Paris Cedex 05
France
lkallmey@linguist.jussieu.fr

Tatjana Scheffler
Department of Linguistics
619 Williams Hall
University of Pennsylvania
Philadelphia, PA 19104-6305
U.S.A.
tatjana@ling.upenn.edu

Abstract

In this paper we propose a syntactic and semantic analysis of complex questions. We consider questions involving pied piping and stranding and we propose elementary trees and semantic representations that allow to account for both constructions in a uniform way.

1 Introduction

In questions where the wh-word is embedded into a larger NP, there are two structural possibilities, shown in (1) and (2).

(1) (a) The picture of whom does John like?
(b) Which boy's father did you see?

(2) (a) Whom does John like a picture of?
(b) Which painting did you see a photograph of?

The larger NP containing the question word can be pied-piped as in (1) to the beginning of the sentence together with the wh-word. This requires some kind of syntactic or semantic *reconstruction*, i.e.: For scopal purposes, the matrix NP must contribute its semantics (at least in one of the readings) approximately in the position of its trace, while the wh-word itself has of course the widest possible scope.

Native speakers judge pied-piping of embedding NPs ungrammatical in some cases. Particularly, although pied-piping is always fine in relative clauses, a direct question like (3b) is ungrammatical.[1]

(3) (a) On the corner of which street does his friend live?
(b) *A picture of whom does John like?

[1]This was pointed out to us by one anonymous reviewer.

However, as examples (1a) and (3a) show, pied-piping is found with some determiners. We therefore generally allow this construction in the grammar, and attribute the infelicity of some examples to independent factors.

In another construction, shown in (2), the matrix NP can be stranded in its object position, yielding potential problems for semantic compositionality in frameworks that do not use transformations.

Constructions as (2) are claimed to be only possible if all embedding NPs (those which are stranded) are *non-specific*. This goes back to Fiengo and Higginbotham (1981), who show in a much broader context that extraction out of NPs is not possible if an embedding NP is specific. Thus, we get the following judgments:

(4) (a) Who did John see a picture of?
(b) *Who did John see the picture of?
(c) *Who did John see every picture of?

We see that the range of determiners is lexically specified by the construction that they appear in (i.e., the extraction configuration). As for the lexical restrictions with regard to pied-piping above, these effects will not concern us in this paper. They must be dealt with by independent processes, e.g. lexical constraints.

In this paper we show how an approach to the semantics of Tree Adjoining Grammar that uses semantic feature structures and variable unification as in Kallmeyer and Romero (2004) can provide the correct variable bindings for both types of questions. The paper proposes elementary trees and semantic representations that allow to account for both constructions, (1) and (2), in a uniform way.

2 LTAG Semantics

In approaches to TAG semantics (see e.g. Kallmeyer and Joshi, 2003; Joshi et al., 2003; Gardent and Kallmeyer, 2003) each elementary tree is commonly associated with its appropriate semantic representation. In this paper we

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 32-39.

use the framework presented in Kallmeyer and Romero (2004) that follows this line: We use flat semantic representations with unification variables (similar to MRS, Copestake et al., 1999). The semantic representations contain propositional metavariables. Constraints on the relative scope of these metavariables and propositional labels are used to provide underspecified representations of scope ambiguities. To keep track of the necessary variable unifications, semantic feature structures are associated with each node in the elementary tree. For semantic computation, the nodes in the derivation tree contain the semantic information associated with the elementary trees. Semantic feature structures have features POS for all node positions *pos* that can occur in elementary trees.[2] The values of these features are feature structures that consist of a T and a B feature (top and bottom) whose values are feature structures with features I for individual variables, P for propositional labels etc.

Unification follows the usual definitions for unification in Feature-based TAG syntax: For each edge from γ_1 to γ_2 with position p: 1) the T feature of position p in γ_1 and the T feature of the root of γ_2 are identified, and 2) if γ_2 is an auxiliary tree, then the B feature of the foot node of γ_2 and the B feature of position p in γ_1 are identified. Furthermore, for all γ occurring in the derivation tree and all positions p in γ such that there is no edge from γ to some other tree with position p: the T and B features of $\gamma.p$ are identified. By these unifications, some of the variables in the semantic representations get values. Then, the union of all semantic representations is built which yields an underspecified representation.

At the end of a derivation, all possible *disambiguations*, i.e. injective functions from the remaining propositional variables to labels, must be found to obtain the different possible scopings of the sentence. The disambiguated representations are interpreted conjunctively.

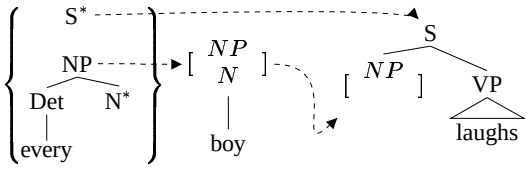


Figure 1: Syntax of (5) *Every boy laughs*.

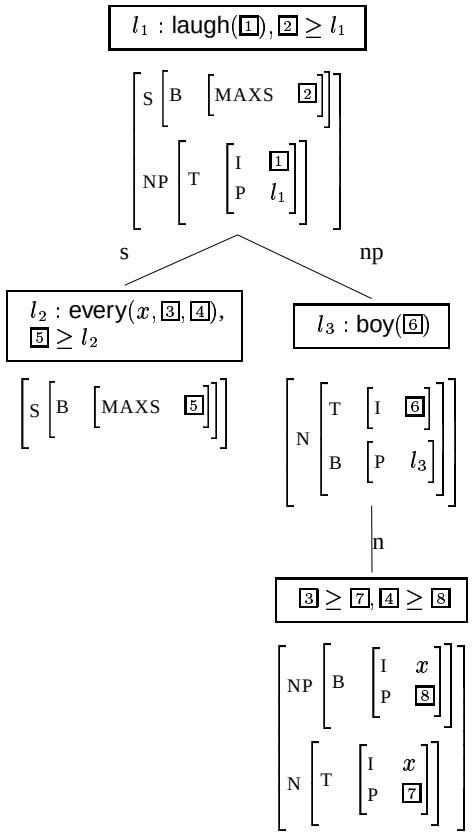


Figure 2: Semantics of (5) *Every boy laughs*.

3 Quantifiers

Following Joshi and Vijay-Shanker (1999); Kallmeyer and Joshi (2003) and in particular Romero et al. (2004), we assume that quantificational NPs as *every* in (5) and also *who* in (6) have a multicomponent set containing an auxiliary tree that contributes the scope part and a second elementary tree that contributes the predicate argument part.

(5) Every boy laughs.

(6) Who laughs?

However, in contrast to preceding approaches, we assume the predicate argument tree for quantifiers that are determiners as *every* in (5) to be an auxiliary tree. In other words, we assume determiners to be adjoined to their nouns. This corresponds to a standard analysis as pursued in the XTAG grammar (XTAG Research Group, 1998) and also in the French LTAG (Abeillé, 2002) for example. With semantic unification, this approach is possible since the NP tree can be linked to the verb tree via feature unification although there is no direct link in the derivation tree. An example is shown in Fig. 1 and 2.

The derivation in Fig. 1 seems non-local because the two components of the quantifier attach to different elementary trees. This apparent non-locality is however no problem: First, we allow this kind of non-local adjunc-

[2] For the sake of readability, we use names np, vp, ... for the node positions instead of the usual Gorn adresses.

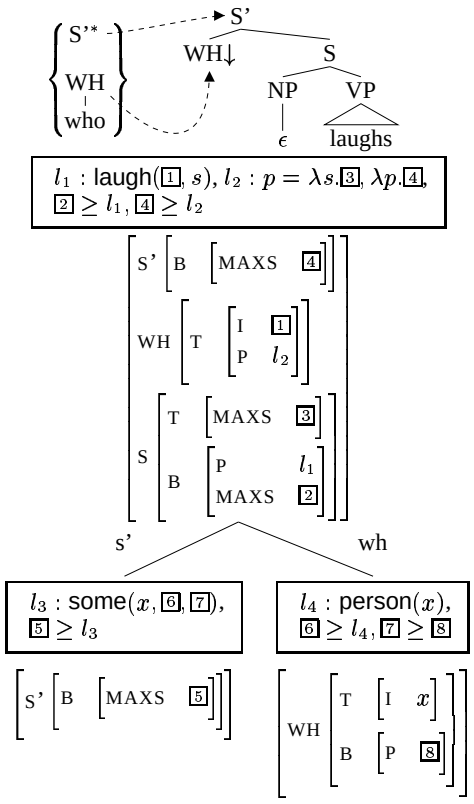


Figure 3: Syntax and semantics of (6) *Who laughs?*

tion only for scope trees, i.e., trees with just one single S node, and therefore the strong generative capacity of the grammar is not affected. Second, this derivation can also be understood in a local way: If we adopt flexible composition (Joshi et al., 2003), then we can consider the combination of *every* and *boy* as a wrapping of *boy* around *every*. The result is a derived *every* multicomponent that is then attached to *laughs*. Viewed in this way, the derivation is local. Such a non-local flexible composition analysis for the scope parts of quantifiers has already been proposed in Joshi et al. (2003) in order to derive certain constraints for relative quantifier scope in inverse linking configurations. In other words, there is independent motivation for this analysis.

The derivation tree with the semantic representations and the semantic feature structure for (5) is shown in Fig. 2. The unifications lead to the following feature identities: $\boxed{5} = \boxed{2}$ (adjunction of the scope part), $\boxed{1} = \boxed{6}$ (substitution of *boy* into *laughs*), $\boxed{6} = x$ and $\boxed{8} = l_1$ (adjunction of determiner to *boy* and final top-bottom unification at NP node), and $\boxed{7} = l_3$ (adjunction of *every* to *boy* and final top-bottom unification at N node). Replacing the variables by their values and building then the union of all semantic representations leads to (7):

(7) $l_1 : \mathsf{laugh}(x),\ l_2 : \mathsf{every}(x, \boxed{3}, \boxed{4}),\ l_3 : \mathsf{boy}(x)$
$\boxed{2} \geq l_1, \boxed{2} \geq l_2, \boxed{3} \geq l_3, \boxed{4} \geq l_1$

There is only one disambiguation, namely $\boxed{2} \rightarrow l_2, \boxed{3} \rightarrow l_3, \boxed{4} \rightarrow l_1$, that leads to the semantics $\mathsf{every}(x, \mathsf{boy}(x), \mathsf{laugh}(x))$.

The feature *maximal scope* (MAXS) is needed to provide the correct maximal scope of quantifiers. This is important in questions (see below). Furthermore, MAXS is also used to make sure that quantifiers embedded under attitude verbs such as *think* cannot scope over the embedding verb. This constraint is largely assumed to hold for quantifiers (see Kallmeyer and Romero, 2004, for further discussion).

Following Romero et al. (2004), we assume that wh-operators are similar to quantifiers in the sense that they also have a separate scope part and they also have a MAXS scope limit. But their scope limit is provided by the S' node, not the S node. For an analysis of (6), see Fig. 3. The MAXS features together with the semantics of the question verb make sure that all wh-operators have scope over the question proposition (here l_2) and all quantifiers scope below this proposition. The minimal nuclear scope of the wh-operator (variable $\boxed{8}$) is provided by the question proposition l_2.

4 Stranding of Prepositions

Syntactically, the stranding examples in (2) are more complex than the pied piping examples in (1). Therefore we consider them first for developing our syntactic analysis.

A multicomponent analysis as proposed in (Kroch, 1989) that puts the wh-word (*whom* in (2a)) and the stranded part (*a picture of* in (2a)) into one elementary tree set is not acceptable since this would violate the principle of minimality of elementary trees: In LTAG, elementary trees represent extended projections of lexical items and encapsulate all syntactic/semantic arguments of the lexical anchor. They are minimal in the sense that only the arguments of the anchor are encapsulated, all recursion is factored away. These linguistic properties of elementary trees are formulated in the *Condition on Elementary Tree Minimality (CETM)* from Frank (1992). Even a separation of *whom* and *a picture of* into just two different elementary trees or tree sets would violate this principle. Therefore, we need at least three different elementary trees (or tree sets) for *whom*, *a* and *picture of*.

There are essentially two possible syntactic analyses for sentences such as (2a): the embedded PP could be seen as a modifier or a complement of the higher NP. In the first case, we would assume an extra elementary tree

for *of*, in the second case *picture of* would not be separated. (Kroch, 1989) shows that only a complement analysis can account for the reported ungrammaticality of (8). Thus, we propose the syntactic structure in Fig. 4 for (2a).

(8) *Where did you meet a friend from?

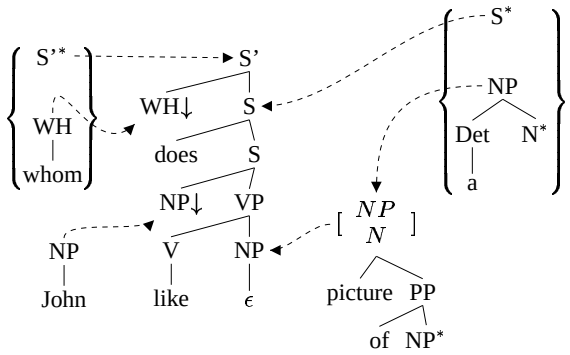


Figure 4: Syntactic Analysis for (2a).

As noted above, the non-local attachment of the multi-component set for *a* does not affect the complexity of the grammar significantly, as one of the components is a degenerate tree. If one wishes to avoid such an attachment, the derivation can alternatively be seen as a case of flexible composition: *picture_of* first attaches flexibly to the lower component of *a*, and the derived *a*-tree set then attaches to the tree of *like*. The lexicon entries and the semantic composition that we give below does not depend on one particular of these syntactic analysis, that may therefore be chosen for independent, syntactic reasons.

A completely different analysis of long extractions in TAG that has been proposed in (Kahane et al., 2000) and further pursued for semantics in (Kallmeyer, 2003) is the possibility to start from the wh-word, to adjoin first all material inside the NP that embeds the wh-word and then adjoin the main verb of the question. This works for pied-piping and for stranding cases. However, it means departing considerably from TAG standard analyses for questions and relative clauses, a step that we would like to avoid. The analyses we propose in this paper are consistent with the proposals made so far for simple questions and relative clauses (see Kroch, 1987; Abeillé, 2002).

The semantic derivation of (2a) corresponding to the proposed syntactic analysis is given in Fig. 5. In this sentence, the second participant in the verbal semantics does not come directly from the wh-phrase. In contrast, it is provided by the embedded PP. We therefore propose the following in order to allow intervening PPs: instead of passing the argument variable from the wh-NP directly to the verb, it is passed to the bottom feature of the empty NP (node address np2). The verb's argument comes from the top feature structure of that NP. So if nothing adjoins to the empty NP, the wh-NP variable will be passed up as the argument. In our case, however, another individual variable intervenes and becomes the argument.

The feature identities from the semantic computation of (2a) are $\boxed{8} = \boxed{5}$, $\boxed{11} = l_2$, $\boxed{2} = x$, $\boxed{1} = y$, $\boxed{14} = \boxed{4} = \boxed{7}$, $\boxed{16} = \boxed{2} = x$, $\boxed{19} = l_1$, $\boxed{6} = z$, $\boxed{15} = z$, $\boxed{18} = l_6$. This leads to the semantic representation (9):

(9)
$$\begin{array}{l}\mathsf{John}(y),\ l_1 : \mathsf{like}(y,z,s),\ l_2 : p = \lambda s.\boxed{7}, \lambda p.\boxed{5},\\ l_3 : \mathsf{some}(x,\boxed{9},\boxed{10}),\ l_4 : \mathsf{person}(x),\\ l_5 : \mathsf{a}(z,\boxed{12},\boxed{13}),\ l_6 : \mathsf{picture\_of}(z,x),\\ \boxed{7} \geq l_1, \boxed{5} \geq l_2, \boxed{5} \geq l_3, \boxed{9} \geq l_4, \boxed{10} \geq l_2,\\ \boxed{7} \geq l_5, \boxed{12} \geq l_6, \boxed{13} \geq l_1\end{array}$$

There is one single disambiguation, namely $\boxed{5} \rightarrow l_3$, $\boxed{9} \rightarrow l_4$, $\boxed{10} \rightarrow l_2$, $\boxed{7} \rightarrow l_5$, $\boxed{12} \rightarrow l_6$, $\boxed{13} \rightarrow l_1$ which leads to $\lambda p.\mathsf{some}(x, \mathsf{person}(x), p = \lambda s.\mathsf{a}(z, \mathsf{picture\_of}(z,x), \mathsf{like}(y,z,s)))$ for the question.

5 Pied-Piping

With the same elementary trees and the same semantic representations, pied-piping constructions as (1a) can be analysed. A derivation of that sentence can be found in Fig. 6.[3]

The only additional modification we have to make is a distinction between the minimal nuclear scope of non-wh quantifiers and the minimal nuclear scope of wh quantifiers, since in (1a), both have to come from the same node (the wh-NP).[4] We continue to use the feature P for the first, and introduce a feature WP for the second. Of course, this does not affect the analysis in Fig. 5. The semantic derivation in Fig. 6 proceeds exactly parallel, with all the same feature identities as in Fig. 5, except for the value of $\boxed{2}$: here, $\boxed{2} = z$. But $\boxed{2}$ does not occur in the semantic representations, only in the feature structures. Therefore, the resulting semantics is the same.

6 Genitives

Another possible type of pied-piping sentences are those with possessive pre-nominal modifiers, such as (1b), or (10):

(10) Whose house did you see?

(Han, 2002) discusses a TAG analysis of relative clauses with complex wh-phrases such as (11) and (12):

(11) the problem whose solution is difficult

(12) the problem whose solution's proof is difficult

[3] We left out the attachments of the scope parts and of *John* in Fig. 6 because they proceed exactly as in Fig. 5.

[4] This distinction is also necessary for in-situ wh-words where the minimal nuclear scope of the wh-quantifier comes from the lower NP, see Romero et al. (2004).

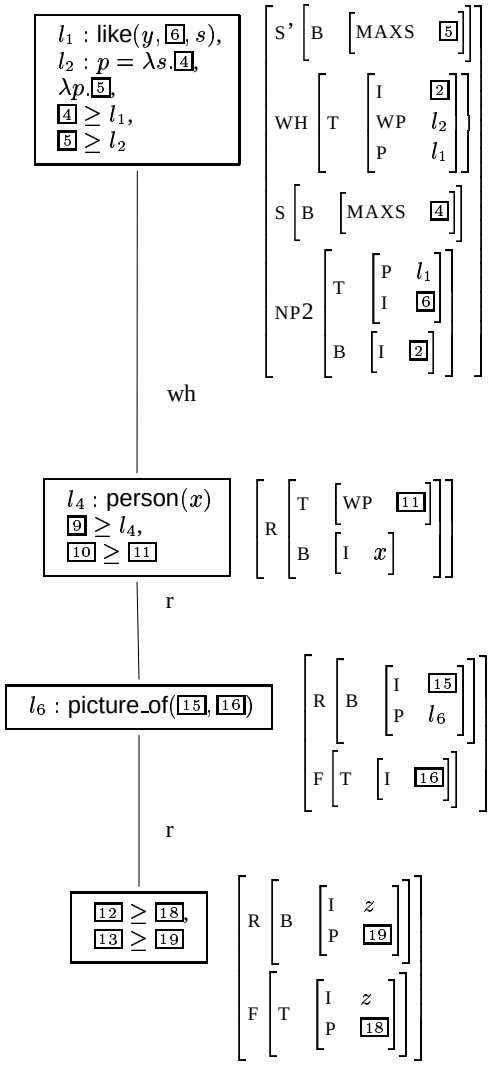


Figure 6: Abridged semantic derivation for (1a).

The structure of these relative clauses is almost identical to our questions above, so solutions to the relative clause problem will carry over to the direct questions.

For the syntax, Han proposes, similar to our treatment for *a picture of* above, a different lexical entry for the genitive *'s* (and *se* respectively), a predicative auxiliary tree where the outer NP adjoins into the embedded wh-phrase.

In order to get the correct variable bindings, Han makes use of a complex LINK predicate, which effectively introduces a separate semantic variable for the item that is possessed, and the one that is the possessor (the wh-phrase), which both have to be unified with variables in the embedding phrase. The use of underspecified feature structures allows for a simpler representation. The elementary tree for *'s* is given in Fig. 7, along with an appropriate semantics.[5] The semantic feature structure ensures that the

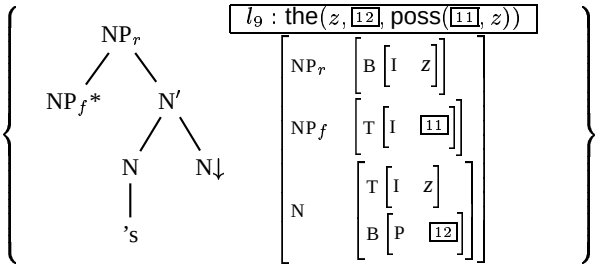


Figure 7: Lexical entry for *'s*.

correct individual variable [I z] is handed upwards in the tree, so that predicates such as *see* will only have access to this variable. On the other hand, the wh-phrase's own variable is passed downwards (which becomes relevant if the genitive adjoins into a real phrase like *which boy* — then the wh's variable is needed for the predicate *boy*).

The syntactic derivation of an embedded genitive question like (10) using this lexical entry is found in Fig. 8.

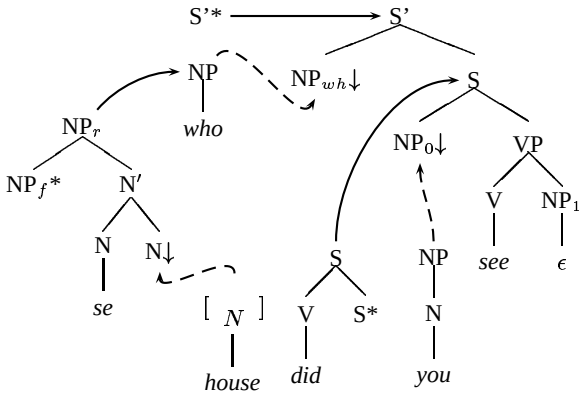


Figure 8: Syntax of (10) *Whose house did you see?*

The elementary tree for the possessive adjoins into the root node of the initial tree for *who*. It has no scopal effects, so the scopal properties of simple questions are kept. In particular, the question word continues to have the widest possible scope.

Fig. 9 shows the semantic derivation of the sentence (10). The feature identities from the semantic computation are 1 = 7, 4 = z = 13, 2 = 5, 6 = l_1, 3 = x, 10 = l_2, 11 = y, 12 = l_6. This leads to the semantic representation (13):

[5] We modified the semantic representation Han gave to fit with our formalism and notation.

(13) $\mathsf{you}(x), l_1 : \mathsf{see}(x, z, s), \lambda p.\boxed{1}, l_2 : p = \lambda s.\boxed{2}, l_3 : \mathsf{some}(y, \boxed{8}, \boxed{9}), l_4 : \mathsf{person}(y), l_5 : \mathsf{the}(z, l_6, \mathsf{poss}(y, z)), l_6 : \mathsf{house}(z), \boxed{2} \geq l_1, \boxed{1} \geq l_2, \boxed{1} \geq l_3, \boxed{8} \geq l_4, \boxed{9} \geq l_2$

There is one disambiguation, namely $\boxed{1} \rightarrow l_3$, $\boxed{8} \rightarrow l_4$, $\boxed{9} \rightarrow l_2$, $\boxed{2} \rightarrow l_1$. This results in the semantics $\lambda p.\mathsf{some}(y, \mathsf{person}(y), p = \lambda s.\mathsf{see}(x, z, s) \wedge \mathsf{you}(x) \wedge \mathsf{the}(z, \mathsf{house}(z), \mathsf{poss}(y, z)))$ for question (10).

7 Conclusion

This paper proposes an analysis for stranding and pied-piping of wh-phrases that takes into account syntax and semantics of these constructions. As mentioned above, most previous approaches dealing with these data have only considered syntactic aspects. They are problematic since they violate the Condition on Elementary Tree Minimality (CETM). Those analyses that respect the CETM and that lead to a suitable semantics depart considerably from standard LTAG analyses for questions and relative clauses. This is not the case for the analysis proposed here: we have shown that we obtain syntactic analyses that extend the standard analyses and that allow to derive adequate semantic representations for the data in question. The proposed analysis is such that stranding and pied-piping constructions are treated in parallel, i.e., with the same elementary trees.

Acknowledgments

For many fruitful discussions of the analyses this paper proposes we would like to thank Olga Babko-Malaya, Aravind K. Joshi, Maribel Romero and all members of the XTAG Group at the University of Pennsylvania. Furthermore, we are indebted to two anonymous reviewers for their helpful comments.

References

Anne Abeillé. 2002. *Une grammaire électronique du français*. CNRS Editions, Paris.

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 1999. Minimal Recursion Semantics. An Introduction. Draft, Stanford University.

Robert Fiengo and James Higginbotham. 1981. Opacity in NP. In *Linguistic Analysis*, 7(4):395–421.

Robert Frank. 1992. *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives*. Ph.D. thesis, University of Pennsylvania.

Claire Gardent and Laura Kallmeyer. 2003. Semantic construction in Feature-Based TAG. In *Proceedings of the 10th EACL*, Budapest, Hungary.

Chung-hye Han. 2002. Compositional Semantics for Relative Clauses in Lexicalized Tree Adjoining Grammars. In *Proceedings of TAG+7*, pages 101–110, Venice, Italy.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible composition in LTAG, quantifier scope and inverse linking. In *Proceedings of the 5th IWCS*, pages 179–194, Tilburg, NL.

Aravind K. Joshi and K. Vijay-Shanker. 1999. Compositional Semantics with Lexicalized Tree-Adjoining Grammar (LTAG): How Much Underspecification is Necessary? In H. C. Blunt and E. G. C. Thijsse, editors, *Proceedings of the Third International Workshop on Computational Semantics (IWCS-3)*, pages 131–145, Tilburg.

Sylvain Kahane, Marie-Hélène Candito, and Yannick de Kercadio. 2000. An alternative description of extraction in TAG. In *Proceedings of TAG+5*, pages 115–122, Paris, France.

Laura Kallmeyer. 2003. LTAG Semantics for Relative Clauses. In Harry Bunt, Ielka van der Sluis, and Roser Morante, editors, *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*, pages 195–210, Tilburg, NL.

Laura Kallmeyer and Aravind K. Joshi. 2003. Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation*, 1(1-2):3–58.

Laura Kallmeyer and Maribel Romero. 2004. LTAG Semantics with Semantic Unification. In *Proceedings of TAG+7*, Vancouver, Canada.

Anthony S. Kroch. 1987. Unbounded dependencies and subjacency in a Tree Adjoining Grammar. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 143–172. John Benjamins, Amsterdam.

Anthony Kroch. 1989. Asymmetries in long-distance extraction in a Tree Adjoining Grammar. In Baltin and Kroch, editors, *Alternative Conceptions of Phrase Structure*. University of Chicago.

Maribel Romero, Laura Kallmeyer, and Olga Babko-Malaya. 2004. LTAG Semantics for Questions. In *Proceedings of TAG+7*, Vancouver, Canada.

XTAG Research Group. 1998. A Lexicalized Tree Adjoining Grammar for English. Technical Report 98–18, Institute for Research in Cognitive Science, Philadelphia.

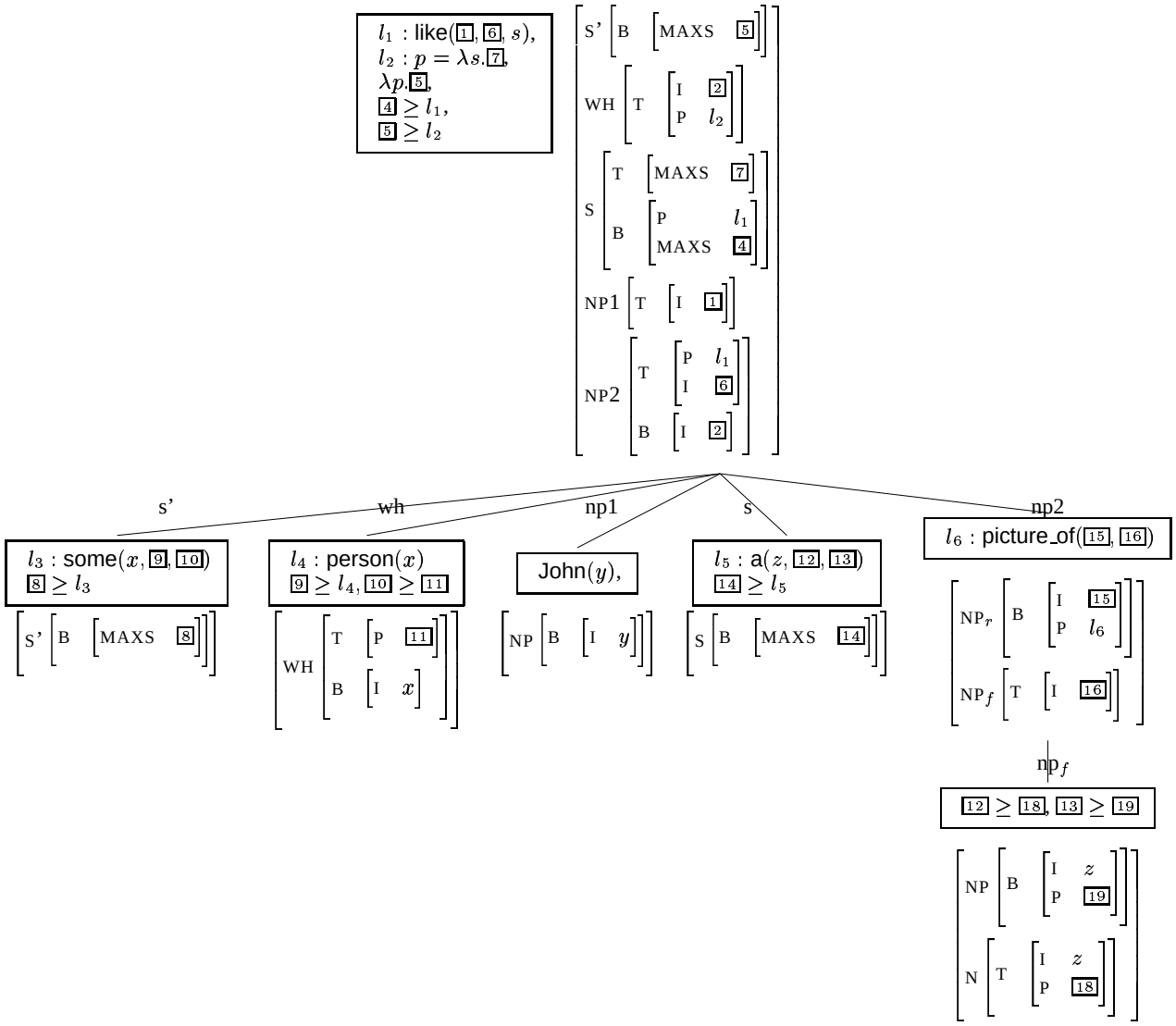


Figure 5: Semantic Derivation of (2a) *Whom does John like a picture of?*

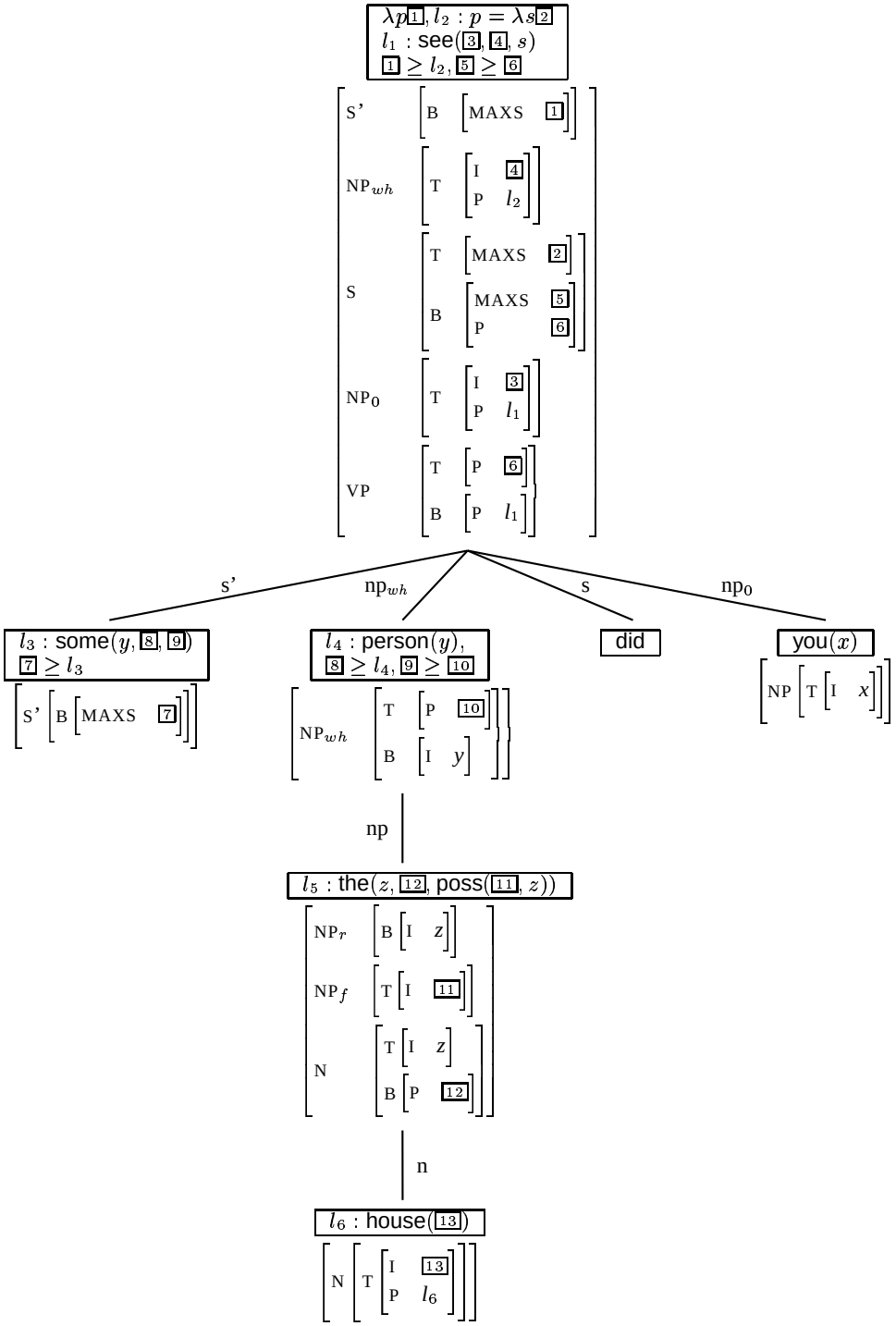


Figure 9: Semantic derivation of (10) *Whose house did you see?*

Tree-local MCTAG with Shared Nodes: Word Order Variation in German and Korean

Laura Kallmeyer **SinWon Yoon**
UFRL, University Paris 7
2 place Jussieu, Case 7003, 75251 Paris Cedex 05
{laura.kallmeyer,swyoon}@linguist.jussieu.fr

Abstract

Tree Adjoining Grammars (TAG) are known not to be powerful enough to deal with scrambling in free word order languages. The TAG-variants proposed so far in order to account for scrambling are not entirely satisfying. Therefore, an alternative extension of TAG is introduced based on the notion of node sharing. Considering data from German and Korean, it is shown that this TAG-extension can adequately analyse scrambling data, also in combination with extraposition and topicalization.

1 Introduction

1.1 LTAG and scrambling

Lexicalized Tree Adjoining Grammars (LTAG, (Joshi and Schabes, 1997)) is a tree-rewriting formalism. An LTAG consists of a finite set of trees (elementary trees) associated with lexical items. Larger trees are derived by substitution (replacing a leaf with a new tree) and adjunction (replacing an internal node with a new tree). LTAG elementary trees represent extended projections of lexical items and encapsulate all syntactic arguments of the lexical anchor. They are minimal in the sense that only the arguments of the anchor are encapsulated, all recursion is factored away.

Roughly, *scrambling* is the permutation of elements (arguments and adjuncts) of a sentence (we use the term *scrambling* in a purely descriptive sense without implying any theory of movement). A special case is *long-distance* scrambling where arguments or adjuncts of an embedded infinitive are 'moved' out of the embedded VP. This occurs for instance in languages such as German, Hindi, Japanese and Korean. These languages are therefore often said to have a free word order. Consider for example the German sentence (1). In (1), the accusative NP *es* is an argument of the embedded infinitive *zu reparieren* but it precedes *der Mechaniker*, the subject of the main verb *verspricht* and it is not part of the embedded VP. It has been argued that in German there is no bound on the number of scrambled elements and no bound on the depth of scrambling (i.e., in terms of movement, the number of VP borders crossed by the moved element). (See for example (Rambow, 1994a; Meurers, 2000; Müller, 2002) for descriptions of scrambling data.)

(1) ... dass [es]$_1$ der Mechaniker [t_1 zu reparieren] verspricht
... that it the mechanic to repair promises
'... that the mechanic promises to repair it'

As shown in (Becker et al., 1991), TAG are not powerful enough to describe scrambling in German in an adequate way. By this we mean that a TAG analysis of scrambling with the correct predicate-argument structure is not possible, i.e., an analysis with each argument attaching to the verb it depends on.

Let us consider the analysis of (1) in order to get an idea of why scrambling poses a problem for TAG. If we leave aside the complementizer *dass*, elementary trees for *verspricht* and *reparieren* might look as shown in Fig. 1. In the derivation, the *verspricht*-tree adjoins to the root of the *reparieren*-tree and the NP *der Mechaniker* is substituted for the subject node of *verspricht*.[1] This leads to the third tree in Fig. 1. When adding *es*, there is a problem: it should be added to *reparieren* since it is one of its arguments. But at the same time, it should precede *Mechaniker*, i.e., it must be adjoined either to the root or to the NP$_{nom}$ node in the derived tree. The root node belongs to *verspricht* and the NP$_{nom}$ node belongs to *Mechaniker*. Consequently, an adjunction to one of them would not give the desired predicate-argument structure. If it was only for (1), one could add a tree to the grammar

[1] The fact that *der Mechaniker* is at the same time logical subject of *reparieren* is accounted for in the semantics, see for example (Gardent and Kallmeyer, 2003).

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 40-47.

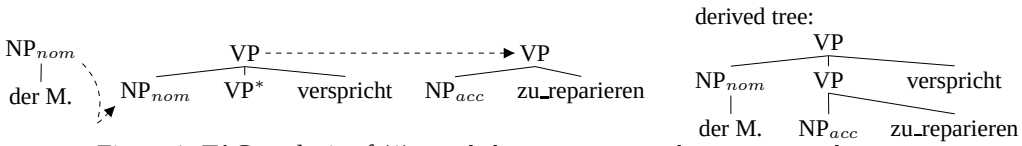


Figure 1: TAG analysis of (1) *dass* $[es]_1$ *der Mechaniker* $[t_1$ *zu reparieren*$]$ *verspricht*

for *reparieren* with a scrambled NP that allows adjunction of *verspricht* between the NP and the verb. But as soon as there are several scrambled elements that are arguments of different verbs, this does not work any longer. In general, it has been shown (Joshi et al., 2000) that adopting specific elementary trees it is possible to deal with a part of the difficult data: TAG can describe scrambling up to depth 2 (two crossed VP borders). But this is not sufficient. Even though examples of scrambling of depth > 2 are rare, they can occur (see Kulick, 2000).

1.2 TAG variants proposed for scrambling

The problem of long-distance scrambling and TAG is the fact that the trees representing the syntax of scrambled German subordinate clauses do not have the simple nested structure that ordinary TAG generates. In TAG, according to the Condition on Elementary Tree Minimality (CETM, (Frank, 1992)) (positions for) all of the arguments of the lexical anchor of an elementary tree are included in that tree. But in the scrambled tree the arguments of several verbs are interleaved freely. All TAG extensions that have been proposed to accommodate this interleaving involve factoring the elementary structures into multiple components and inserting these components at multiple positions in the course of the derivation.

One of the first proposals made was an analysis of German scrambling data using non-local MCTAG with additional dominance constraints (Becker et al., 1991). However, the formal properties of non-local MCTAG are not well understood and it is assumed that the formalism is not polynomially parsable. Therefore this approach is no longer pursued but it has influenced the different subsequent proposals.

An alternative formalism for scrambling is V-TAG (Rambow, 1994a; Rambow, 1994b; Rambow and Lee, 1994), a formalism that has nicer formal properties than non-local MCTAG. V-TAG also use multicomponent sets (so-called *vectors*) for scrambled elements, in this it is a variant of MCTAG. Additionally, there are dominance links between the trees of one vector. In contrast to MCTAG, the trees of a vector are not required to be added simultaneously. The lexicalized V-TAGs that are of interest for natural languages are polynomially parsable. Even though the formalism does not pose the problems of non-local MCTAG in terms of parsing complexity, it is still a non-local formalism in the sense that, as long as the dominance links are respected, arbitrary nodes can be chosen to attach the single components of a vector. This makes the formalism harder to understand than local TAG-variants since one needs a more gobal picture of what is going on in a derivation. Furthermore, in order to formulate certain locality restrictions (e.g., for wh-movement and also for scrambling), one needs an additional means to put constraints on what can interleave with the different trees of a vector or in other words constraints on how far a dominance link can be stretched. V-TAG allows to put *integrity constraints* on certain nodes that disallow these nodes to occur between two trees linked by a dominance link. This has the effect that these nodes act as barriers. This explicit marking of barriers is somewhat against the original appealing TAG idea that such constraints result from the CETM which imposes the position of the moved element and the verb it depends on to be in the same elementary structure, and from the further possibilities to combine this structure. In other words, in local formalisms with an extended domain of locality such as TAG or tree-local and set-local MCTAG such constraints result from the form of the elementary structures and the locality of the derivation.

D-tree substitution grammars (DSG, Rambow, Vijay-Shanker, and Weir, 2001) are another TAG-variant one could use for scrambling. DSG are a description-based formalism, i.e., the objects a DSG deals with are tree descriptions. A problem with DSG is that the expressive power of the formalism is probably too limited to deal with all natural language phenomena: according to (Rambow et al., 2001) it 'does not appear to be possible for DSG to generate the copy language". This means that the formalism is probably not able to describe cross-serial dependencies in Swiss German. Furthermore, DSG is non-local and therefore, as in the case of V-TAG, additional constraints (so-called *path constraints*) have to be put on material interleaving with the different parts of an elementary structure.

Another TAG-variant proposed in order to deal with scrambling are Segmented Tree Adjoining Grammars (SegTAG, Kulick, 2000). SegTAG can generate the copy language and therefore describe cross-serial dependencies. But the formalism uses a rather complex operation on trees, segmented adjunction, that consists partly of a standard TAG adjunction and partly of a kind of tree merging or tree unification. In this operation, two different things get mixed up, the more or less resource-sensitive adjoining operation of standard TAG where sub-

trees cannot be identified,[2] and the completely different unification operation. Furthermore, the formal properties of SegTAG are not clear. Kulick suggests that SegTAGs are probably in the class of LCFRS but there is no actual proof of this. However, if SegTAG is in LCFRS, the generative power of the formalism is probably too limited to deal with scrambling in a general way. In order to treat scrambling up to a certain depth, Kulick therefore allows certain extensions of SegTAG.

All these TAG variants are interesting with respect to scrambling and they give a lot of insight into what kind of structures are needed for scrambling. But, as explained above, none of them is entirely satisfying. The most convincing one is V-TAG since this formalism can deal with scrambling, it is polynomially parsable and the set of languages it generates contains the set TAL of all tree adjoining languages (in particular the copy language). But, as already mentioned, V-TAG has the inconvenient of being a non-local formalism. For the reasons explained above, it is desireable to find a local TAG extension for scrambling (as opposed to the non-locality of derivations in V-TAG, DSG and non-local MCTAG) such that locality constraints for movements follow only from the form of the elementary structures and from the local character of derivations. This paper proposes a local TAG-variant that can deal with scrambling, at least with an arbitrarily large set of scrambling phenomena, that is polynomially parsable and that properly extends TAG in the sense that TAL is a proper subset of the languages it generates.

In section 2, tree-local MC-TAG with shared nodes (SN-MCTAG) and in particular restricted SN-MCTAG (RSN-MCTAG) are introduced. Section 3 to 5 show the analyses of different word order variations using this formalism, namely scrambling, extraposition and topicalization, considering data from German and Korean.

2 Tree-local MCTAG with shared nodes (SN-MCTAG)

To illustrate the idea of shared nodes, consider again example (1). In standard TAG, nodes to which new elementary trees are adjoined or substituted disappear, i.e., they are replaced by the new elementary tree. E.g., after the derivation steps shown in Fig. 1, the root node of the *reparieren* tree does not exist any longer. It is replaced by the *verspricht* tree and its daughters have become daughters of the foot node of the *verspricht* tree. I.e., the root node of the derived tree is considered being part of only the *verspricht* tree. Therefore, an adjunction at that node is an adjunction at the *verspricht* tree. However, this standard TAG view is not completely justified: in the derived tree, the root node and the lower VP node might as well be considered as belonging to *reparieren* since they are results of identifying the root node of *reparieren* with the root and the foot node of *verspricht*.[3] Therefore, we propose that the two nodes in question belong to both, *verspricht* and *reparieren*. In other words, these nodes are shared by the two elementary trees. Consequently, they can be used to add new elementary trees to *verspricht* and (in contrast to standard TAG) also to *reparieren*.

We use a multicomponent TAG (MCTAG, Joshi, 1987; Weir, 1988). This means that the elements of the grammar are sets of elementary trees. In each derivation step, one of these sets is chosen and the trees in this set are added simultaneously (by adjunction or substitution) to different nodes in the already derived tree. We assume tree-locality, i.e., the nodes to which the trees of such a set are added must all belong to the same elementary tree. Standard tree-local MCTAGs are strongly equivalent to TAG but they allow to generate a richer set of derivation structures. In combination with shared nodes, tree-local multicomponent derivation extends the weak generative power of the grammar.

Let us go back to (1). Assume the tree set on the left of Fig. 2 for *es*. Adopting the idea of shared nodes, this tree set can be added to *reparieren* using the root of the already derived tree for adjunction of the first tree and the NP_{acc} node for substitution of the second tree. The operation is tree-local since both nodes are part of the *reparieren* tree.

In general, the notion of shared nodes means the following: When substituting an elementary tree α into an elementary tree γ, in the resulting tree, the root node of the subtree α is considered being part of α and of γ. When adjoining an elementary β at a node that is part of the elementary trees $\gamma_1, \ldots, \gamma_n$, then in the resulting tree, the root and foot node of β are both considered being part of $\gamma_1, \ldots, \gamma_n$ and β. Consequently, if an elementary γ' is added to an elementary γ and if there is then a sequence of adjunctions at root or foot nodes starting from γ', then each of these adjunctions can be considered as an adjunction at γ since it takes place at a node shared by γ, γ' and all the subsequently adjoined trees. In Fig. 2 for example the *es*-tree is adjoined to the root of a tree that was adjoined to *reparieren*. Therefore this adjunction can be

[2] More precisely, only the root of the new elementary tree and eventually (i.e., in case of an adjunction) the foot node get identified with the node the new tree attaches to. But there is no unification of whole subtrees.

[3] Actually, in a Feature-Structure Based TAG (FTAG, (Vijay-Shanker and Joshi, 1988)), the top feature structure of the root of the derived tree is the unification of the top of the root of *verspricht* and the top of the root of *reparieren*. The bottom feature structure of the lower VP node is the unification of the bottom of the foot of *verspricht* and the bottom of the root of *reparieren*. In this sense, the root of the *reparieren* tree gets split into two parts. The upper part merges with the root node of the *verspricht* tree and the lower part merges with the foot node of the *verspricht* tree.

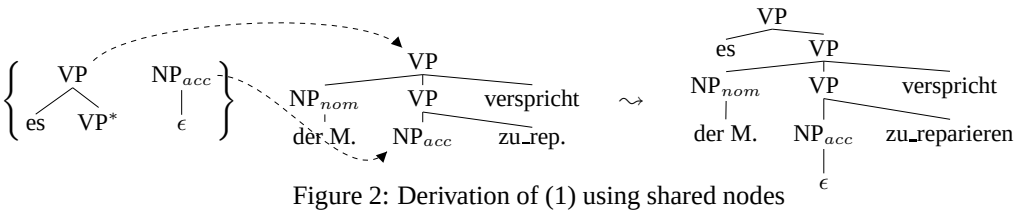


Figure 2: Derivation of (1) using shared nodes

considered being an adjunction at *reparieren*. An adjunction at a node where other trees already have been added (e.g., this adjunction of *es* to the root of *reparieren*) is called a *secondary* adjunction while a first adjunction at a node is called a *primary* adjunction.

Concerning formal properties, SN-MCTAG is hard to compare to other local TAG-related formalisms since arbitrarily many trees can be added by secondary adjunction to a single elementary tree. Therefore, we define a restricted version, *restricted SN-MCTAG (RSN-MCTAG)* that limits the number of secondary adjunctions to an elementary tree by allowing secondary adjunction only in combination with at least one simultaneous primary adjunction or substitution. E.g., in Fig. 2, *es* is secondarily adjoined to *reparieren* while the second element of the tree set is primarily added (substituted) to *reparieren*.

Obviously, all tree adjoining languages can be generated by RSN-MCTAGs since a TAG is an MCTAG with unary multicomponent sets. It can be shown that for each RSN-MCTAG of a specific type, an equivalent simple Range Concatenation Grammars (RCG, (Boullier, 1998; Boullier, 1999)) and therefore an equivalent LCFRSs (linear context-free rewriting systems, (Weir, 1988)) can be constructed. LCFRSs are mildly context-sensitive and in particular polynomially parsable and therefore, this also holds for these specific RSN-MCTAGs. For a formal definition of SN-MCTAG and RSN-MCTAG and a sketch of the proof of the mildly context-sensitivity see (Kallmeyer, 2004). The additional restriction imposed on RSN-MCTAG in order to obtain the equivalence to LCFRS puts a limit on the complexity of the scrambling data one can analyze. This limit however is variable in the sense that an arbitrarily large limit can be chosen. Consequently, based on empirical studies, the limit can be chosen such that all scrambling data are covered that are assumed to occur in real texts. In this respect, RSN-MCTAG differs crucially from TAG where the limit is fixed (scrambling up to depth 2 can be described and nothing more). In this sense one can say that RSN-MCTAG can analyze scrambling in general since it can anlyze any arbitrarily large finite set of scrambling data.

There are mainly two crucial differences between SN-MCTAG and V-TAG: firstly, in V-TAG the adjunctions of auxiliary trees from the same set are not required to be simutaneously. In this respect, V-TAG differs from standard MCTAG in general. Secondly, V-TAG is non-local in the sense of non-local MCTAG while RSN-MCTAG is local, even though the locality is not based on the parent relation in the TAG derivation tree as it is the case in standard local MCTAG. As a consequence of the locality, in contrast to other TAG variants for scrambling, we do not need dominance links in RSN-MCTAG. The locality condition put on the derivation sufficiently constrains the possibilities for attaching the trees from elementary tree sets: different trees from a tree set attach to different nodes of the same elemenary tree, so the dominance relations between these different nodes are crucial for the dominance relation between the different trees from the tree set. Because of this dominance links are not necessary. This is different of course for non-local TAG-variants such as V-TAG or DSG where one can in principle attach the different components of an elementary structure at arbitrary nodes in the derived tree.

3 Scrambling

In many SOV languages, such as German, Hindi, Japanese and Korean, constituents (argument or adjunct) display a larger freedom in term of ordering in clauses. This phenomenon is called *scrambling*. (See (Uszkoreit, 1987) for a description of word order in German and (Lee, 1993) for Korean.) The constituents of the lower clause can even occur in the upper clause, (so-called *long distance* scrambling). E.g., the arguments *es* and *jadoncha-lul* of the embedded verb move into the upper clause in German (1), repeated as (2)a., and in the Korean sentence (2)b.

(2) a. ... dass es$_1$ der Mechaniker [t_1 zu reparieren] verspricht

b. jadoncha-lul$_1$ keu-ka [t_1 surihakess-tako]
the car$_{acc}$ he$_{nom}$ [t_1 repair-to]
yaksokhaessta
promises
'He promises to repair the car'

Generally, in both languages, it is assumed that there is no bound on the number of elements that can scramble in one sentence, and there is no bound on the distance over which each element can scramble. In the following we will show how RSN-MCTAG allows to deal with long distance scrambling. Elementary trees for word order variations of (3) are shown in Fig. 3. We propose

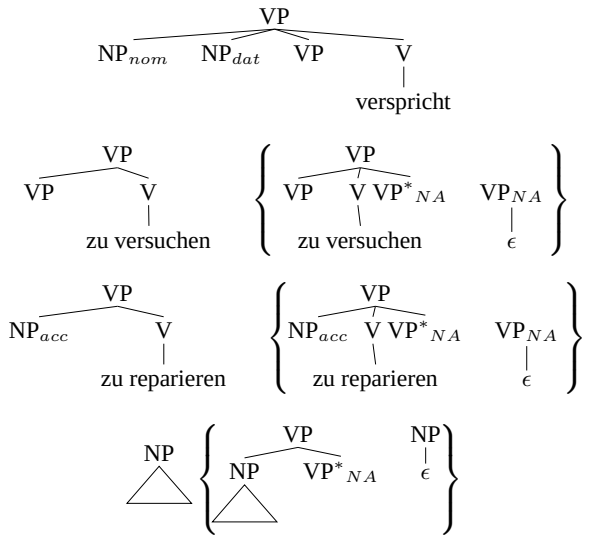


Figure 3: Elementary trees for word order variations of (3) *... dass er dem Kunden* [[*das Auto zu reparieren*] *zu versuchen*] *verspricht*

single trees for non-scrambled elements, and tree sets for scrambled elements.

(3) ... dass er dem Kunden [[das Auto zu reparieren]
... that he$_{nom}$ the customer$_{dat}$ the car$_{acc}$ to repair
zu versuchen] verspricht
to try promises
'... that he promises the customer to try to repair the car'

(4) ... dass er das Auto$_1$ dem Kunden [[t_1 zu reparieren] zu versuchen] verspricht

Consider (4) where the most deeply embedded NP$_{acc}$ *das Auto* is scrambled into the upper clause. For *das Auto*, the tree set is used. Further, we also use tree sets for the NP$_{dat}$ *dem Kunden* which intervenes between the scrambled argument and its clause, and for the VP clause *reparieren* of witch argument is scrambled out over a clause of depth ≥ 2. For the non-scrambled NP$_{nom}$ *er*, and for the non-scrambled VP *versuchen*, single trees are used. Fig. 4 shows the different derivation steps for (4). First, *verspricht* and *versuchen* are combined by substitution. In the resulting derived tree (on the right on top of the figure), the bold VP node is now shared by *verspricht* and *versuchen*. Then the auxiliary tree in the tree set for *reparieren* adjoins to the shared node. This is a primary adjunction at *versuchen*. The initial tree is substituted for the VP leaf of *versuchen*. The former root node of the *reparieren* auxiliary tree, i.e., the bold VP node in the tree in the middle of the bottom of the figure, is now shared by *verspricht*, *versuchen* and *reparieren*. The next *secondary* adjunctions can occur at this new shared node: *dem Kunden* is added as sketched in the figure, and then *das Auto* is added in the same way. The tree for *er* is added into the substitution slot in the *verspricht* tree.

Note that a scrambled elements always adjoins to a VP node and the scrambled element is to the left of the foot node. Therefore it precedes everything that is below or on the right of the VP node to which it adjoins. Consequently, given the form of the verbal elementary trees in Fig. 3 where the verb is always below or right of all VP nodes allowing adjunction, the order $x\ v$ for an x being a nominal or a verbal argument of v is always respected.

Since all scrambled elements attach to a VP node in the elementary tree of the verb they depend on, they cannot attach to the VP of a higher finite verb that embeds the sentence in which the scrambling occurs. Therefore, this analysis correctly predicts that scrambling can never proceed out of tensed clauses. In other words, a barrier effect is obtained without posing any explicit barrier as it is done in V-TAG. Instead, the locality of scrambling is a consequence of the form of the elementary trees and of the locality of the derivations.

In contrast to German, Korean allows scrambling out of a tensed clause. For example, in (5) the argument *jadoncha-lul* is scrambled out of a tensed clause. This difference can be captured by using in Korean the node label S instead of VP for the root and the foot node in the auxiliary trees for scrambling.[4]

(5) jadoncha-lul$_1$ keu-ka [kokaek-i t_1
the car$_{acc}$ he$_{nom}$ [the customer$_{nom}$ t_1
kuiphaess-tako] malhaessta.
buy-that] said
'He said that the customer bought the car'

4 Extraposition

In German and Korean, clausal arguments can optionally appear behind the finite verb. This is called *extraposition*. E.g., in (6), the *reparieren* VP occurs behind the finite verb *verspricht*. The same goes for the Korean extraposition (7).

(6) ... dass er$_{nom}$ dem Kunden$_{dat}$ t_1 verspricht, [das Auto$_{acc}$ zu reparieren]$_1$
'... that he promises the customer to repair the car'

(7) keu-ka$_{nom}$ kokaek-ekey$_{dat}$ t_1 yaksokhassta, [jadoncha-lul$_{acc}$ surihakess -tako]$_1$
'He promises the customer to repair the car'

[4] One aspect we did not consider in this paper but that definitely needs to be spelled out is the fact that in both languages, German and Korean, not all verbs allow scrambling to the same degree. In German, this is related to the difference between obligatorily and optionally coherent verbs (see (Meurers, 2000; Müller, 2002)). These facts probably can be modelled using specific features that control the scrambling possibilities of a verb.

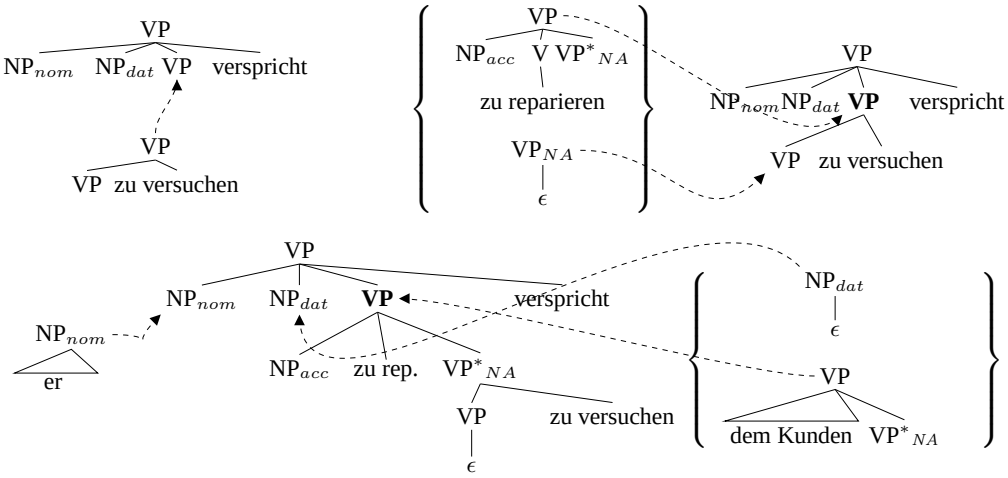


Figure 4: Derivation for (4) ... *dass er das Auto dem Kunden zu reparieren zu versuchen verspricht*

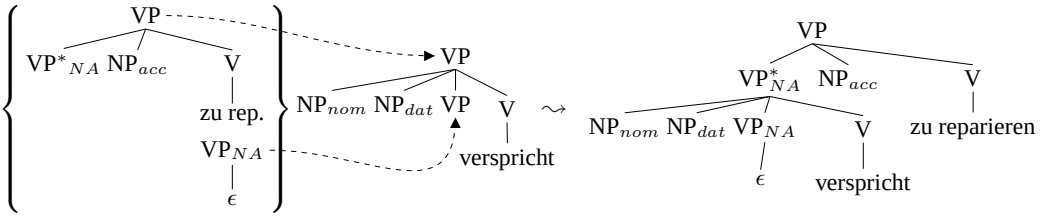


Figure 5: Derivation for (6) ... *dass er dem Kunden verspricht, das Auto zu reparieren*

Extraposition is doubly unbounded, as it is the case for *scrambling*. In order to analyze *extraposition*, we propose tree sets as the one for *reparieren* in Fig. 5. They resemble to those for *scrambling* except that the foot node is on the left because the extraposed material goes to the right of the finite verb. For the NP arguments in (6), we use the single trees shown in Fig. 3. The derivation for (6) is as sketched in Fig. 5.

The following differences between German and Korean are observed: both languages allow extraposition of complete VPs. Furthermore, in German, infinitives without their arguments can be extraposed (so-called *third construction*, see (8)a), which is not possible in Korean (see (9)a). In Korean however, arguments of embedded verbs can be extraposed while leaving their verb behind (see (9)b), which is not possible in German (see (8)b).[5]

(8) a. ... dass er es t_1 verspricht, [zu reparieren]$_1$

b. $*$... dass er [t_1 zu reparieren] verspricht, [es]$_1$

(9) a. $*$keu-ka$_{nom}$ jadoncha-lul$_{acc}$ t_1 yaksokhassta, [surihakess-tako]$_1$

b. keu-ka$_{nom}$ [t_1 surihakess-tako] yaksokhassta, [jadoncha-lul$_{acc}$]$_1$

[5]For this reason, Korean extraposition is often called *right-forward scrambling*.

To account for the difference between (8a) and (9a), we disallow the adjunction of scrambled elements at the root nodes of Korean auxiliary extraposition trees.[6] For (9b), in Korean, we propose additional tree sets for extraposed NPs. They are similar to the tree sets for scrambled NPs in Fig. 3, except that the foot node is on the left. Such tree sets do not exist in German.

5 Topicalization

Korean *topicalization* is realized with the topic marker *-nun*(*-un*). The topicalized constituent has to appear in the beginning of clauses, e.g., *jadoncha-nun* in (10a.): an element marked by *-nun*(*-un*) can also appear in sentence medial position e.g., *jadoncha-nun* in (10b.). It is perceived, in Korean, that an element with *-nun*(*-un*) in sentence initial position receives the theme reading, i.e., *topicalization*, and the counterpart in sentence medial position the contrastive reading. To describe *topicalization* movement, a topic argument may be inserted into the verbal projection tree at [Spec, CP] (see, e.g., (Suh, 2002)).

[6]In German, even arguments of embedded VPs can be left behind as in ... *dass er [es]$_1$ verspricht, [[t_1 zu reparieren] zu versuchen]*. For such cases, we propose an additional VP node on the spine of extraposed infinitives where deeper embedded infinitives can be added. For reason of space, we will not go into the details here.

(10) a. jadoncha-nun$_1$ keu-ka [t_1 kuiphakess-tako]
the car$_{top}$ he$_{nom}$ [t_1 buy-to]
yaksokhassta.
promises
'As for the car, he promises to buy (it)'

b. keu-ka jadoncha-nun kuiphakess-tako yaksokhassta.
'He promises to buy the car'

German *topicalization* is more strict. German exhibits the verb second effect (V2), i.e., the finite verb (main verb or auxiliary) occupies the second position in the clause. This divides the clause into two parts: the part before the finite verb, the *Vorfeld* (VF), and the part between the finite verb and non-finite verb, the *Mittlefeld* (MF). The VF must contain exactly one constituent. This constituent is considered having moved into the VF. This movement is called *topicalization*. E.g., in (11) the auxiliary verb *hat* appears in second position, the NP$_{acc}$ *das Buch* that moved from the MF into the first position is topicalized.

(11) das Buch$_2$ hat ihm$_1$ niemand [t_1 t_2 zu geben] versucht.
the book has him nobody [t_1 t_2 to give] tried.
'Nobody has tried to give him the book.'

In both languages, *topicalization* concerns exactly one element, and the element has to appear in the beginning of the clause, while scrambling and extraposition can occur for more than one element. I.e., no operation to add constituents in front of topicalized element is accepted. Furthermore, in German matrix clauses, topicalization is obligatory. We capture these restrictions by certain features. The last step in a derivation for a sentence exhibiting *topicalization* is the adjunction of the topicalized constituent. The feature of the final derived root node becomes $\left[\begin{smallmatrix}\text{CP}\\ \text{CP}\end{smallmatrix}\right]$. It prevents adding other constituents at the root.[7]

Topicalization and *scrambling* can occur simultaneously as in (11) where *ihm* is long-distance scrambled and *das Buch* is long-distance topicalized. Fig. 6 shows the derivation for (11): Starting with the initial tree for *versucht*, the auxiliary tree for *geben* is adjoined at the root node with top category CP and bottom category VP (we assume here feature structures as labels with different top and bottom features), and simultaneously the initial VP tree is added into the lower VP. After this, the $\left[\begin{smallmatrix}\text{CP}\\ \text{VP}\end{smallmatrix}\right]$ root node is shared by *versucht* and *geben*. Then, *niemand* and *ihm* are subsequently added. This gives the tree on the left of the bottom of the figure. Next, *hat* is adjoined at the root which leads to a $\left[\begin{smallmatrix}\text{CP}\\ \text{C'}\end{smallmatrix}\right]$ root node shared (among others) by *geben* and *versucht*. Finally, the topicalized element is adjoined to the root node.

For topicalized elements in Korean, we propose the same kind of tree set as for German topicalized elements, except that the category of the foot node is unspecified. This does not fix the position of the topicalized element between CP and C'(as in German).

6 Conclusion

Since TAG are not powerful enough to describe scrambling data in free word order languages, alternative formalisms are needed. The proposals made so far in the litereature are not entirely satisfying. Therefore, we developed a new TAG extension, restricted MCTAG with shared nodes (RSN-MCTAG). The basic idea is that, after having performed an adjunction or substitution at some node, this node does not disappear (as in standard TAG) but instead, in the resulting derived tree, the node is shared between the old tree and the newly added tree. Consequently, further adjunctions at that node can be considered being adjunctions at either of the trees. In combination with tree-local multicomponent derivation, this modification of the TAG derivation gives sufficient additional power to analyse the difficult scrambling data.

Considering data from German and Korean, we showed that RSN-MCTAG can adequately analyse scrambling data, also in combination with extraposition and topicalization. The analyses proposed in the paper treat long-distance scrambling, long-distance extraposition and long-distance topicalization and they take into account the differences German and Korean exhibit with respect to these phenomena.

Acknowledgments

For helpful comments and fruitful discussions of the subject of this paper, we would like to thank Anne Abeillé, David Chiang, Aravind Joshi and Seth Kulick. Furthermore, we are grateful to three anonymous reviewers for their valuable suggestions for improving the paper.

References

Tilman Becker, Aravind K. Joshi, and Owen Rambow. 1991. Long-distance scrambling and tree adjoining grammars. In *Proceedings of ACL-Europe*.

Pierre Boullier. 1998. A Proposal for a Natural Language Processing Syntactic Backbone. Technical Report 3342, INRIA.

Pierre Boullier. 1999. On TAG Parsing. In *TALN 99, 6^e conférence annuelle sur le Traitement Automatique*

[7] We also pursued an alternative analysis, namely putting the slot for the topicalized element (a substitution node) and the verb it depends on in the same initial tree. I.e., the topicalized element is added by substitution while scrambled or extraposed elements are added by adjunction. This is a more obvious way to capture the restrictions for *topicalization*. Unfortunately, this approach does not work with some combinations of topicalization and scrambling as for example *[es]$_1$ hat er [t_1 zu reparieren]$_2$ dem Kunden [t_2 zu versuchen] versprochen.*

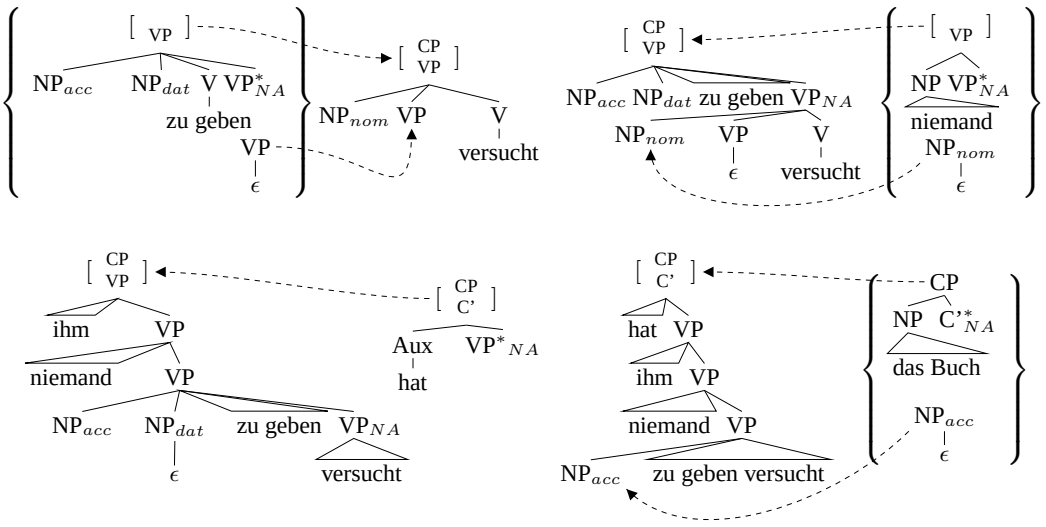


Figure 6: Derivation for (11) *... das Buch hat ihm niemand zu geben versucht*

des Langues Naturelles, pages 75–84, Cargèse, Corse, July.

Robert Frank. 1992. *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives*. Ph.D. thesis, University of Pennsylvania.

Claire Gardent and Laura Kallmeyer. 2003. Semantic Construction in FTAG. In *Proceedings of EACL 2003*, Budapest.

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, Berlin.

Aravind K. Joshi, Tilman Becker, and Owen Rambow. 2000. Complexity of scrambling: A new twist to the competence/performance distinction. In Anne Abeillé and Owen Rambow, editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analyses and Processing*. CSLI.

Aravind K. Joshi. 1987. An introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87–114. John Benjamins, Amsterdam.

Laura Kallmeyer. 2004. Tree-local multicomponent tree adjoining grammars with shared nodes. Unpublished manuscript, University of Paris 7. Under revision for resubmission, April.

Seth Norman Kulick. 2000. *Constraining Non-local Dependencies in Tree Adjoining Grammar: Computational and Linguistic Perspectives*. Ph.D. thesis, University of Pennsylvania.

Young-Suk Lee. 1993. *Scrambling as Case-driven Obligatory Movement*. Ph.D. thesis, University of Pennsylvania. Published as technical report IRCS-93-06.

Walt Detmar Meurers. 2000. *Lexical Generalizations in the Syntax of German Non-Finite Constructions*. Ph.D. thesis, Universität Tübingen.

Stefan Müller. 2002. *Complex Predicates: Verbal Complexes, Resultative Constructions, and Particle Verbs in German*. CSLI Stanford.

Owen Rambow and Young-Suk Lee. 1994. Word order variation and Tree-Adjoining Grammars. *Computational Intelligence*, 10(4):386–400.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Tree Substitution Grammars. *Computational Linguistics*.

Owen Rambow. 1994a. *Formal and Computational Aspects of Natural Language Syntax*. Ph.D. thesis, University of Pennsylvania.

Owen Rambow. 1994b. Multiset-Valued Linear Index Grammars: Imposing dominance constraints on derivations. In *Proceedings of ACL*.

Chung-Mok Suh. 2002. Topicalization and Focusing in Korean. In *The Twelfth International Conference on Korean Linguistics*, pages 511–522.

Hans Uszkoreit. 1987. *Word Order and Constituent Syntax in German*. CSLI Stanford.

K. Vijay-Shanker and Aravind K. Joshi. 1988. Feature structures based tree adjoining grammar. In *Proceedings of COLING*, pages 714–719, Budapest.

David J. Weir. 1988. *Characterizing mildly context-sensitive grammar formalisms*. Ph.D. thesis, University of Pennsylvania.

Subclasses of Tree Adjoining Grammar for RNA Secondary Structure

Yuki Kato
Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916-5, Ikoma, Nara 630-0192, Japan
yuuki-ka@is.naist.jp

Hiroyuki Seki
Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916-5, Ikoma, Nara 630-0192, Japan
seki@is.naist.jp

Tadao Kasami
Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916-5, Ikoma, Nara 630-0192, Japan
kasami@empirical.jp

Abstract

Several grammars have been proposed for representing RNA secondary structure including pseudoknots. In this paper, we introduce subclasses of multiple context-free grammars which are weakly equivalent to these grammars for RNA, and clarify the generative power of these grammars as well as closure property.

1 Introduction

Much attention has been paid to RNA secondary structure prediction techniques based on context-free grammar (cfg) since cfg can represent stem-loop structure (Figure 1 (a)) by its derivation tree and recognition (or *secondary structure prediction* in biological words) can be performed in $O(n^3)$ time where n is the length of an input sequence (primary structure). Especially, techniques based on CKY (Cocke-Kasami-Younger) algorithm have been widely investigated (Durbin et al., 1998). *Pseudoknot* (Figure 1 (b)) is one of the typical substructures found in an RNA secondary structure. An alternative representation of a pseudoknot is arc depiction in which arcs cross (see Figure 2). It has been recognized that pseudoknots play an important role in RNA functions such as ribosomal frameshifting and splicing. However, it is known that cfg cannot represent pseudoknot structure.

In bioinformatics, a few grammars have been proposed to represent pseudoknots (Uemura et al., 1999; Rivas and Eddy, 2000) (also see (Condon, 2003)). In the pioneering paper, Uemura et al. (1999) define two subclasses of tree adjoining grammar (tag) called *sl-tag* and *esl-tag*, and argue that esl-tag is appropriate for representing RNA secondary structure including pseudoknots. Rivas and Eddy (2000) provide keen observation on representation of RNA secondary structure by a sequence with a single "hole" and introduce a new class of grammars for deriving sequences with hole. These grammars have generative power stronger than cfg while recognition can be performed in polynomial time. However, relation among the generative power of these grammars and/or mildly csg has not been clarified.

In this paper, we identify grammars for RNA secondary structure (Uemura et al., 1999; Rivas and Eddy, 2000) as subclasses of multiple context-free grammar (mcfg) (Kasami et al., 1988a; Seki et al., 1991) and clarify inclusion relation among the classes of languages generated by these grammars.

The rest of this paper is organized as follows. Section 2 reviews the grammars mentioned above. In section 3, these grammars are characterized as subclasses of mcfg. Generative power and closure property of these grammars are discussed in section 4. Section 5 concludes the paper.

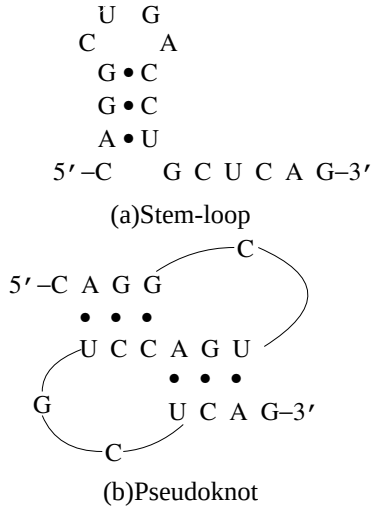


Figure 1: Example of RNA secondary structure

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 48-55.

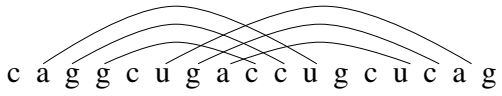


Figure 2: Arc depiction of Figure 1 (b)

2 Preliminaries

2.1 Tree Adjoining Grammar

We will use standard notations for tree adjoining grammar (Joshi and Schabes, 1997). The empty sequence is denoted by ε. For a sequence $\alpha \in S^*$, let $|\alpha|$ denote the length of α.

A *tree adjoining grammar* (*tag*) is a 5-tuple $G = (N, T, S, \mathcal{I}, \mathcal{A})$ where N and T are finite sets of nonterminals and terminals respectively, S the start symbol, $\mathcal{I}$ a finite set of *initial trees* (*center trees*) and $\mathcal{A}$ a finite set of *adjunct trees* (*auxiliary trees*). The path of an adjunct tree from the root node to the foot node is called the *backbone*. *Selective adjoining* (SA), *null adjoining* (NA) and *obligatory adjoining* (OA) are defined in the standard way. For trees s and t, if t' is obtained by adjoining s into t, we write $t \vdash_s t'$ (or simply $t \vdash t'$). We write the reflective and transitive closure of $\vdash$ as $\vdash^*$. We call t' a *derived tree* (or a tree derived from t) if $t \vdash^* t'$ for some $t \in \mathcal{I} \cup \mathcal{A}$. A node n is *inactive* if the constraint for the node is NA, otherwise *active*. If no active node in a tree t has OA constraint, then t is called *mature*. The tree set of a tag G is defined as $T(G) = \{t \mid s \vdash^* t,\ s \in \mathcal{I}$ and t is mature$\}$. $T(G)$ can be alternatively characterized in a bottom up way as follows. Let us define a series of tree sets $T_0(G),\ T_1(G), \ldots$.

(T1) $T_0(G) = \{t \in \mathcal{I} \cup \mathcal{A} \mid t$ is mature$\}$.

(T2) $T_{n+1}(G) = T_n(G) \cup \{t \mid t_0 \vdash_{s_1} t_1 \vdash_{s_2} \cdots \vdash_{s_k} t_k = t,\ t_0 \in \mathcal{I} \cup \mathcal{A},\ s_i \in T_n(G)\ (1 \le i \le k),\ p_1, \ldots, p_k$ are different addresses of t_0, s_i is adjoinable to t_0 at $p_i\ (1 \le i \le k)$ and t is mature$\}$.

It is not difficult to show that $T(G) = \{t \mid t \in T_n(G)$ for some $n \ge 0$ and yield$(t) \in T^*\}$. This characterization of $T(G)$ by (T1) and (T2) is frequently used in proofs in section 3.

The language generated by G is defined as $L(G) = \{w \mid w =$ yield$(t),\ t \in T(G)\}$, which is called a *tree adjoining language* (*tal*). Let TAG denote the class of tags and TAL denote the class of tals. We use the same notational convention, i.e., a language generated by an xxg is called an xxl, the class of xxgs is denoted by XXG and the class of xxls is denoted by XXL.

We now define *simple linear tag* (*sl-tag*) and *extended simple linear tag* (*esl-tag*) introduced in (Uemura et al., 1999). Let $G = (N, T, S, \mathcal{I}, \mathcal{A})$ be a tag. An elementary tree is *simple linear* if it has exactly one active node, and for an adjunct tree, the active node is on the backbone of the tree. A tag G is a *simple linear tag* (*sl-tag*) if and only if all elementary trees in G are simple linear. An adjunct tree is *semi-simple linear* if it has two active nodes, where one is on the backbone and the other is elsewhere. A tag G is an *extended simple linear tag* (*esl-tag*) if and only if all initial trees in G are simple linear and all adjunct trees in G are either simple linear or semi-simple linear.

Example 1 (Uemura et al., 1999). Let $G = (N, T, S, \mathcal{I}, \mathcal{A})$ be an sl-tag where $N = \{S\}$, $T = \{a, c, g, u\}$ and elementary trees in $\mathcal{I}$ and $\mathcal{A}$ are shown in Figure 3. In the figure, $z \in \{a, c, g, u\}$, $(x, y) \in \{(a, u), (u, a), (c, g), (g, c)\}$ and an active node is denoted by S^*. Figure 4 shows a derivation of a pseudoknot. □

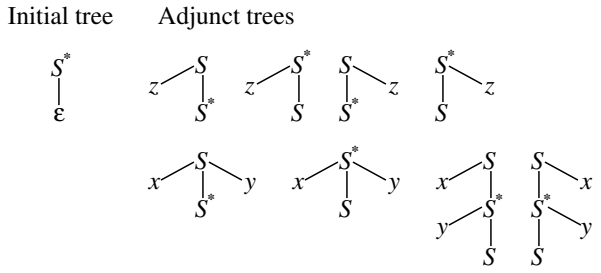


Figure 3: Elementary trees in Example 1

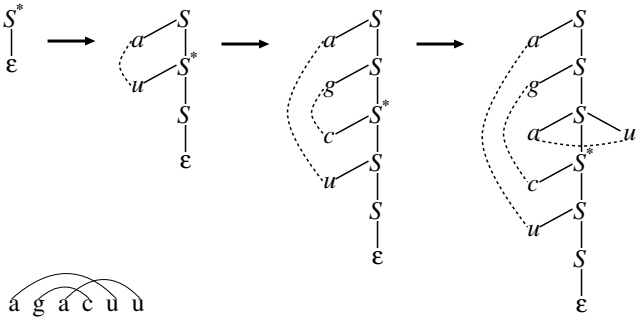


Figure 4: A derivation of a pseudoknot in Example 1

By definition,

$$\text{SL-TAL} \subseteq \text{ESL-TAL} \subseteq \text{TAL}. \qquad (*1)$$

On the inclusion relation among CFL, SL-TAL and ESL-TAL, the following has been shown in Propositions 1 to 3 of (Uemura et al., 1999):

$$L_2 = \{\sharp a_1^k b_1^k \sharp a_2^l b_2^l \sharp a_3^m b_3^m \sharp a_4^n b_4^n \sharp \mid k, l, m, n \ge 1\} \in \text{CFL} \setminus \text{SL-TAL}, \qquad (*2)$$

$$\{a^n b^n c^n \mid n \ge 0\} \in \text{SL-TAL} \setminus \text{CFL}, \qquad (*3)$$

$$\text{CFL} \subseteq \text{ESL-TAL}. \qquad (*4)$$

2.2 Multiple Context-Free Grammar

A *multiple context-free grammar* (*mcfg*) or *linear context-free rewriting system* (Vijay-Shanker et al., 1987) is a 5-tuple $G = (N, T, F, P, S)$ where N is a finite set of nonterminals, T a finite set of terminals, F a finite set of functions, P a finite set of (production) rules and S the start symbol. For each $A \in N$, a positive integer denoted as $\dim(A)$ is given and A derives $\dim(A)$-tuples of terminal sequences. For the start symbol S, $\dim(S) = 1$. For each $f \in F$, positive integers d_i $(0 \leq i \leq k)$ are given and f is a total function from $(T^*)^{d_1} \times \cdots \times (T^*)^{d_k}$ to $(T^*)^{d_0}$ which satisfies the following condition (F):

(F) Let $\overline{x_i} = (x_{i1}, \ldots, x_{id_i})$ denote the ith argument of f for $1 \leq i \leq k$. The hth component of function value for $1 \leq h \leq d_0$, denoted by $f^{[h]}$, is defined as

$$f^{[h]}[\overline{x_1}, \ldots, \overline{x_k}] = \beta_{h0} z_{h1} \beta_{h1} z_{h2} \cdots z_{hv_h} \beta_{hv_h} \quad (*)$$

where $\beta_{hl} \in T^*$ $(0 \leq l \leq v_h)$ and $z_{hl} \in \{x_{ij} \mid 1 \leq i \leq k,\ 1 \leq j \leq d_i\}$ $(1 \leq l \leq v_h)$. The total number of occurrences of x_{ij} in the right hand sides of $(*)$ from $h = 1$ through d_0 is at most one.

Each rule in P has the form of $A_0 \to f[A_1, \ldots, A_k]$ where $A_i \in N$ $(0 \leq i \leq k)$ and $f : (T^*)^{\dim(A_1)} \times \cdots \times (T^*)^{\dim(A_k)} \to (T^*)^{\dim(A_0)} \in F$. If $k \geq 1$, then the rule is called a *nonterminating rule*, and if $k = 0$, then it is called a *terminating rule*.

We define the relation $\stackrel{*}{\Rightarrow}$ and derivation trees (refer to Figure 5) recursively by the following (L1) and (L2):

(L1) If $A \to \alpha \in P$ $(\alpha \in T^*)$, then $A \stackrel{*}{\Rightarrow} \alpha$ and a tree with the single node labeled $A : \alpha$ is a derivation tree for α.

(L2) If $A \to f[A_1, \ldots, A_k] \in P$, $A_i \stackrel{*}{\Rightarrow} \alpha_i = (\alpha_{i1}, \ldots, \alpha_{i\dim(A_i)})$ $(1 \leq i \leq k)$ and $t_1, \ldots, t_k$ are derivation trees for $\alpha_1, \ldots, \alpha_k$, then $A \stackrel{*}{\Rightarrow} f[\alpha_1, \ldots, \alpha_k]$ where $f[\alpha_1, \ldots, \alpha_k]$ denotes the $\dim(A)$-tuple of terminal sequences obtained from the right hand sides of $(*)$ in condition (F) by substituting α_{ij} $(1 \leq i \leq k,\ 1 \leq j \leq \dim(A_i))$ into x_{ij}, and a tree with the root labeled $A : f$ which has $t_1, \ldots, t_k$ as (immediate) subtrees from left to right is a derivation tree for $f[\alpha_1, \ldots, \alpha_k]$.

The language generated by an mcfg G is defined as $L(G) = \{w \in T^* \mid S \stackrel{*}{\Rightarrow} w\}$.

To introduce subclasses of MCFG, we define a few terminologies. Let $G = (N, T, F, P, S)$ be an arbitrary mcfg. The *dimension* of G is defined as $\dim(G) = \max\{\dim(A) \mid A \in N\}$. For a function $f \in F$, let $\mathrm{rank}(f)$ denote the number of arguments of f. The *rank* of G is defined as $\mathrm{rank}(G) = \max\{\mathrm{rank}(f) \mid f \in F\}$. For a function $f : (T^*)^{d_1} \times \cdots \times (T^*)^{d_k} \to (T^*)^{d_0}$, let $\deg(f) = \Sigma_{j=0}^{k} d_j$, which is called the *degree* of f. Finally, let us define the degree of G as $\deg(G) = \max\{\deg(f) \mid f \in F\}$. By definition, $\deg(G) \leq \dim(G)(\mathrm{rank}(G) + 1)$. With these parameters, we define subclasses of MCFG. An mcfg G with $\dim(G) \leq m$ and $\mathrm{rank}(G) \leq r$ is called an (m, r)-mcfg. Likewise, an mcfg G with $\dim(G) \leq m$ is called an m-mcfg.

It has been proved that

$$\text{TAL} \subset \text{(2,2)-MCFL} \subset \text{2-MCFL} \subset \text{MCFL}, \quad (*5)$$

where the proper inclusion relation from left to right in $(*5)$ were given by Lemma 4.15 of (Seki et al., 1991), Theorem 1 of (Rambow and Satta, 1994) and Lemma 5 of (Kasami et al., 1988a), respectively.

Example 2. Consider the (2,2)-mcfg $G_3 = (\{S, A\}, \{a, c, g, u\}, F_3, P_3, S)$ for generating RNA sequences, where P_3 and F_3 are as follows:

$$\begin{aligned}
&S \to J[A],\\
&A \to XS_1[A, A] \mid XS_2[A, A] \mid XS_3[A, A],\\
&A \to BF_1[A, A] \mid BF_2[A, A] \mid BF_3[A, A],\\
&A \to BP_{\alpha\beta}[A]\\
&\quad ((\alpha, \beta) \in \{(a, u), (u, a), (c, g), (g, c)\}),\\
&A \to UP_{\alpha}^{1,L}[A] \mid UP_{\alpha}^{1,R}[A] \mid UP_{\alpha}^{2,L}[A] \mid UP_{\alpha}^{2,R}[A]\\
&\quad (\alpha \in \{a, c, g, u\}),\\
&A \to (\varepsilon, \varepsilon),\\
&J[(x_1, x_2)] = x_1 x_2,\\
&XS_1[(x_{11}, x_{12}), (x_{21}, x_{22})] = (x_{11}, x_{21} x_{12} x_{22}),\\
&XS_2[(x_{11}, x_{12}), (x_{21}, x_{22})] = (x_{11} x_{21}, x_{12} x_{22}),\\
&XS_3[(x_{11}, x_{12}), (x_{21}, x_{22})] = (x_{11} x_{21} x_{12}, x_{22}),\\
&BF_1[(x_{11}, x_{12}), (x_{21}, x_{22})] = (x_{11}, x_{12} x_{21} x_{22}),\\
&BF_2[(x_{11}, x_{12}), (x_{21}, x_{22})] = (x_{11} x_{12}, x_{21} x_{22}),\\
&BF_3[(x_{11}, x_{12}), (x_{21}, x_{22})] = (x_{11} x_{12} x_{21}, x_{22}),\\
&BP_{\alpha\beta}[(x_1, x_2)] = (\alpha x_1, x_2 \beta),\\
&UP_{\alpha}^{1,L}[(x_1, x_2)] = (\alpha x_1, x_2),\\
&UP_{\alpha}^{1,R}[(x_1, x_2)] = (x_1 \alpha, x_2),\\
&UP_{\alpha}^{2,L}[(x_1, x_2)] = (x_1, \alpha x_2),\\
&UP_{\alpha}^{2,R}[(x_1, x_2)] = (x_1, x_2 \alpha).
\end{aligned}$$

Functions have mnemonic names where XS, BF, BP and UP stand for crossing, bifurcation, base pair and unpair, respectively. The RNA sequence `agacuu` in Figure 4 can be generated by the above rules as follows: $A \stackrel{*}{\Rightarrow} BP_{gc}[(\varepsilon, \varepsilon)] = (g, c)$, $A \stackrel{*}{\Rightarrow} BP_{au}[(g, c)] = (ag, cu)$, $A \stackrel{*}{\Rightarrow} BP_{au}[(\varepsilon, \varepsilon)] = (a, u)$, $A \stackrel{*}{\Rightarrow} XS_2[(ag, cu), (a, u)] = (aga, cuu)$ and $S \stackrel{*}{\Rightarrow}$

$J[(aga, cuu)] = agacuu$. G_3 has a derivation tree (Figure 5) for agacuu which represents the pseudoknot shown in Figure 4. □

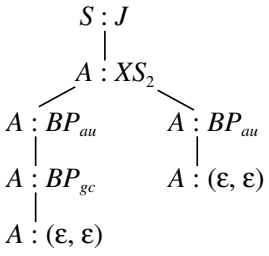


Figure 5: A derivation tree in G_3

Recognition problem for mcfg can be solved in polynomial time:

Proposition 1 (Kasami et al., 1988b; Seki et al., 1991). Let G be an mcfg with $\deg(G) = e$. For a given $w \in T^*$, whether $w \in L(G)$ or not can be decided in $O(n^e)$ time where $n = |w|$. □

3 Subclasses of MCFG

3.1 A Subclass of MCFG for SL-TAL

Grammars G and G' are called weakly equivalent if $L(G) = L(G')$. Remember that each elementary tree in an sl-tag contains exactly one active node as shown in Figure 6 (An inactive node and an active node are denoted like A^ϕ and B^*, respectively in the figure). By utilizing this restriction, we can define a translation from an sl-tag into a weakly equivalent (2,2)-mcfg simpler than that of (Vijay-Shanker et al., 1986). Namely, for an adjunct tree in Figure 6 (a), construct an mcfg rule $A \to f[B]$ where $f[(x_1, x_2)] = (u_1x_1v_1, v_2x_2u_2)$. This translation motivates us to define the following subclass of (2,1)-MCFG.

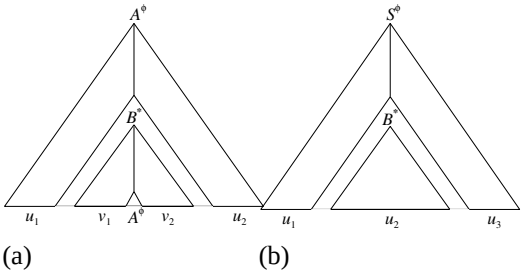


Figure 6: Elementary trees in sl-tag

Definition 1. A (2,1)-mcfg $G = (N, T, F, P, S_0)$ is an *sl-mcfg* if G satisfies the following conditions (1) and (2):

(1) For each nonterminal A other than S_0, $\dim(A) = 2$.

(2) Each nonterminating rule has the form of either $S_0 \to J[A]$ where $J[(x_1, x_2)] = x_1x_2$ or $A \to f[B]$ where $A, B \in N \setminus \{S_0\}$ and $f[(x_1, x_2)] = (u_1x_1v_1, v_2x_2u_2)$ for some $u_j, v_j \in T^*$ $(j = 1, 2)$. Such a function f is called a *simple linear function*. □

Lemma 2. SL-TAL = SL-MCFL.

Proof. (SL-TAL $\subseteq$ SL-MCFL) Let $G = (N, T, S, \mathcal{I}, \mathcal{A})$ be a given sl-tag. We will construct an sl-mcfg $G' = (N', T, F, P, S_0)$ as follows:

(1) $N' = N \cup \{S_0\}$ where $\dim(S_0) = 1$ and $\dim(A) = 2$ for each $A \in N$.

(2) P (and F) are the smallest sets which satisfy the following conditions (a) through (c):

(a) $S_0 \to J[S] \in P$ and $J \in F$.

(b) For each adjunct tree $t \in \mathcal{A}$ shown in Figure 6 (a),
- $A \to f[B] \in P$ and $f \in F$ where $f[(x_1, x_2)] = (u_1x_1v_1, v_2x_2u_2)$, and
- $A \to (u_1v_1, v_2u_2)$ if B in Figure 6 (a) does not have OA constraint (i.e., t is mature).

(c) For each initial tree $t \in \mathcal{I}$ shown in Figure 6 (b),
- $S \to g[B] \in P$ and $g \in F$ where $g[(x_1, x_2)] = (u_1x_1u_2, x_2u_3)$, and
- $S \to (u_1u_2, u_3)$ if t is mature.

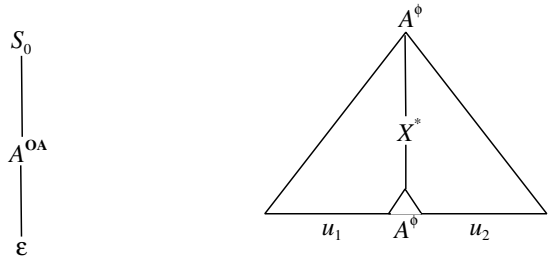
We can show that there exists a tree $t \in T_n(G)$ for some $n \geq 0$ such that yield$(t) = w_1Aw_2$ $(A \in N,\ w_1, w_2 \in T^*)$ if and only if $A \overset{*}{\Rightarrow}_{G'} (w_1, w_2)$.
(SL-MCFL $\subseteq$ SL-TAL) Let $G = (N, T, F, P, S_0)$ be a given sl-mcfg. Construct an sl-tag $G' = (N', T, S_0, \mathcal{I}, \mathcal{A})$ as follows:

(1) $N' = N \cup \{X\}$ where $X \notin N$.

(2) $\mathcal{I}$ consists of initial trees shown in Figure 7 (a) for $S_0 \to J[A] \in P$.

(3) $\mathcal{A}$ is the smallest set satisfying:
- For each $A \to f[B] \in P$ where $f[(x_1, x_2)] = (u_1x_1v_1, v_2x_2u_2)$, the adjunct tree shown in Figure 6 (a) belongs to $\mathcal{A}$.
- For each $A \to (u_1, u_2) \in P$, the adjunct tree in Figure 7 (b) belongs to $\mathcal{A}$.

Proof of $L(G) = L(G')$ can be done in a similar way to the converse direction. □



(a) (b)

Figure 7: Constructed elementary trees

3.2 A Subclass of MCFG for ESL-TAL

In this subsection, we will define a subclass of (2,2)-MCFG which exactly generates ESL-TAL. Let $G = (N, T, S, \mathcal{I}, \mathcal{A})$ be a given esl-tag. By virtue of Property 2 of (Uemura et al., 1999), we can assume that G is in normal form such that for every semi-simple linear adjunct tree $t \in \mathcal{A}$, yield$(t) \in N$. Thus, for each leaf v of t, either v is the foot node or the label of v is ε (see Figure 8). From this observation, we define a subclass of (2,2)-MCFG by adding rules corresponding to adjunct trees shown in Figure 8 to the definition of sl-mcfg.

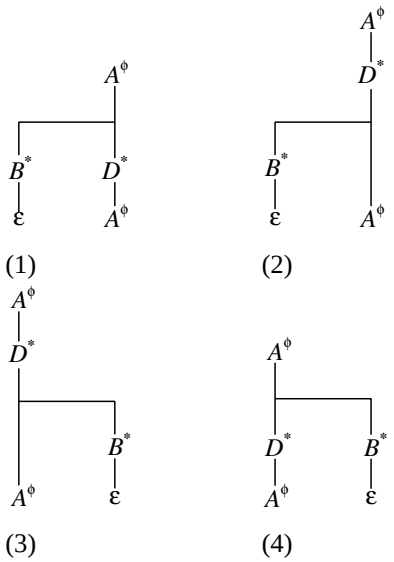


Figure 8: Semi-simple linear adjunct trees in normal form

Definition 2. A (2,2)-mcfg $G = (N, T, F, P, S_0)$ is an *esl-mcfg* if each nonterminating rule has one of the following forms (1) through (3):

(1) $A \to J[B]$ where $\dim(A) = 1$ and $\dim(B) = 2$.

(2) $A \to f[B]$ where f is a simple linear function.

(3) $A \to g[B, D]$ where $\dim(A) = \dim(D) = 2$, $\dim(B) = 1$, $g \in \{C_1, C_2, C_3, C_4\}$ and

$$C_1[x_1, (x_{21}, x_{22})] = (x_1 x_{21}, x_{22}),$$
$$C_2[x_1, (x_{21}, x_{22})] = (x_{21} x_1, x_{22}),$$
$$C_3[x_1, (x_{21}, x_{22})] = (x_{21}, x_1 x_{22}),$$
$$C_4[x_1, (x_{21}, x_{22})] = (x_{21}, x_{22} x_1).$$

□

Lemma 3. ESL-TAL = ESL-MCFL.

Proof. (ESL-TAL $\subseteq$ ESL-MCFL) Let $G = (N, T, S, \mathcal{I}, \mathcal{A})$ be a given esl-tag in normal form (Uemura et al., 1999). We construct an esl-mcfg $G' = (N', T, F, P, S_0)$ from G as follows:

(1) $N' = N \cup \{A' \mid A \in N\}$ where $\dim(A') = 1$ and $\dim(A) = 2$ for $A \in N$.

(2) P (and F) are the smallest sets which satisfy the following conditions (a) through (d):

(a) For each $A \in N$, $A' \to J[A] \in P$ and $J \in F$.

(b) Same as (2) (b) (c) in the proof of (SL-TAL $\subseteq$ SL-MCFL) in Lemma 2.

(c) For each semi-simple linear adjunct tree t shown in Figure 8 (1),
- $A \to C_1[B', D] \in P$ and $C_1 \in F$, and
- $A \to (\varepsilon, \varepsilon) \in P$ if t is mature.

(d) For each semi-simple linear adjunct tree (2) through (4) in Figure 8, the rules using C_2, C_3 and C_4, respectively, instead of C_1 belong to P.

We can show that there exists a tree $t \in T_n(G)$ for some $n \geq 0$ such that yield$(t) = w_1 A w_2$ ($A \in N$, $w_1, w_2 \in T^*$) if and only if $A \overset{*}{\Rightarrow}_{G'} (w_1, w_2)$.
Proof of (ESL-MCFL $\subseteq$ ESL-TAL) is similar and is omitted here. □

3.3 A Subclass of MCFG for RPL

Rivas and Eddy (2000) introduce *crossed-interaction grammar* (*cig*) which is similar to mcfg, and define *RNA pseudoknot grammar* (*rpg*) as a subclass of CIG to describe RNA secondary structure including pseudoknots. In this subsection, we reformulate RPG as a subclass of MCFG.

Definition 3. A (2,2)-mcfg $G = (N, T, F, P, S)$ is called an rpg if a nonterminating rule is one of the following forms (1) through (3):

(1) $A \to J[B]$.

(2) $A \to BF[E_1, E_2]$ where $\dim(A) = 2$, $\dim(E_1) = \dim(E_2) = 1$ and $BF[x_1, x_2] = (x_1, x_2)$.

(3) $A \to f[B,D]$ where $\dim(A) = \dim(B) = \dim(D) = 2$, $f \in \{XS_1, XS_2, XS_3, W\}$, XS_i $(i = 1,2,3)$ is defined in Example 2 and $W[(x_{11},x_{12}),(x_{21},x_{22})] = (x_{11}x_{21}, x_{22}x_{12})$. □

Proposition 4.

$$\text{RPL} \subseteq \text{(2,2)-MCFL}. \qquad (*6)$$

□

We obtain the following property on recognition complexity.

Proposition 5. For a given $w \in T^*$ $(n = |w|)$, whether $w \in L$ or not can be decided in $O(n^6)$ time if L is an rpl, $O(n^5)$ time if L is an esl-tal, and $O(n^4)$ time if L is an sl-tal.

Proof. For an rpg G, $\deg(G) \leq 6$, for an esl-mcfg G, $\deg(G) \leq 5$ and for an sl-mcfg G, $\deg(G) \leq 4$. The proposition follows from Proposition 1, Lemmas 2 and 3. □

The above complexity results were first shown in (Uemura et al., 1999) for ESL-TAL and SL-TAL and in (Rivas and Eddy, 2000) for RPL by providing an individual recognition algorithm for each class. On the other hand, by identifying these classes of languages as subclasses of MCFL, we can easily obtain the same results as stated in Proposition 5. Akutsu (2000) defines a structure called a simple pseudoknot and proposes an $O(n^4)$ time exact prediction algorithm and $O(n^{4-\delta})$ time approximation algorithm without using grammar. Note that the set of simple pseudoknots can be generated by an sl-tag.

4 Inclusion Relation

First, we summarize the inclusion relation among the classes of languages stated in $(*1)$ through $(*6)$.

Proposition 6. (1) $(\text{CFL} \cup \text{SL-TAL}) \subseteq \text{ESL-TAL} \subseteq \text{TAL} \subset \text{(2,2)-MCFL}$.

(2) $\text{RPL} \subseteq \text{(2,2)-MCFL} \subset \text{2-MCFL} \subset \text{MCFL}$. □

In the following, we refine the above proposition.

4.1 $(\text{CFL} \cup \text{SL-TAL}) \subset \text{ESL-TAL}$

First, we introduce a normal form of esl-mcfg and then show closure properties of SL-TAL and ESL-TAL. By using sl-mcfg and esl-mcfg, we can prove these properties in a simple way. Some of these properties will be used for proving inclusion relation between SL-TAL and ESL-TAL.

Definition 4. An esl-mcfg is in normal form if the following conditions (1) and (2) hold:

(1) For each $A \to f[B]$ where $f[(x_1,x_2)] = (u_1x_1v_1, v_2x_2u_2)$ is a linear function, $|u_1v_1v_2u_2| = 1$.

(2) For each $A \to (u_1,u_2)$ $(u_1,u_2 \in T^*)$, $u_1 = u_2 = \varepsilon$. □

Remark that a similar normal form is defined for esl-tag in (Uemura et al., 1999). It is easy to prove the following lemma.

Lemma 7. For a given esl-mcfg G, a normal form esl-mcfg G' can be constructed from G such that $L(G') = L(G)$. □

Theorem 8. SL-TAL and ESL-TAL have the following properties.

(1) SL-TAL contains every linear language.

(2) SL-TAL is closed under union, homomorphism, intersection with regular languages and regular substitution, but is not closed under concatenation, Kleene closure, positive closure or substitution.

(3) ESL-TAL is closed under intersection with regular languages and substitution.

Proof. (1) For linear cfg rules $A \to u_1Bv_1$ and $A \to u$, construct sl-mcfg rules $A \to f[B]$ where $f[(x_1,x_2)] = (u_1x_1v_1, x_2)$ and $A \to (u,\varepsilon)$, respectively.

(2) (regular substitution) Let $G = (N,T,F,P,S_0)$ be an sl-mcfg in normal form. We also assume that each rule $A \to f[B] \in P$ has a unique label, say r, and write $r : A \to f[B] \in P$. Let $s : T \to 2^{(T')^*}$ be a regular substitution and for each $\alpha \in T$, let $s(\alpha) = L(G_\alpha)$ where $G_\alpha = (N_\alpha, T', P_\alpha, S_\alpha)$ is a regular grammar. We now construct an sl-mcfg $G' = (N',T',F',P',S_0)$ such that $L(G') = s(L(G))$ as follows. G' will simulate G_α by a linear function instead of generating $\alpha \in T$. To do this, we introduce a nonterminal $X^{[r]}$ in G' where $X \in N_\alpha$ and $r : A \to f[B] \in P$ such that the definition of f contains $\alpha \in T$.

- $N' = N \cup \{X^{[r]} \mid X \in N_\alpha \setminus \{S_\alpha\},\ \alpha \in T,\ r : A \to f[B] \in P\}$.
- F' consists of J, $UP_\beta^{1,L}$, $UP_\beta^{1,R}$, $UP_\beta^{2,L}$, $UP_\beta^{2,R}$ $(\beta \in T')$ of Example 2 and $EPS[\,] = (\varepsilon,\varepsilon)$.
- P' is the smallest set satisfying:
  - If $S_0 \to J[A] \in P$, then $S_0 \to J[A] \in P'$.
  - Assume that $r : A \to f[B] \in P$ where $f[(x_1,x_2)] = (\alpha x_1, x_2)$ $(\alpha \in T)$. If $X \to \beta Y \in P_\alpha$ $(X,Y \in N_\alpha,\ \beta \in T')$,

then $X^{[r]} \to UP_\beta^{1,L}[Y^{[r]}] \in P'$, and if $X \to \beta \in P_\alpha$ ($X \in N_\alpha$, $\beta \in T'$), then $X^{[r]} \to UP_\beta^{1,L}[B] \in P'$ where $S_\alpha^{[r]}$ is identified with A for simplicity.

– For the other rules in P, similar construction can be defined. For example, if $f[(x_1, x_2)] = (x_1, x_2\alpha)$ ($\alpha \in T$), then we will use $UP_\beta^{2,R}$ instead of $UP_\beta^{1,L}$.

Proof of $L(G') = s(L(G))$ is easy.
The other closure properties can be easily proved.
(concatenation) Let $L = \{\sharp a_1^k b_1^k \sharp a_2^l b_2^l \mid k, l \geq 1\}$ and $L' = \{\sharp a_3^m b_3^m \sharp a_4^n b_4^n \sharp \mid m, n \geq 1\}$, both of which are sl-tals. An sl-mcfg which generates L is such that $S_0 \to J[S]$, $S \to \text{add}^{\sharp\sharp}[A]$ where $\text{add}^{\sharp\sharp}[(x_1, x_2)] = (\sharp x_1, \sharp x_2)$, $A \to f[A] \mid B$ where $f[(x_1, x_2)] = (a_1 x_1 b_1, x_2)$ and $B \to g[B] \mid (a_1 b_1, a_2 b_2)$ where $g[(x_1, x_2)] = (x_1, a_2 x_2 b_2)$. Construction of an sl-mcfg which generates L' is similar. The concatenation of them, i.e., $LL' = L_2$ defined in ($*2$) is not an sl-tal.
(Kleene closure, positive closure) By the next corollary, SL-TAL is a union closed full trio. If SL-TAL is closed under Kleene closure or positive closure, then by Theorem 3.1 of (Mateescu and Salomaa, 1997), SL-TAL is closed under concatenation, which is a contradiction.
(substitution) Let $L_1 = \{\sharp d_1 \sharp d_2 \sharp d_3 \sharp d_4 \sharp\}$, which is a finite language and thus an sl-tal, and let s be a substitution such that $s(d_i) = \{a_i^n b_i^n \mid n \geq 1\}$ ($1 \leq i \leq 4$), which is also an sl-tal by (1) of this theorem. Then $s(L_1) = L_2$ defined in ($*2$), which is not an sl-tal.

(3) (intersection with regular languages) Same as the proof of Theorem 3.9 (3) of (Seki et al., 1991).
(substitution) Easy. □

Corollary 9. SL-TAL is a full trio (or cone). (That is, SL-TAL is closed under homomorphism, inverse homomorphism and intersection with regular languages.) ESL-TAL is a substitution closed full abstract family of languages (full AFL). (That is, ESL-TAL is a full trio and closed under union, concatenation, Kleene closure and substitution.)

Proof. (full trio) By Theorem 3.2 of (Mateescu and Salomaa, 1997) and (2) of Theorem 8. (full AFL) By Theorem 3.3 of (Mateescu and Salomaa, 1997) and (1), (3) of Theorem 8. □

Now we show inclusion relation between SL-TAL and ESL-TAL.

Theorem 10. Let $L_3 = \{\sharp a_1^k b_1^k c_1^k \sharp a_2^l b_2^l c_2^l \sharp a_3^m b_3^m c_3^m \sharp a_4^n b_4^n c_4^n \sharp \mid k, l, m, n \geq 1\}$. Then, $L_3 \in$ ESL-TAL $\setminus$ (CFL $\cup$ SL-TAL).

Proof. Let h_1 be a homomorphism such that $h_1(a_1) = a_1$, $h_1(b_1) = b_1$, $h_1(c_1) = c_1$ and $h_1(x) = \varepsilon$ for $x \in \{a_i, b_i, c_i \mid i = 2, 3, 4\} \cup \{\sharp\}$. Then $h_1(L_3) = \{a_1^k b_1^k c_1^k \mid k \geq 1\}$, which is not a cfl. Since CFL is closed under homomorphism, L_3 is not a cfl. Similarly, let h_2 be a homomorphism such that $h_2(c_i) = \varepsilon$ for $i = 1, 2, 3$ and identity on the other symbols. Then $h_2(L_3) = L_2$ defined in ($*2$), which is not an sl-tal. By Theorem 8 (2), L_3 is not an sl-tal. We can easily give an esl-mcfg which generates L_3. □

4.2 RPL = (2,2)-MCFL

We introduce a condition (S) which states that for each argument (x_{i1}, x_{i2}) of a function of an mcfg, the order of the occurrences of its components x_{i1} and x_{i2} is not interchanged in the function value.

(S) Let $G = (N, T, F, P, S)$ be a 2-mcfg and f be an arbitrary function in F such that

$$f[(x_{11}, x_{12}), \ldots, (x_{n1}, x_{n2})] = (\alpha_1, \alpha_2).$$

For each i ($1 \leq i \leq n$), if both of x_{i1} and x_{i2} occur in $\alpha_1\alpha_2$, then x_{i1} occurs to the left of the occurrence of x_{i2}, i.e., $\alpha_1\alpha_2 = \beta_1 x_{i1} \beta_2 x_{i2} \beta_3$ for some $\beta_j \in (N \cup T)^*$ ($1 \leq j \leq 3$).

Lemma 11. For a given 2-mcfg G, we can construct a 2-mcfg G' satisfying condition (S) and $L(G') = L(G)$. □

Lemma 12. Let $G = (N, T, F, P, S)$ be a (2,2)-mcfg satisfying condition (S). Then we can construct an rpg G' such that $L(G') = L(G)$.

Proof. Let $G = (N, T, F, P, S)$ be an arbitrary (2,2)-mcfg satisfying condition (S). We construct an rpg G' weakly equivalent to G as follows. The number of functions $f : (T^*)^2 \times (T^*)^2 \to (T^*)^2$ satisfying condition (S) is 18. A half of them can be obtained from the other half of them by interchanging the first and second arguments. Among the remaining nine functions, four are rpg functions. The others are $f_1 = (x_{11}, x_{12}x_{21}x_{22})$, $f_2 = (x_{11}x_{12}, x_{21}x_{22})$, $f_3 = (x_{11}x_{12}x_{21}, x_{22})$, $f_4 = (x_{11}, x_{21}x_{22}x_{12})$, $f_5 = (x_{11}x_{21}x_{22}, x_{12})$. (We omit variables in the left hand sides.) For example, $A \to f_1[B, D]$ can be simulated by $A \to XS_2[B, Y_1]$, $Y_1 \to BF[Y_2, Y_3]$, $Y_2 \to \varepsilon$ and $Y_3 \to J[D]$. The other four functions can be simulated by rpg functions in a similar way. □

By Proposition 6 (2), Lemmas 11 and 12, we obtain the following theorem.

Theorem 13. RPL = (2,2)-MCFL. □

The following corollary follows from Proposition 6, Theorems 10 and 13.

Corollary 14. $(\text{CFL} \cup \text{SL-TAL}) \subset \text{ESL-TAL} \subseteq \text{TAL} \subset \text{RPL} = \text{(2,2)-MCFL}$. □

Whether the inclusion ESL-TAL $\subseteq$ TAL is proper or not is an open problem.

5 Conclusions

In this paper, some formal grammars for RNA secondary structure have been identified as subclasses of MCFG and their generative powers have been compared. To the authors' knowledge, the exact definition of pseudoknot in a biological or geometrical sense is not known and then it is difficult to answer which class of grammars is the minimum to represent pseudoknots. However, SL-TAG cannot generate RNA sequences obtained by repeating a simple pseudoknot shown in Figure 2 by ($*2$), and ESL-TAG (or ESL-MCFG) can be the minimum grammars which can represent such a class of pseudoknots.

Meanwhile, Satta and Schuler (1998) introduce a subclass of TAG (, which we will call *SS-TAG*) and show that ss-tals are recognizable in $O(n^5)$ time. The definition of ss-tag is slightly more general than that of esl-tag while keeping the constraint such that there exists (at most) one active node in the backbone. We conjecture that the generative power of ESL-TAG, SS-TAG and (2,2)-MCFG with $\deg(G) \leq 5$ are all the same.

Secondary structure is represented by a derivation (or derived) tree (see Figures 4 and 5). Comparison of the tree generative power of esl-tag and rpg is an interesting problem. To apply these grammars to RNA structure prediction, a probabilistic model should be introduced by extending these grammars such as stochastic cfg (Durbin et al., 1998), which is left as future work.

Acknowledgments

The authors would like to express their thanks to Professor S. Kanaya of Nara Institute of Science and Technology for his valuable discussions. The authors also thank Professor K. Asai of the University of Tokyo, Professor Y. Sakakibara of Keio University and members of his laboratory for their helpful comments.

References

Tatsuya Akutsu. 2000. *Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots*. Discrete Applied Mathematics, 104:45–62.

Anne Condon. 2003. *Problems on RNA secondary structure prediction and design*. ICALP2003, LNCS 2719:22–32.

Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. 1998. *Biological Sequence Analysis*. Cambridge University Press.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. *Tree adjunct grammars*. J. Computer & System Sciences, 10(1):136–163.

Aravind K. Joshi and Yves Schabes. 1997. *Tree adjoining grammars* in Grzegorz Rozenberg and Arto Salomaa, Eds., Handbook of Formal Languages, volume 3 (Beyond Words):69–123. Springer.

Aravind K. Joshi, K. Vijay-Shanker, and David J. Weir. 1988. *The convergence of mildly context-sensitive grammar formalisms*. Institute for Research in Cognitive Science, University of Pennsylvania.

Tadao Kasami, Hiroyuki Seki, and Mamoru Fujii. 1988. *Generalized context-free grammar and multiple context-free grammar*. IEICE Trans., J71-D(5):758–765 (in Japanese).

Tadao Kasami, Hiroyuki Seki, and Mamoru Fujii. 1988. *On the membership problem for head languages and multiple context-free languages*. IEICE Trans., J71-D(6):935–941 (in Japanese).

Yuki Kato, Hiroyuki Seki, and Tadao Kasami. 2004. *On the generative power of grammars for RNA secondary structure*. IEICE Technical Report, COMP-2003-75.

Alexandru Mateescu and Arto Salomaa. 1997. *Aspects of classical language theory* in Grzegorz Rozenberg and Arto Salomaa, Eds., Handbook of Formal Languages, volume 1 (Word, Language, Grammar):175–251. Springer.

Owen Rambow and Giorgio Satta. 1994. *A two-dimensional hierarchy for parallel rewriting systems*. IRCS Report 94-02, Institute for Research in Cognitive Science, University of Pennsylvania.

Elena Rivas and Sean Eddy. 2000. *The language of RNA: A formal grammar that includes pseudoknots*. Bioinformatics, 16(4):334–340.

Giorgio Satta and William Schuler. 1998. *Restrictions on tree adjoining languages*. Proc. 17th Int'l. Conf. on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING/ACL98).

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. *On multiple context-free grammars*. Theoretical Computer Science, 88:191–229.

Yasuo Uemura, Aki Hasegawa, Satoshi Kobayashi, and Takashi Yokomori. 1999. *Tree adjoining grammars for RNA structure prediction*. Theoretical Computer Science, 210:277–303.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1986. *Tree adjoining and head wrapping*. Proc. 11th Intl. Conf. on Computational Linguistics (COLING86):202–207.

K. Vijay-Shanker, David J. Weir and Aravind K. Joshi. 1987. *Characterizing structural descriptions produced by various grammatical formalisms*. Proc. 25th Annual Meeting of the Association for Computational Linguistics (ACL87).

SuperTagging and Full Parsing

Alexis Nasr
Lattice
Université Paris 7
Paris, France
alexis.nasr@linguist.jussieu.fr

Owen Rambow
Department of Computer Science
Columbia University
New York, NY, USA
rambow@cs.columbia.edu

Abstract

We investigate an approach to parsing in which lexical information is used only in a first phase, supertagging, in which lexical syntactic properties are determined without building structure. In the second phase, the best parse tree is determined without using lexical information. We investigate different probabilistic models for adjunction, and we show that, assuming hypothetically perfect performance in the first phase, the error rate on dependency arc attachment can be reduced to 2.3% using a full chart parser. This is an improvement of about 50% over previously reported results using a simple heuristic parser.

1 Introduction

Over the last ten years, there has been a great increase in the performance of parsers. Current parsers use the notion of lexical head when generating phrase structure parses, and use bilexical dependencies – probabilities that one particular head depends on another – to guide the parser. Current parsers achieve an score of about 90% when measuring just the accuracy of choosing these dependencies (Collins, 1997; Chiang, 2000; Clark et al., 2002; Hockenmaier and Steedman, 2002). Interestingly, the choice of formalism (headed CFG, TAG, or CCG) does not greatly change the parsers' accuracy, presumably because in all approaches, the underlying information is the same – word-word dependencies.

Supertagging followed by "lightweight" parsing has been proposed as an alternative to full parsing. The idea behind supertagging (Bangalore and Joshi, 1999) is to extend the notion of "tag" from a part of speech or a part of speech including morphological information to a tag that represents rich syntactic information as well, in particular active valency including subcategorization (who can/must be my dependents?), passive valency (who can be my governor?), and notions specific to particular parts of speech, such as voice. If words in a string can be tagged with this rich syntactic information in a supertag, then, Bangalore and Joshi (1999) claim, the remaining step of determining the actual syntactic structure is trivial. They propose a "lightweight dependency parser" (LDA) which is a heuristically-driven, very simple program that creates a dependency structure from the sequence of supertags. It uses no information gleaned from corpora at all, and performs with an (unlabeled) accuracy of about 95%, given the correct supertag.

The question arises how much better we can do if we use a more sophisticated way of determining the parse from the supertags, such as a chart parser. Of course, we do not want to give up the notion of a parsing stage which is relatively simple. In this paper, we extend the parsing stage by using a chart parser and probabilistic models, but we use only models that relate supertags to each other. In fact, such models are also used during supertagging, except that in supertagging, the only relation between supertags we are interested in modeling probabilistically is linear precedence, while for parsing we will use structural dependency as well. Thus, our approach conservatively extends the supertagging-and-LDA approach, and remains quite different from the current work on parsing based on bilexical probability models following (Collins, 1997). A secondary question we investigate in this paper is the issue of how best to model multiple adjunctions at a same node.

The paper is structured as follows. In Section 2, we provide some more motivation for this work. We present our formalization of TAG and discuss how to derive such a grammar formalized in that way from a corpus in Section 3. We present the parser in Section 4. In Section 5 we discuss three different ways in which we we estimate parameters for the statistical models. In Section 6, we present two baselines, and our main results. We discuss related work in Section 7, and conclude in Section 8.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 56-63.

2 Motivation

In this paper, we assume we have the correct supertag and we investigate the quality of the resulting parse. The state of the art in supertagging is currently in the 80%-85% range, depending on the grammar (see for example (Chen, 2001)). Thus, the task we set out to examine is not a realistic "real-world" task, and the question arises why we should be interested in the question at all. In this section, we try to motivate our research agenda by first arguing why supertag-based non-lexical parsing is interesting, and then by arguing why we need to show it is feasible.

2.1 Supertag-Based Parsing Is Interesting

There are several reasons to investigate supertag-based parsing. The main point is that the models involved are potentially simpler than those in bilexical parsing: no bilexical structural information is needed for deriving the parsing model. This holds the promise that when porting a supertagger-based parser to a new domain, a non-lexical structural model can be reused from a previous domain, and only a supertagged corpus in the new domain is needed (to train the supertagger), not a structurally annotated corpus.

Furthermore, this approach uses an explicit lexicalized grammar. As a consequence, when porting a parser to a new domain, learned parser preferences in the supertagger can be overridden explicitly for domain-idiosyncratic words before the parse happens. This overriding can happen through manually written or learned rules. By way of anecdotal example, in a recent application of the parser of Collins (1997) in which the WSJ-trained parser was applied to rather different text, sentences such as *John put the book on the table* were mostly analyzed with the PP attached to the noun, not the verb (as was always required in that domain). In the application, this had to be fixed by writing special post-processing code to rearrange the output of the parser; in our approach, we could simply state that *put* should always have a PP argument *before* the parse, and correct any output of the supertagger using simple hand-written rules.

And finally, we point out that is is a different approach from the dominant bilexical one, and it is always worthwhile to pursue new approaches, especially as the performance of the bilexical parsers seems to be plateauing. In fact recent work has questioned to what extent bilexical parsers even profit from bilexical information that they use (Gildea, 2001; Klein and Manning, 2003).

2.2 But Is Supertag-Based Parsing Feasible?

Bangalore and Joshi (1999) claim that supertagging is "almost parsing". What this means is that the syntactic information provided by supertags is so rich that there is little structural ambiguity left and the parse is almost entirely determined by the supertags. In fact, since the supertags determine both active and passive valency, the only remaining ambiguity is related to attachment of trees to nodes with the same label; for example, the standard PP-attachment ambiguity of *see a man with a telescope* is resolved in the supertags, as the tag for *with* specified adjunction to a VP or an NP. The remaining issues of structural ambiguity which are not resolved by supertags include conjunctions, noun-noun compounds (which however are not given meaningful analyses in the PTB), attachment of adjuncts in sentences with several clauses (such as *John told Mary to leave today*), and so on.

There is thus both a practical and a theoretical interest in knowing how much ambiguity remains after supertagging, or, put differently, to what extent supertagging is in fact "almost parsing". Practically, the performance of a parser with correct supertags as input gives us an upper bound on supertag-based parsing. The current figure of 95% (using the heuristic LDA) may seem a bit low as an upper bound. Theoretically, it is interesting to know to what extent, in a corpus, syntactic structure is disambiguated by specifying both active and passive valency of words.

3 Representing a TAG as a Set of FSMs

For the purpose of our parser, we represent a Tree Adjoining Grammar as a set of finite-state machines (FSMs). The FSMs form a (lexicalized) Recursive Transition Network (RTN). To extract an RTN from the Penn Treebank (PTB), we first extract a TAG, and then convert it to an RTN. This first step does not represent the research reported in this paper, and we describe it only for the sake of clarity. We use the approach of (Chen, 2001) (which is similar to (Xia et al., 2000) and (Chiang, 2000)). We use sections 02 to 21 of the Penn Treebank. However, we optimize the head percolation in the grammar extraction module to create meaningful dependency structures, rather than (for example) maximally simple elementary tree structures. For example, we include long-distance dependencies (*wh*-movement, relativization) in elementary trees, we distinguish passive transitives without *by*-phrase from active intransitives, and we include strongly governed prepositions (as determined in the PTB annotation, including passive *by*-phrases) in elementary verbal trees. Generally, function words such as auxiliaries or determiners are dependents of the lexical head,[1] conjunctions (including punctuation functioning as conjunction) are dependent on the first conjunct and take the second conjunct as their argument, and conjunction chains are represented as right-branching rather than flat.

[1]This is a linguistic choice and not forced by the formalism or the PTB. We prefer this representation as the resulting dependency tree is closer to predicate-argument structure.

In the second step, we directly compile a set of FSMs which are used by the parser. To derive a set of FSMs from a TAG, we do a depth-first traversal of each elementary tree in the grammar (but excluding the root and foot nodes of adjunct auxiliary trees) to obtain a sequence of nonterminal nodes. As usual, the elementary trees are tree schemas, with positions for the lexical heads. Substitution nodes are represented by obligatory transitions, adjunction by optional transitions (self-loops). (Note that in this paper, we assume adjunction as defined by Schabes and Shieber (1994).) Each node becomes two states of the FSM, one state representing the node on the downward traversal on the left side (the **left node state**), the other representing the state on the upward traversal, on the right side (the **right node state**). For leaf nodes, the two states immediately follow one another. The states are linearly connected with ϵ-transitions, with the left node state of the root node the start state, and its right node state the final state (except for predicative auxiliary trees – see below). We give a sample grammar in Figure 1 and the result of converting one of its trees to an FSM in Figure 2.

For each pair of adjacent states representing a substitution node, we add transitions between them labeled with the names of the trees that can substitute there. For the lexical head, we add a transition on that head. For footnodes of predicative auxiliary trees which are left auxiliary trees (in the sense of Schabes and Waters (1995), i.e., all nonempty frontier nodes are to the left of the footnode), we take the left node state as the final state. Finally, in the basic model in which adjunctions are modeled as independent, we proceed as follows for non-leaf nodes. (In Section 5, we will see two other models that treat non-leaf nodes in a more complex manner.) To each non-leaf state, we add one self loop transition for each tree in the grammar that can adjoin at that state from the specified direction (i.e., for a state representing a node on the downward traversal, the auxiliary tree must adjoin from the left), labeled with the tree name. There are no other types of leaf nodes since we do not traverse the passive valency structure of adjunct auxiliary tees. The result of this phase of the conversion is a set of FSMs, one per elementary tree of the grammar, whose transitions refer to other FSMs.

Note that the treatment of footnodes makes it impossible to deal with trees that have terminal, substitution or active adjunction nodes on both sides of a footnode. It is this situation (iterated, of course) that makes TAG formally more powerful than CFG; in linguistic uses, it is very rare, and no such trees are extracted from the PTB.[2] As a result, the grammar is weakly equivalent to a CFG. In fact, the construction treats a TAG as if were a Tree Insertion Grammar (TIG, Schabes and Waters (1995)), or rather, it coerces a TAG to be a TIG: during the traversal, both terminal nodes and nonterminal (i.e., substitution) nodes between the footnode and the root node are ignored (because the traversal stops at the footnode), thus imposing the constraint that the trees may not be wrapping trees and that no further adjunction may occur to the right of the spine in a left auxiliary tree.

4 Parsing with FSMs

The parsing algorithm is a simple extension of the chart parsing algorithm for CFG. The difference is in the use of finite state machines in the items in the chart. In the following, we will call t-**FSM** an FSM M if it is derived from tree t in the original TAG (or TIG) grammar G. If T is the parse table for input sentence W and GDG G, then $T_{i,j}$ contains (M, q) where M is a t-FSM, and q is one of the final states of M, iff we have a complete derivation of substring $w_i \cdots w_j$ in G such that the root of the derivation tree is labeled t.

Before starting the parse, we create a tailored grammar by selecting those trees associated with the words in the input sentence, and substituting the actual words for the positions of the lexical head. (Note that the crucial issue is how to associate trees with words in a sentence; in this paper, we assume that the correct tree is used.)

Initialization: We start by adding, for each i, $1 \leq i \leq n$, w_i to $T_{i,i}$.

Completion: If $T_{i,j}$ contains either the input symbol w or an item (M, q) such that q is a final state of M, and M is a t-FSM, then add to $T_{i,j}$ all (M', q') such that M' is a FSM which transitions from a start state to state q' on input w or t.

Add a single backpointer from (M', q') in $T_{i,j}$ to (M, q) or w in $T_{i,j}$.

Scanning: If (M_1, q_1) is in $T_{i,k}$, and $T_{k+1,j}$ contains either the input symbol w or the item (M_2, q_2) where q_2 is a final state and M_2 is a t-FSM, then add (M_1, q) to $T_{i,j}$ (if not already present) if M_1 transitions from q_1 to q on either w or t.

Add a double backpointer from (M_1, q) in $T_{i,j}$ to (M_1, q_1) in $T_{i,k}$ (left backpointer) and to either w or (M_2, q_2) in $T_{k+1,j}$ (right backpointer).

Note that because we are using a dependency grammar, each scanning step corresponds to one attachment of a lexical head to another. At the end of the parsing process, a packed parse forest has been built. The nonterminal nodes are labeled with pairs (M, q) where M is an FSM and q a state of this FSM. Obtaining the dependency trees from the packed parse forest is performed

[2] Our construction cannot handle Dutch cross-serial dependencies (not surprisingly), but it can convert the TAG analysis of *wh*-movement in English and similar languages, because the predicative auxiliary verbal trees do not have terminal or substitution nodes on both sides of the foot node.

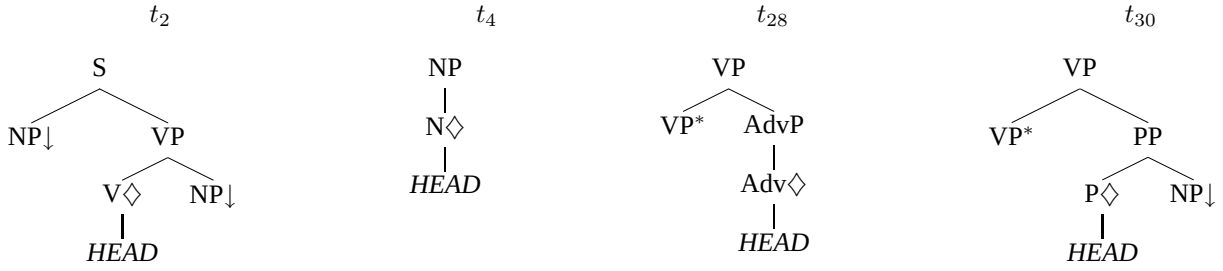


Figure 1: Sample small grammar: trees for a transitive verb, a nominal argument, and two VP adjuncts from the right

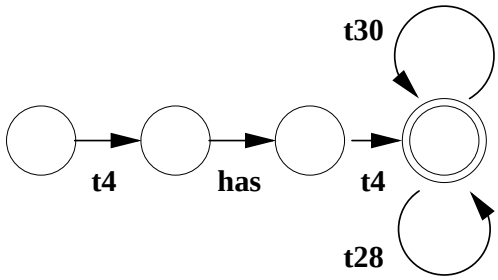


Figure 2: FSM derived according to Model 1 for tree t_2 from the grammar in Figure 1, instantiated for the verb *has*

in two stages. In a first stage, a forest of binary phrase-structure trees is obtained from the packed forest and in a second stage, each phrase-structure tree is transformed into a dependency tree.

5 Probabilistic Models

The parser introduced in Section 4 associates to a supertag sequence $S = S_1 \dots S_n$ one or several analyses. Each analysis $\mathcal{A}$ can be seen as a set of $n-1$ attachment operations and the selection of one supertag token as the root of the analysis (the single supertag that is not attached in another supertag). For the sake of uniformity, we will consider the selection of the root as a special kind of attachment, $\mathcal{A}$ is therefore of cardinality n. In the following, $LEFT(x, y)$ (respect. $RIGHT(x, y)$) denotes the set of attachments that occurred on the left (respect. right) side of node y of supertag x. For an attachment operation A, $O(A)$ returns its type (adjunction, substitution, root). *Root* designates the unique event in $\mathcal{A}$ that selects the root.

From a probabilistic point of view, each attachment operation is considered as an event and an analysis $\mathcal{A}$ as the joint event $A_1, \dots, A_n$. A large range of different models can be used to compute such a joint probability, from the simplest which considers that all events are independent to the model that considers that they are all dependent. The three models that we descibe in this section vary in the way they model multi-adjunction (when several auxiliary trees are attached to a single node from the same direction). The reason to focus on this phenomenon comes from the fact that it is precisely at this level that much of the structural ambiguity occurs. For example, in a sentence containing three or more conjoined NPs (such as *dogs, hamsters, cats, and bats*), there is massive ambiguity of attachment, as each conjunction can attach to any of the preceding nouns. However, only one structure (in our corpus, the right-branching one) is correct. Thus, a precise model is needed. The three models described below consider that substitution operations are independent of all the other attachments that make up an analysis. The general model is therefore:

$$\begin{aligned} P(\mathcal{A}) &= P(Root) \\ &\times \prod_{A \in \mathcal{A} | O(A) = subst} P(A) \\ &\times \prod_{s \in S, i \in nodes(s)} P(LEFT(s,i)) \\ &\times \prod_{s \in S, i \in nodes(s)} P(RIGHT(s,i)) \end{aligned}$$

This basically follows (Resnik, 1992; Schabes, 1992). The models we discuss here differ in how to compute the terms $P(RIGHT(s,i))$ and $P(LEFT(s,i))$.

The probability of each attachment is estimated by maximum likelihood (the counts are obtained in the same step as the grammar extraction), and are added to the corresponding transition in the governor's automaton as its weight. When the probabilistic model associates different

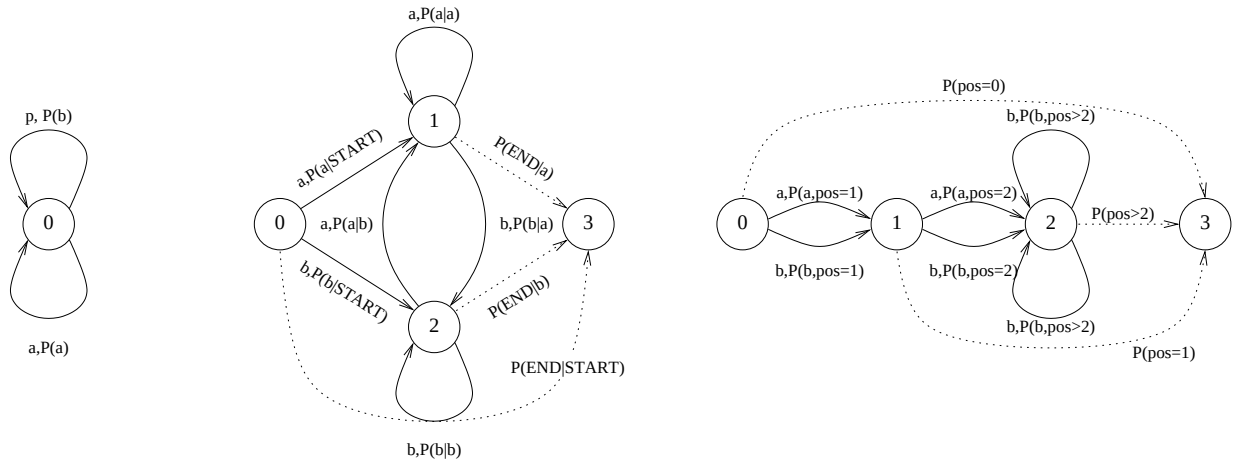


Figure 3: Three models of adjunction; these correspond to the last node of the FSM in Figure 2, with the model on the left exactly as shown in Figure 2; here, a represents t_{30} of Figure 2 and b, t_{28}

probabilities to attachments that were not distinguished in an automaton, the structure of the latter will be changed in order to account for this difference. This change in the structure will, of course, leave unchanged the language recognized by the automaton . The three models for adjunction will be illustrated on a simple example where two supertags a and b are candidate for adjunction at a given node. In the following models, we estimate parameters from the corpus obtained by running the TAG extraction algorithm over the PTB training corpus. We can then easily count the relevant events.

5.1 Model 1: Independent Adjunctions

In this model, an adjunction on one node is considered independent from the other adjunctions that can take place on the same node. The probability of each adjunction depends on the dependent supertag, on the governor supertag, and on the node of the governor supertag at which the attachment takes place. However, it is independent of the order of the attachment. The model does therefore not distinguish between attachments that only differ in their order. This model corresponds to the left part of figure 3, the attachment of an a, for example, does not depend on what was attached before and how many attachment took place. For example, the probability of the sequence $abab$ being adjoined is modeled as follows (we use here and subsequently a simplified notation where $P(a)$ designates the adjunction of a at the relevant node in the relevant tree):

$$P(abab) = P(a)P(b)P(a)P(b)$$

5.2 Model 2: Positional Model

This model adds to the first one the knowledge of the *order* of an attachment. But when modeling the probability that supertag a attaches on a given node at order i, it does not take into account the attachments that happened for $order < i$. Such models also add a new parameter which is the maximum number of attachment that are distinguished. The graphical representation of the model as a finite state automaton, as it appears to the right in Figure 3, gives an intuitive account of the nature of the model. It is made of a series of transitions between consecutive pairs of nodes. The first "bundle" of transitions models the first attachment on the node, the second bundle, the second attachment, and so on, until the maximum number of attachments is reached. This limit on the number of attachments concerns only the probabilistic part of the automaton, more attachment can occur on this node, but their probabilities will not be distinguished. These attachments correspond to the loops on state 2 of the automaton. ϵ-transitions allow the attachments to stop at any moment by transitioning to state 3. (The ϵ-transitions are shown as dotted lines for reading convenience, they are formally regular transitions in the FSM.) Under Model 2, the probability of the sequence $abab$ being adjoined is:

$$\begin{aligned} P(abab) &= P(a, pos = 1) \\ &\times P(b, pos = 2) \\ &\times P(a, pos > 2) \\ &\times P(b, pos > 2) \end{aligned}$$

5.3 Model 3: N-Gram Model

The previous model takes into account the order of an attachment and disregards the nature of the attachments that happened before (or after) a given attachment. The model described here is, in a sense, complementary to

the previous one since it takes into account, in the probability of an attachment, the nature of the attachment that occurred just before and ignores the order of the current attachment. The probability of a series of attachments on the same side of the same node will be computed by an order-1 Markov chain, represented as a finite state automaton in the central part of Figure 3. The transitions with probabilities $P(x|START)$ (respect. $P(END|x)$) correspond to the occurrence of supertag x as the first (respect. the last) attachment at this node and the transition with probability $P(END|START)$ corresponds to the null adjunction (the probability that no adjunction occurs at a node). The probability of the sequence $abab$ being adjoined is now:

$$\begin{aligned} P(abab) &= P(a|START) \\ &\times P(b|a) \\ &\times P(a|b) \\ &\times P(b|a) \\ &\times P(END|b) \end{aligned}$$

5.4 Finding the n-best parses

We extend our parser by augmenting entries in the parse table with probabilities. As usual, only the highest probability is retained for a given analysis. The algorithm for extracting parses is augmented to choose the best parse (or n-best parses) in the usual manner. Note that the different models discussed in this section only affect the manner in which the TAG grammar extracted from the corpus is converted to an FSM; the parsing algorithm (and code) is always the same.

6 Results

In this study, we are interested in exploring how parsing performs in the presence of the correct supertag. As a result, in the following, we report on data which has been correctly supertagged. We used Sections 02 to 21 of the Penn Treebank for training, the first 800 sentences of Section 00 for development, and Section 23 for testing only. The figures we report are accuracy figures: we evaluate how many dependency relations have been found. The root node is considered to have a special dependency relation. There is no need to report recall and precision, as each sentence always has a number of dependency relations which is equal to the number of words. In the evaluation, we disregard true (non-conjunction) punctuation. The figures for the LDA are obtained by using the LDA as developed previously by Bangalore Srinivas, but using the same grammar we used for the full parser. Note that none of the numbers reported in this section can be directly compared to any numbers reported elsewhere, as this task differs from the tasks discussed in other research on parsing.

We use two different baselines. First, we use the performance of the LDA of (Bangalore and Joshi, 1999). The performance of the LDA on Section 00 is about 94.3%, on Section 23 95.1%. Second, we use the full chart parser, but randomly choose a parse from the parse forest. This baseline measures to what extent using a probabilistic model in the chart parser actually helps. The performance of this baseline is 94.7% on Section 00, 94.6% on Section 23. As we can see, the supertags provide sufficient information to result in high baselines. The results are summarized in Figure 4.

There are several clear conclusions to be drawn from Figure 4. First, a full parse has advantages over a heuristic parse, as even a random choice of a tree from the parse forest in the chart (i.e., without use of a probabilistic model) performs nearly as well as the heuristic LDA. Second, the use of even a simple probabilistic model using no lexical probabilities at all, and modeling adjunctions as entirely independent, reduces the error rate over the non-probabilistic baseline by 22.8%, to 4.04%. Third, the modeling of multiple adjunctions at one node as independent is not optimal, and two different models can further reduce the error rate substantially. Specifically, we can increase the error reduction to 53.0% by modeling the first adjunction (from left to right) separately from all subsequent ones. However, presumably due to sparseness of data, there is no major advantage to using more than one position (and modeling the first and second adjunction separately). Furthermore, switching to the n-gram model in which an adjunction is conditioned on the previously adjoined supertag as well as the governing supertag, the error reduction is further increased slightly to 56.6%, with an error rate of 2.27%. This is the best result obtained on the development corpus.

7 Related Work

We are not aware of any other work that directly investigates the extent to which supertagging determines parsing. Chiang (2000) also parses with an automatically extracted TIG, but unlike our approach, he uses standard TAG/TIG parsing techniques (i.e., he reconstructs the derived tree in the chart, not the derivation tree). Rogers (1994) proposes a different context-free variant, "regular-form TAG". The set of regular-form TAGs is a superset of the set of TIGs, and our construction cannot capture the added expressive power of regular-form TAG. Our conversion to FSMs is very similar to that of Evans and Weir (1997). One important difference is that they model TAG, while we model TIG. Another difference is that they use FSMs to encode the sequence of actions that need to be taken during a standard TAG parse (i.e., reconstructing the derived tree), while we encode

Method	Accuracy on Sec 00	Accuracy on Sec 23
Baseline: LDA	94.35%	95.14%
Baseline: full parse with random choice	94.73%	94.69%
Model 1 (Independent Adjunction)	95.96%	
Model 2 (Positional Model): 1 position	97.54%	
Model 2 (Positional Model: 2 position	97.49%	
Model 2 (Positional Model: 3 position	97.57%	
Model 3 (N-Gram Model), using Supertag	97.73%	97.61%
Model 3 (N-Gram Model), using Category	97.29%	

Figure 4: Results (accuracy) for different models using the Gold-Standard supertag on development corpus (Section 00, first 800 sentences) with add-0.001 smoothing, and for the best performing model as well as the baselines on the test corpus (Section 23)

the active valency of the lexical head in the FSM. A result, in retrieving the derivation tree, each item in the parse tree corresponds to an attachment of one word to another, and there are fewer items. Furthermore, our FSMs are built left-to-right, while Evans and Weir only explore FSMs constructed bottom-up from the lexical anchor of the tree (not unlike (Eisner, 2000)). As a result, we can perform a strict left-to-right parse, which is not straightforwardly possible in standard TAG parsing using FSMs.

Our parsing algorithm is similar to the work of Alshawi et al. (2000). They use cascaded head automata to derive dependency trees, but leave the nature of the cascading under-formalized. Eisner (2000) provides a formalization of a system that uses two different automata to generate left and right children of a head. His formalism is very close to the one we present, but we use a single automaton. Also, the relation to an independently proposed syntactic formalism such as TAG is less obvious.

In related work (Rambow et al., 2002), we have used the same automata constructed from an extracted TAG for parsing, but instead of using them in a chart parser, we have used them to construct a single large FSM that produces a dependency tree. Needless to say, the number of embeddings allowed by such an approach is limited.

8 Conclusion

We have provided further evidence for the claim of Bangalore and Joshi (1999) that supertagging is "almost parsing", and we have quantified the "almost" to be 97.7%.[3] This figure represents the dependency accuracy that can be obtained when the input is represented as a sequence of supertags, with no lexical information used in the parse (and hence not in the training of the parser, either). This shows that an architecture is viable in which *all* information related to the specific lexemes is assigned in a first pass before structure is constructed, and structure is constructed only in a second pass in which *no* lexical information is used (other than the lexical emit probability for supertags). This result motivates further research into supertagging accuracy. If supertagging accuracy is improved, a lightweight parser in conjunction with supertagging may perform as well as a full bilexical parser, or even better. Furthermore, for certain applications, a lightweight parser may be appealing because only the supertagger needs to be retrained which can be done with less effort. Finally, the explicit and declarative nature of the grammar used makes it easy to write hand-written rules to override the supertagger in cases in which the application designer wishes to correct a systematic parser error.

[3] We note that this figure holds for the particular grammar that we used; other grammars may result in different figures.

References

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. 2000. Learning dependency translation models as collections of finite-state head transducers. *cl*, 26(1):45–60.

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–266.

John Chen. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *38th Meeting of the Association for Computational Linguistics (ACL'00)*, pages 456–463, Hong Kong, China.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage ccg parser. In *acl02*, pages 327–334.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Madrid, Spain, July.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry C. Bunt and Anton Nijholt, editors, *New Developments in Natural Language Parsing*. Kluwer Academic Publishers.

Roger Evans and David Weir. 1997. Automaton-based parsing for lexicalized grammars. In *5th International Workshop on Parsing Technologies (IWPT97)*, pages 66–76.

Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP01)*, pages 167–202, Pittsburgh, PA.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *acl02*, pages 335–342.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *41st Meeting of the Association for Computational Linguistics (ACL'03)*.

Owen Rambow, Srinivas Bangalore, Tahir Butt, Alexis Nasr, and Richard Sproat. 2002. Creating a finite-state parser with application semantics. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Republic of China.

Philip Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING '92)*, Nantes, France, July.

James Rogers. 1994. Capturing cfls with Tree Adjoining Grammars. In *32nd Meeting of the Association for Computational Linguistics (ACL'94)*. ACL.

Yves Schabes and Stuart Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 1(20):91–124.

Yves Schabes and Richard C. Waters. 1995. Tree Insertion Grammar: A cubic-time, parsable formalism that lexicalizes Context-Free Grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–514.

Yves Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*.

Fei Xia, Martha Palmer, and Aravind Joshi. 2000. A uniform method of grammar extraction and its applications. In *Proc. of the EMNLP 2000*, Hong Kong.

Computing Semantic Representation: Towards ACG Abstract Terms as Derivation Trees

Sylvain Pogodalla
LORIA - Campus Scientifique
BP239
F-54602 Vandœuvre-lès-Nancy – France
Sylvain.Pogodalla@loria.fr

Abstract

This paper proposes a process to build semantic representation for Tree Adjoining Grammars (TAGs) analysis. Being in the derivation tree tradition, it proposes to reconsider derivation trees as abstract terms (λ-terms) of Abstract Categorial Grammars (ACGs). The latter offers a flexible tool for expliciting compositionality and semantic combination. The chosen semantic representation language here is an underspecified one. The ACG framework allows to deal both with the semantic language and the derived tree language in an equivalent way: as concrete realizations of the abstract terms. Then, in the semantic part, we can model linguistic phenomena usually considered as difficult for the derivation tree approach.

Introduction

When dealing with the computation of semantic representation for TAG analysis, two main approaches are usually considered. The first one gives the derivation trees a central role for the computation (Schabes and Shieber, 1994; Candito and Kahane, 1998; Kallmeyer, 2002; Joshi et al., 2003), and the second one relies on a direct computation on the derived tree (Frank and van Genabith, 2001; Gardent and Kallmeyer, 2003).

The present article wants to explore the intuition that the two approaches are indeed bound: derivation trees are a specification of the operations that are to be processed, but the derived trees hold the precise descriptions of these operations. We propose to exhibit those operations by separating them from the syntactic trees. Then, under the specifications given by the derivation trees, we show how to build the semantic representations.

The tools we use for this purpose are Abstract Categorial Grammars (ACGs) (de Groote, 2001). The main feature of an ACG is to generate two languages: an *abstract language* and an *object language*. Whereas the abstract language may appear as a set of grammatical or parse structures, the object language may appear as its realization, or the concrete language it generates. For instance, (de Groote, 2002) proposes as object language the tree language of TAGs (encoded in linear λ-terms) and, as abstract language, a tree language (also encoded in linear λ-terms) and *very close to the derivation tree language*. In this paper, we use the same abstract language, and, as object language, λ-terms that encode underspecified semantic representation as in (Bos, 1995; Blackburn and Bos, 2003). Thus, we realize our program to separate the computation specification and the operation definition. As for Montague's semantics, missing information is represented by bound λ-variables and replacement and variable catching by application instead of unification (as in (Frank and van Genabith, 2001; Gardent and Kallmeyer, 2003)).

The next section briefly describes the underlying principles of ACGs. Then we show how syntactic parts of TAGs are modelled and how we translate, through the abstract terms (our derivation trees), the combination of intial and auxiliary trees to their semantic representations by means of some examples.

1 ACG Principles

An ACG $\mathcal{G}$ defines:

1. two sets of typed λ-terms: Λ_1 (based on the typed constant set C_1) and Λ_2 (based on the typed constant set C_2);
2. a morphism $\mathcal{L} : \Lambda_1 \rightarrow \Lambda_2$;
3. a distinguished type $\mathbf{S}$.

(de Groote, 2001) defines both Λ_1 and Λ_2 as sets of *linear* λ-terms. In this paper, we use simply typed λ-terms for Λ_2, using the translation of intuitionnistic logic into

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 64-71.

linear logic $A \to B = (!A) \multimap B$ (Girard, 1987; Danos and Cosmo, 1992). We don't elaborate on that subject in this paper, but it does not change the main properties of ACGs[1]. Then the abstract language $\mathcal{A}(\mathcal{G})$ and the object language $\mathcal{O}(\mathcal{G})$ are defined as follows:

$$\mathcal{A}(\mathcal{G}) = \{t \in \Lambda_1 | t : \mathbf{S}\}$$
$$\mathcal{O}(\mathcal{G}) = \{t \in \Lambda_2 | \exists u \in \mathcal{A}(\mathcal{G})\ t = \mathcal{L}(u)\}$$

Note that $\mathcal{L}$ binds the parse structures in $\mathcal{A}(\mathcal{G})$ to the concrete expressions of $\mathcal{O}(\mathcal{G})$. Depending on the choice of Λ_1, Λ_2 and $\mathcal{L}$, it can map for instance derivation trees and derived trees for TAGs (de Groote, 2002), derivation trees of context-free grammars and strings of the generated language (de Groote, 2001), derivation trees of m-linear context-free rewriting systems and strings of the generated language (de Groote and Pogodalla, 2003). Of course, this link between an abstract and a concrete structure can apply not only to syntactical formalisms, but also to semantic formalisms.

The main point here is that ACGs can be mixed in different ways: in a transversal way, were two ACGs use the same abstract language, or in a compositional way, were the abstract language of an ACG is the object language of an other one. In this paper, as described in figure 1, we use different ACGs and some composition with $\mathcal{G}$: Λ_2 is the tree language of TAGs, Λ_1 the tree language of our *derivation trees*. For $\mathcal{G}'$, we have the same abstract language and Λ'_2 is the underspecified representation language. In dotted lines is a composition presented in (de Groote, 2001) between strings and derivation trees we do not use here.

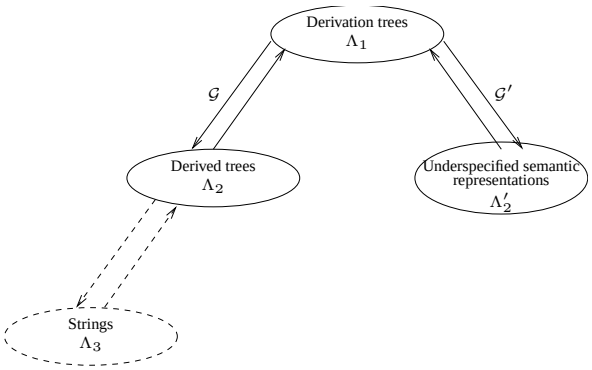


Figure 1: Moving from an object language to another

[1]In particular, this means that, provided there is no vacuous abstraction in $\mathcal{L}(C_1)$ and every $c \in \mathcal{L}(C_1)$ is such that it has $t \in C_2$ as subterm, we can decide if, for $u \in \Lambda_2$ if $u \in \mathcal{O}(\mathcal{G})$ and what is (are) the antecedent(s) (Pogodalla, 2004).

2 TAGs as ACGs

This section refers to (de Groote, 2002), which proposes to encode TAGs into ACGs. Given a TAG G, Λ_1 is build as follows:

- for every non-termninal symbol X, there are two types X_S and X_A standing for places where substitution and adjunction can occur respectively;
- for every elementary tree γ, there is a constant $c_\gamma \in C_1$. Moreover, for every non-terminal symbol X, there is a constant $I_X : X_A$.

For instance, given the trees of table 1, we have the constants and their types (for concision, we suppress parameters that are not used in the next examples of this paper, namely nodes where no adjunction occur[2]):

$$\begin{aligned} c_{\textit{every}} &: \mathbf{N}_A \\ c_{\textit{dog}} &: \mathbf{N}_A \multimap \mathbf{N}_S \\ c_{\textit{chases}} &: \mathbf{S}_A \multimap \mathbf{VP}_A \multimap \mathbf{N}_S \multimap \mathbf{N}_S \multimap \mathbf{S}_S \\ c_{\textit{usually}} &: \mathbf{VP}_A \multimap \mathbf{VP}_A \end{aligned}$$

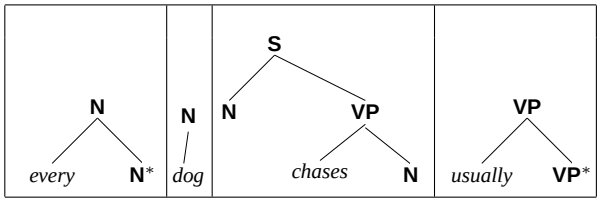


Table 1: Examples of elementary trees

To completely define the ACG $\mathcal{G}$, we need to define Λ_2 and $\mathcal{L}$. The types of Λ_2 are made of the single type τ, representing the type of trees. For any non-terminal symbol X, there are constants $X_0, \ldots, X_i$ where i is the maximal number of children of the X nodes in the elementary trees. For any terminal symbol X in G, there is a constant $X : \tau \in C_2$. Then $\mathcal{L}$ is defined by sending any X_S type to the type τ, and any X_A types to the type $\tau \multimap \tau$. Corresponding to the trees of table 1, we have for instance:

$$\begin{aligned} \mathcal{L}(I_X) &= \lambda x.x : \tau \multimap \tau \\ \mathcal{L}(c_{\textit{every}}) &= \lambda x.\mathbf{N_2}(\textbf{every}\ x) : \tau \multimap \tau \\ \mathcal{L}(c_{\textit{dog}}) &= \lambda N.N(\mathbf{N_1}\textbf{dog}) : (\tau \multimap \tau) \multimap \tau \\ \mathcal{L}(c_{\textit{chases}}) &= \lambda SV.\lambda x.\lambda y.S(\mathbf{S_2}x(V \\ &\quad (\mathbf{VP_2}\textbf{chases}\ y))) \\ &\quad : (\tau \multimap \tau) \multimap (\tau \multimap \tau) \multimap \tau \multimap \tau \multimap \tau \end{aligned}$$

Note that in the adjunction operation, the auxiliary tree is a parameter. But it also has a higher-order type, that

[2]For instance, the type of $c_{\textit{every}}$ should be $\mathbf{Det}_A \multimap \mathbf{N}_A \multimap \mathbf{N}_A$.

is a function from trees to trees. We let the reader check that $\mathcal{L}(c_{chases} I_{\mathbf{S}} I_{\mathbf{VP}}(c_{dog} c_{every})(c_{cat} c_{some}))$ correspond the derived tree associated to *every dog chases some cat* of figure 2.

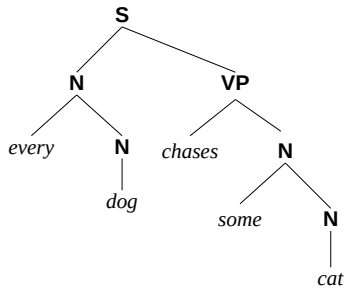


Figure 2: $\mathcal{L}(c_{chases} I_{VP}(c_{dog} c_{every})(c_{cat} c_{some})) = \mathbf{S}_2(\mathbf{N}_2\ \textit{every}\ (\mathbf{N}_1\ \textit{dog}\ ))(\mathbf{VP}_2\ \textit{chases}\ (\mathbf{N}_2\ \textit{some}\ (\mathbf{N}_1\ \textit{cat}\ )))$

We note two important things. First, the abstract terms, as $c_{chases} I_{\mathbf{S}} I_{\mathbf{VP}}(c_{dog} c_{every})(c_{cat} c_{some})$ can be represented by a tree structure where the children of a node are its arguments. Then erasing the I_X arguments, and directing the edges downward if the argument is of type X_S and upward if the argument is of type X_A, we get the usual notion of derivation tree. Second, the auxiliary trees are modelled as higher-order function. We use the same approach in our semantic modelling, getting some type raising, as in Montague's semantics. But let us precise the ACG we use for the semantic representation.

3 Semantic representation for TAGs as ACGs

The semantic representation language we use is an underspecified one presented in (Bos, 1995; Blackburn and Bos, 2003): the predicate logic "unplugged". The aim of this language, the *underspecified representation language* (URL) is to specify in a single formula the possible formulas (of the *semantic representation language* (SRL)) associated to an ambiguous expression. For instance, the expression *every dog chases a cat* has the two possible meanings:

$$\forall x(\mathbf{dog}(x) \Rightarrow \exists y(\mathbf{cat}(y) \wedge \mathbf{chases}(x,y)))$$
$$\exists y(\mathbf{cat}(y) \wedge \forall x(\mathbf{dog}(x) \Rightarrow \mathbf{chases}(x,y)))$$

To mark the difference between the SRL and the URL, both being first order languages, we translate the usual first order logic symbols of SRL. This translation is straightforward, using boldface symbols (e.g, **All**, **And**, **Imp**, etc.). In SRL, the two previous formulas are restated as follows:

$$\mathbf{All}(x, \mathbf{Imp}(\mathbf{dog}(x), \mathbf{Some}(y, \mathbf{And}(\mathbf{cat}(y), \mathbf{chases}(x,y)))))$$
$$\mathbf{Some}(y, \mathbf{And}(\mathbf{cat}(y), \mathbf{All}(x, \mathbf{Imp}(\mathbf{dog}(x), \mathbf{chases}(x,y)))))$$

Both these formulas have the property that:

- they have at leat two subformulas: one quantified by **All**, one quantified by **Some**;
- $\mathbf{chases}(x,y)$ is a subformulas of the two quantified subformulas.

The URL relies on the speficication of subformula constraints that the SRL formulas have to satisfy, and the two SRL formulas above can be described by the following URL formula:

$$\exists h_0 h_1 h_2 l_1 l_2 l_3 l_4 l_5 l_6 l_7 xy(l_1 : \mathbf{All}(x, l_2)$$
$$\wedge l_2 : \mathbf{Imp}(l_3, h_1) \wedge l_3 : \mathbf{dog}(x) \wedge l_4 : \mathbf{Some}(y, l_5)$$
$$\wedge l_5 : \mathbf{And}(l_6, h_2) \wedge l_6 : \mathbf{cat}(y)$$
$$\wedge l_7 : \mathbf{chases}(x,y) \wedge h_1 \geq l_7 \wedge h_2 \geq l_7 \wedge h_0 \geq l_1$$
$$\wedge h_0 \geq l_4)$$

illustrated in figure 3. The syntax of URL is basically the same that first-order logic, except that if atomic formulas remain the same, formulas are built from *holes* and *labels*, the latter being used as place holder for logical formulas in the underspecified representation language. We use the usual logical symbols ($\exists$, $\wedge$), an infix predicate $\geq$ to specify the constraints and an infix operateur : for URL. The symbol $h \geq l$ imposes the constraint for a formula that is associated to l to be a subformula of the one associated to h. $l : p$ indicates that a predicate p of SRL is labelled in URL by l.

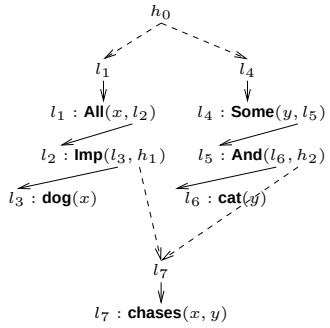


Figure 3: URL formula for *every dog chases a cat*

We want to underline the difference between URL and SRL because our concern in this paper is not to build and manage SRL formulas, but only URL formulas, that is underspecified representations. So that the object language of the ACG we are designing is URL.

Coming back to the figure 1, we established in the previous section the $\mathcal{G}$ ACG to encode TAGs. We know want to rely on the common abstract language, Λ_1, the one of derivation trees, to build the $\mathcal{G}'$ ACG that model the semantic behaviour, with URL as Λ'_2. So let us now define $\mathcal{G}'$.

First is Λ'_2:

- the types we use are e, h, l, p, t where e stands for entities, h for holes, l for labels, p for predicate of the logical language and t for truth values;
- the constants are $\geq$, :, $\exists_l$, $\exists_e$, $\exists_h$, $\wedge$, **Imp**, **And**, **Some**, **All** and the set of the predicate symbols of the logical language (**dog**, **chases**, etc. in the examples). Their types are described in table 2.

Note we have three existential quantifiers $\exists_l$, $\exists_h$ and $\exists_e$, but we usually note them only $\exists$. Moreover, to keep with the usual logical notation we write $\exists x\, P$ instead of $\exists(\lambda x.P)$ where x is a free variable of P.

Finally, to define the ACG $\mathcal{G}'$, we need the lexicon $\mathcal{L}'$. It transforms the types from Λ_1 as follows:

$$\begin{aligned}
\mathcal{L}'(\mathbf{N}_S) &= (e \to h \to l \to t) \multimap (h \to l \to t)\\
\mathcal{L}'(\mathbf{N}_A) &= (e \to h \to l \to t)\\
&\quad \multimap (e \to h \to l \to t) \multimap (h \to l \to t)\\
\mathcal{L}'(\mathbf{S}_S) &= h \to l \to t\\
\mathcal{L}'(\mathbf{S}_A) &= (e \to h \to l \to t)\\
&\quad \multimap (e \to h \to l \to t)\\
\mathcal{L}'(\mathbf{VP}_A) &= (h \to l \to t) \multimap (h \to l \to t)
\end{aligned}$$

Contrary to Λ_2, that model derived trees and generates linear terms, we use in Λ_2' non-linear terms, as the intuitionnistic $\to$ shows. The definition of $\mathcal{L}'$ on the terms justifies it. We shall introduce this definition in the next sections, illustrating different linguistic phenomena.

3.1 Quantification

We start with the classical example of quantitification. When dealing with quantifiers as adjunct (Abeillé, 1993), where quantifier is adjoined to the noun, quantifiers are separated from the verb by the noun in the derivation trees. Then the problem of the proposition coming from the **VP** to be part of the scope of the quantifiers arises. (Kallmeyer, 2002) proposes to enrich the derivation trees with additional links to take this kind of linking into account.

We propose to deal with this kind of problems following the Montague's approach of quantification (Montague, 1974): the subject is an argument of the verb, but it is also a higher order function which has the verb predicate as argument. So the lexicon for the ACG $\mathcal{G}'$ could define :

$$\begin{aligned}
\mathcal{L}'(c_{dog}) &= \lambda q.q(\lambda x h l.h \geq l \wedge l : \mathbf{dog}(x))\\
\mathcal{L}'(c_{cat}) &= \lambda q.q(\lambda x h l.h \geq l \wedge l : \mathbf{cat}(x))\\
\mathcal{L}'(c_{chases}) &= \lambda b a s o.s(b(\lambda x.a(o(\lambda y h' l'.h' \geq l'\\
&\quad \wedge l' : \mathbf{chases}(x, y)))))\\
\mathcal{L}'(c_{every}) &= \lambda r p.\lambda h l.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2\\
&\quad \wedge l_2 : \mathbf{All}(v_1, l_3) \wedge l_3 : \mathbf{Imp}(l_1, h_1)\\
&\quad \wedge h_1 \geq l \wedge r\, v_1\, h\, l_1 \wedge p\, v_1\, h\, l)\\
\mathcal{L}'(c_{some}) &= \lambda r p.\lambda h' l'.\exists h_1' l_1' l_2' l_3' v_1' (h' \geq l_2'\\
&\quad \wedge l_2' : \mathbf{Ex}(v_1', l_3') \wedge l_3' : \mathbf{And}(l_1', h_1')\\
&\quad \wedge h_1' \geq l' \wedge r\, v_1'\, h'\, l_1' \wedge p\, v_1'\, h'\, l')
\end{aligned}$$

It's easy to check that the translation from the abstract term, or the derivation tree in our sense, $t = c_{chases} I_{\mathbf{S}} I_{\mathbf{VP}} (c_{dog} c_{every})(c_{cat} c_{some})$ by $\mathcal{L}'$ has the expected form:

$$\begin{aligned}
\mathcal{L}'(c_{dog} c_{every}) &= \lambda p.\lambda h l.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2\\
&\quad \wedge l_2 : \mathbf{All}(v_1, l_3) \wedge l_3 : \mathbf{Imp}(l_1, h_1)\\
&\quad \wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \mathbf{dog}(v_1)\\
&\quad \wedge p\, v_1\, h\, l)\\
\mathcal{L}'(c_{cat} c_{some}) &= \lambda p.\lambda h' l'.\exists h_1' l_1' l_2' l_3' v_1' (h' \geq l_2'\\
&\quad \wedge l_2' : \mathbf{Ex}(v_1', l_3') \wedge l_3' : \mathbf{And}(l_1', h_1')\\
&\quad \wedge h_1' \geq l' \wedge h' \geq l_1' \wedge l_1' : \mathbf{cat}(v_1')\\
&\quad \wedge p\, v_1'\, h'\, l')\\
\mathcal{L}'(c_{chases} I_{\mathbf{S}} I_{\mathbf{VP}}) = &\quad \lambda s o.s(\lambda x.o(\lambda y h' l'.h' \geq l'\\
&\quad \wedge l' : \mathbf{chases}(x, y)))
\end{aligned}$$

Hence for $\mathcal{L}'(t)$ we have:

$$\begin{aligned}
&\lambda h l.\exists h_1 l_1 l_2 l_3 v_1 (h \geq l_2 \wedge l_2 : \mathbf{All}(v_1, l_3)\\
&\wedge l_3 : \mathbf{Imp}(l_1, h_1) \wedge h_1 \geq l \wedge h \geq l_1 \wedge l_1 : \mathbf{dog}(v_1)\\
&\wedge \exists h_1' l_1' l_2' l_3' v_1' (h \geq l_2' \wedge l_2' : \mathbf{Ex}(v_1', l_3')\\
&\wedge l_3' : \mathbf{And}(l_1', h_1') \wedge h_1' \geq l \wedge h \geq l_1' \wedge l_1' : \mathbf{cat}(v_1')\\
&\quad \wedge h \geq l \wedge l : \mathbf{chases}(v_1, v_1')))
\end{aligned}$$

recovering the one from the figure 3 (modulo variable renaming). To deal with quantification in this example, we don't add any extra-link to the derivation tree (or abstract term) ones, contrary to (Kallmeyer, 2002). Both the subject (the s variable in $\mathcal{L}'(c_{chases})$) and the object parameter (the o variable) are considered as the real functors, applyed to the relation **chases** as in $s(\cdots(o(\cdots \mathbf{chases}(x, y)\cdots)))$. This implies that **N**s and **NP**s have higher-order types (see also the semantic term associated to entities in section 3.4). This is reminiscent to Montague's approach (Montague, 1974).

A term like $\mathcal{L}'(c_{chases})$ also shows the exact contribution of every node. For instance, the b variable stands for the semantic contribution of the **S** node, whereas the a variable stands for the semantic contribution of the **VP**. That is the former can act both on the predicate and its argument (see the type of $\mathcal{L}'(\mathbf{S}_A)$), whereas the latter can only modify the whole relation. The next sections illustrate this point, with adverbs and raising verbs. Then, modelling verbs with phrasal arguments, we show how the b variable can act.

In the sequel of the paper, whenever we introduce a new term which has a similar constrution to a previous one, we don't give its explicit definition (e.g. *loves*, similar to *chases*).

3.2 Adverbs

In the semantic representation we associate to c_{chases} in the previous section, we see, between the subject s and the "**VP** relation", an argument a. Its type ($\mathcal{L}'(\mathbf{VP}_A) = (h \to l \to t) \multimap (h \to l \to t)$) shows it is a verb modifier. So let us introduce a new consant $c_{usually} : \mathbf{VP}_A \multimap$

$\geq$	$: h \to l \to t$	specifies the underspecification constraints
$:$	$: l \to p \multimap t$	labels the logical predicates
$\wedge$	$: t \multimap t \multimap t$	conjunct of descriptions
$\exists_l$	$: (l \to t) \multimap t$	existential quantifier on labels
$\exists_h$	$: (l \to t) \multimap t$	existential quantifier on holes
$\exists_e$	$: (e \to t) \multimap t$	existential quantifier on entities
And, **Imp**	$: l \to h \to p$	conjonction and implication in the embedded logical language
dog, **cat**	$: e \to p$	predicates in the embedded logical language
chases	$: e \to e \to p$	predicate in the embedded logical language

Table 2: Typing of constants of Λ'_2

$\mathbf{VP}_A \in C_1$. We can associate it, with $\mathcal{L}'$, to the term:

$$\lambda a.\lambda r.\lambda hl.\exists h_1 l_1 (r\, h\, l \wedge h \geq l_1 \wedge l_1 : \mathbf{U}(h_1) \\ \wedge h_1 \geq l \wedge a(\lambda h' l'.h' \geq l') h\, l_1)$$

Its first argument, a, correspond to the verb modifier that could also be adjoined to this node (for instance an other adverb *alledgelly*). The second argument, r, corresponds to the verb predicate it modifies. Here, it is l that the adverb **U** should also dominate ($h_1 \geq l$). Then, to express that *usually* is an opaque modifier is just indicating that the label l_1 of **U** has to be the lowest point in the modification induced by a. That is l_1 is also the label argument of a.

So $c_{usually}(c_{alledgedly} I_{\mathbf{VP}})$ is mapped to

$$\lambda r.\lambda hl.\exists h_1 l_1 (r\, h\, l \wedge h \geq l_1 \wedge l_1 : \mathbf{U}(h_1) \\ \wedge h_1 \geq l \wedge \exists h'_1 l'_1 (h \geq l_1 \wedge h \geq l'_1 \wedge l'_1 : \mathbf{A}(h'_1) \\ \wedge h'_1 \geq l_1))$$

where every subformula of h'_1 is a subformula of **A**. Since h'_1 dominates l_1 which is the label of **U**, $\mathbf{U}(h_1)$ is always a subformula of **A**.

As mentionned in (Gardent and Kallmeyer, 2003), there are adverbs that would not have this opaque behaviour and rather pass the label of the verb predicate to other possibles modifiers. In this case, the argument of a is not l_1, but simply l. We illustrate it in the next example, even if not on adverbs.

3.3 Raising Verbs

Raising verbs like *seems* have been modelled in TAGs as adverbs. We can use exactly the same semantic encoding as for adverbs, except that this time it is not considered as opaque. Hence its associated term in Λ'_2 is:

$$\lambda a.\lambda r.\lambda hl.\exists h_1 l_1 (r\, h\, l \wedge h \geq l_1 \wedge l_1 : \mathbf{seems}(h_1) \\ \wedge h_1 \geq l \wedge a(\lambda h' l'.h' \geq l') h\, l)$$

3.4 Verbs with Phrasal Arguments

Going upward in the syntactic tree, we can now try to model expressions that act on **S** nodes like *claims* (see table 3). Coming back to our modelling of *chases*, we had a b argument of type $\mathcal{L}'(\mathbf{S}_A) = (e \to h \to l \to t) \multimap (e \to h \to l \to t)$. So we can associate to a term $c_{claims} : \mathbf{N}_S \multimap \mathbf{S}_A \multimap \mathbf{S}_A \in \Lambda_1$ a term in Λ'_2:

$$\lambda spr.\lambda y.p(s(\lambda x hl.\exists l_1 h_1 (h \geq l_1 \wedge \\ l_1 : \mathbf{claims}(x, h_1) \wedge r y h_1 l)))$$

which specifies that x claims something, the latter being dominated by h_1 (hence **claims**).

So for instance, an expression *Paul claims John loves Mary* would give the abstract term $c_{loves}(c_{claims} c_{Paul} I_{\mathbf{S}}) I_{\mathbf{VP}} c_{John} c_{Mary}$ and its underspecified representation ($\mathcal{L}'(c_{Paul}) = \lambda P.P\mathbf{p}$):

$$\lambda hl.\exists l_1 h_1 (h \geq l_1 \wedge l_1 : \mathbf{claims}(\mathbf{p}, h_1) \\ \wedge h_1 \geq l \wedge l : \mathbf{loves}(\mathbf{j}, \mathbf{m})))$$

because

$$\begin{aligned} \mathcal{L}'(c_{claims} c_{Paul}) &= \lambda r.\lambda y.\lambda hl.\exists l_1 h_1 (h \geq l_1 \\ &\quad \wedge l_1 : \mathbf{claims}(\mathbf{p}, h_1) \wedge r y h_1 l) \\ \mathcal{L}'(c_{loves} t I_{\mathbf{VP}} \\ c_{John} c_{Mary}) &= (\lambda P.P\mathbf{j})(t(\lambda x.(\lambda Q.Q\mathbf{m}) \\ &\quad (\lambda y h' l'.h' \geq l' \wedge \\ &\quad l' : \mathbf{loves}(x, y)))) \\ &= (\lambda P.P\mathbf{j})(t(\lambda x.\lambda h' l'.h' \geq l' \\ &\quad \wedge l' : \mathbf{loves}(x, \mathbf{m}))) \end{aligned}$$

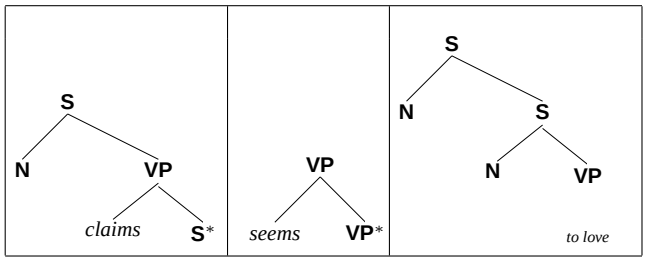


Table 3: Few more trees

Let us now illustrate the long distance dependancy behaviour, together with phrasal arguments. We can see that

if the syntactic properties of the infinitive *to love* (see table 3) really differs from the ones of *loves*, their semantic counterpart only differs in the order of argument (and an extra $\mathcal{L}'(\mathbf{S}_A)$ whose role should be precised). We can naturally associate to $\mathcal{L}'(c_{\textit{to love}})$ the term:

$$\lambda baos.s(b(\lambda x.a(o(\lambda yh'l'.h' \geq l' \wedge l' : \mathbf{loves}(x, y)))))$$

Then analyzing a long distance dependency *Mary Paul claims John seems to love* is the same as analyzing the previous example, except that the $I_{\mathbf{VP}}$ term is replaced by $c_{\textit{seems}}$ and the order of the other arguments is exchanged: $c_{\textit{loves}}(c_{\textit{claims}}c_{\textit{Paul}})(c_{\textit{seems}}I_{\mathbf{VP}})c_{\textit{Mary}}c_{\textit{John}}$. The contribution of $\mathcal{L}'(c_{\textit{seems}}I_{\mathbf{VP}})$ to $\mathcal{L}'(c_{\textit{love}})$ is just adding the conjonction of (modulo the variable renaming) $\exists h_2 l_2(h_1 \geq l \wedge l : \mathbf{loves}(\mathbf{j}, \mathbf{m}) \wedge h_1 \geq l_2 \wedge l_2 : \mathbf{seems}(h_2) \wedge h_2 \geq l)$ instead of only $h_1 \geq l \wedge l : \mathbf{loves}(\mathbf{j}, \mathbf{m}))$ so that we finally have:

$$\begin{array}{l}\lambda hl.\exists l_1 h_1(h \geq l_1 \wedge l_1 : \mathbf{claims}(\mathbf{p}, h_1)\\ \wedge \exists h_2 l_2(h_1 \geq l \wedge l : \mathbf{loves}(\mathbf{j}, \mathbf{m}) \wedge h_1 \geq l_2\\ \wedge l_2 : \mathbf{seems}(h_2) \wedge h_2 \geq l)\end{array}$$

which is the expected result.

3.5 Wh-questions

This section provides an example of an adjunction occurring on the root node of an auxiliary tree which is itself adjoined to a third tree. The expression *who does Paul think John said Bill liked,* can be analyzed with the constants $c_{\textit{who}} : \mathbf{WH}_S \in \Lambda_1$ and $c_{\textit{liked}} : \mathbf{S}_A \multimap \mathbf{VP}_A \multimap \mathbf{WH}_S \multimap \mathbf{N}_S \multimap \mathbf{S}_S \in \Lambda_1$, that correspond to the trees of figure 4. The two other constants $c_{\textit{does think}}$ and $c_{\textit{said}}$, corresponds to the auxiliary trees of the same figure and the derivation tree is $c_{\textit{liked}}(c_{\textit{said}}c_{\textit{John}}(c_{\textit{does think}}c_{\textit{Paul}}I_{\mathbf{S}}))I_{\mathbf{VP}}c_{\textit{who}}c_{\textit{Bill}}$.

Then, we can extend $\mathcal{L}'$ as follows:

$$\begin{array}{rl}\mathcal{L}'(c_{\textit{who}}) = & \lambda phl.\exists v_1 h_1'' l_1''(h \geq l_1''\\ & \wedge l_1'' : \mathbf{W}(v_1, h_1'') \wedge h_1'' \geq l \wedge pv_1 h_1'' l)\\ \mathcal{L}'(c_{\textit{liked}}) = & \lambda baos.o(b(\lambda y.a(s(\lambda xh'l'.h' \geq l'\\ & \wedge l' : \mathbf{liked}(x, y)))))\\ \mathcal{L}'(c_{\textit{said}}) = & \lambda sbr.b(\lambda y.s(\lambda xhl.\exists h_1 l_1(h \geq l_1\\ & \wedge l_1 : \mathbf{S}(x, h_1) \wedge h_1 \geq l \wedge ryh_1 l)))\\ \mathcal{L}'(c_{\textit{does think}}) = & \lambda sbr'.b(\lambda y.s(\lambda xhl.\exists h_1' l_1'(h \geq l_1'\\ & \wedge l_1' : \mathbf{T}(x, h_1') \wedge h_1' \geq l \wedge r'yh_1' l)))\end{array}$$

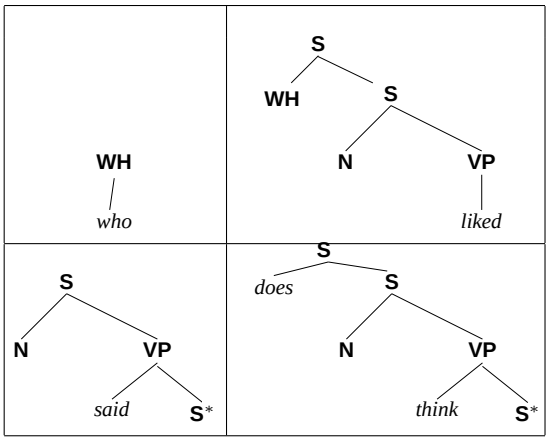


Figure 4: Wh-question example

Then, we have :

$$\begin{array}{rl}\mathcal{L}'(c_{\textit{does think}}c_{\textit{Paul}}I_{\mathbf{S}}) = & \mathcal{L}'(t_0)\\ = & \lambda r'\lambda yhl.\exists l_1' h_1'(h \geq l_1'\\ & \wedge l_1' : \mathbf{T}(\mathbf{p}, h_1') \wedge h_1' \geq l\\ & \wedge r'yh_1' l)\\ \mathcal{L}'(c_{\textit{said}}c_{\textit{John}}t_0 I_{\mathbf{S}}) = & \mathcal{L}'(t_1)\\ = & \lambda r.\lambda yhl.\exists l_1' h_1'(h \geq l_1'\\ & \wedge l_1' : \mathbf{T}(\mathbf{p}, h_1') \wedge h_1' \geq l\\ & \wedge \exists h_1 l_1(h_1' \geq l_1 \wedge l_1 : \mathbf{S}(\mathbf{j}, h_1)\\ & \wedge h_1 \geq l \wedge ryh_1 l))\end{array}$$

This yields the following result:

$$\begin{array}{rl}\mathcal{L}'(c_{\textit{liked}}t_1 I_{\mathbf{VP}}c_{\textit{who}}c_{\textit{Bill}}) = & (\lambda o.o(\lambda yhl.\exists l_1' h_1'(h \geq l_1'\\ & \wedge l_1' : \mathbf{T}(\mathbf{p}, h_1') \wedge h_1' \geq l\\ & \wedge \exists h_1 l_1(h_1' \geq l_1 \wedge l_1 : \mathbf{S}(\mathbf{j}, h_1)\\ & \wedge h_1 \geq l \wedge h_1 \geq l\\ & \wedge l : \mathbf{liked}(\mathbf{b}, y)))))\mathcal{L}'(c_{\textit{who}})\\ = & \lambda hl.\exists v_1 h_1'' l_1''(h \geq l_1''\\ & \wedge l_1'' : \mathbf{W}(v_1, h_1'') \wedge h_1'' \geq l\\ & \wedge \exists l_1' h_1'(h_1'' \geq l_1'\\ & \wedge l_1' : \mathbf{T}(\mathbf{p}, h_1') \wedge h_1' \geq l\\ & \wedge \exists h_1 l_1(h_1' \geq l_1 \wedge l_1 : \mathbf{S}(\mathbf{j}, h_1)\\ & \wedge h_1 \geq l \wedge h_1 \geq l\\ & \wedge l : \mathbf{liked}(\mathbf{b}, v_1))))\end{array}$$

which is the expected one, with $\mathbf{W}$ binding the variable v_1 and dominating $\mathbf{T}$, itself dominating $\mathbf{S}$, itself dominating $\mathbf{liked}(\mathbf{b}, v_1)$.

3.6 Control Verbs

Control verbs, as presented in (Gardent and Kallmeyer, 2003) or (Frank and van Genabith, 2001), with adjunc-

tion on a **S** node (see table 4) to produce an expression like *John tries to sleep*, with the adjunction of *tries to* on *sleep*, is a problem for our approach.

Indeed, it is build from the term $c_{sleep}(c_{tries\ to} I_{\mathbf{VP}} c_{John} I_{\mathbf{S}}) I_{\mathbf{VP}}$ and the typing discipline makes $t = c_{tries\ to} I_{\mathbf{VP}} c_{John} I_{\mathbf{S}}$ of type $\mathbf{S}_A$, hence $\mathcal{L}'(t)$ of type $(e \to h \to l \to t) \multimap e \to h \to l \to t$. If it is clear that the first argument of type $(e \to h \to l \to t)$ concerns the **sleep** predicate (with something like $\lambda xhl.h \geq l \wedge l : \mathbf{sleep}(x)$), the result should not have any e possible argument (it has been filled with **j**).

In other worlds, if we look at adjunctions on **S** nodes in previous sections, the subtrees always lack an **N** (*John seems to love* x, or *John said Bill liked* x) and are always transformed into a subtree lacking an **N** too (*Paul claims John seems to love* x, or *does Paul think John said Bill liked* x). This is not the case anymore with control verbs where the subtree for x *sleep* turns into *John tries to sleep*.

So control verbs cannot be dealt with directly that way with our techniques. We need for instance to differentiate the $\mathbf{S}_A$ type into the usual one $(e \to h \to l \to t) \multimap e \to h \to l \to t$ and another one $(e \to h \to l \to t) \multimap h \to l \to t$. This could be done with a special $\mathbf{S}_{\text{Pro}}$ node, or with an extended type system (for instance additives of linear logic to manage disjunctive types). But this requires further investigation and goes beyond this article.

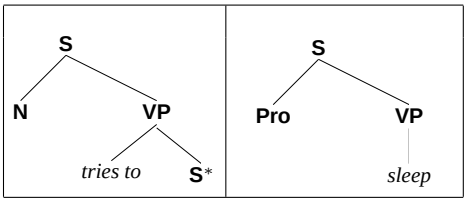


Table 4: Derived trees for control verbs

Conclusion

We propose to reconsider semantic representation computation for TAG from the derivation trees. But derivation trees here are understood as abstract terms of ACGs, even if the informations born by each of the formalism are essentially the same. Whereas they hold the specification of how trees should combine, the locality of computing the meanings held by the different nodes is described in the object vocabulary. It obviates the addition of extra links to manage scoping and shows that derivation trees, by themselves, are enough, even if further investigation are required to handle control verbs.

It also clearly defines the compositional aspects of building semantic representations with a clear and modular distinction between syntax and semantics. The latter point lacks in the derived tree approaches. Moreover, the mathemetical primitives we use are very simple (if expression not always are) and are the same both on the syntactic and the semantic side, and no external principles need to be added.

So, from the ACG point of view, both syntax and semantics are dealt with in an equivalent way: as object languages of the same abstract language. This is interesting because the computation engine to go from the object language to the abstract language in an ACG does not depend on the object language. So the underlying process remains the same for all that cases:

- to compute a derived tree, then a derivation tree, from a string;
- to compute a derivation tree from a URL formula;
- to compute a derived tree, then a string, from a derivation tree;
- to compute an URL formula from a derivation tree.

So that going from one to the other (parsing or generation, in the usual sense) is as difficult (or as easy) as going the other way. Of course, on the semantic side, it means the initial point is an URL formula, and it gives no hint on how to build it from an SRL formula, nor on how to deal with the logical equivalence (be it on the SRL or on the URL level).

Finally, it underlines the interesting feature of ACG to transport or transmit structures from one language to anoher, illustrated between a syntactic formalism and a semantic formalism for TAGs. As suggested by an anonymous referee, the same approach could be used to provide semantic representations to expressions belonging to m-linear context-free languages, since abstract terms have already been proposed for them (de Groote and Pogodalla, 2003).

References

Anne Abeillé. 1993. *Les nouvelles syntaxes*. Armand Colin Éditeur, Paris.

Patrick Blackburn and Johan Bos. 2003. Computational semantics for natural language. `http://www.iccs.informatics.ed.ac.uk/~jbos/comsem/book1.html`. Course Notes for NASSLLI 2003.

Johan Bos. 1995. Predicate logic unplugged. In *Proceedings of the Tenth Amsterdam Colloquium*.

Marie-Hélène Candito and Sylvain Kahane. 1998. Can the tag derivation tree represent a semantic graph? an answer in the light of meaning-text theory. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Framework (TAG+4)*, volume 98-12 of *IRCS Technical Report Series*.

Vincent Danos and Roberto Di Cosmo. 1992. The linear logic primer. http://www.pps.jussieu.fr/~dicosmo/CourseNotes/LinLog/. An introductory course on Linear Logic.

Philippe de Groote and Sylvain Pogodalla. 2003. m-linear context-free rewriting systems as abstract categorial grammars. In Richard Oehrle and James Rogers, editors, *MOL 8, proceedings of the eighth Mathematics of Language Conference*.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155.

Philippe de Groote. 2002. Tree-adjoining grammars as abstract categorial grammars. In *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150. Università di Venezia.

Anette Frank and Josef van Genabith. 2001. Glue tag: Linear logic based semantics construction for ltag - and what it teaches us about the relation between lfg and ltag. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG '01 Conference*, Online Proceedings. CSLI Publications. http://cslipublications.stanford.edu/LFG/6/lfg01.html.

Claire Gardent and Laura Kallmeyer. 2003. Semantic construction in feature-based tag. In *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*.

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science*, 50:1–102.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible composition in ltag: Quantifier scope and inverse linking. In Harry Bunt, Ielka van der Sluis, and Roser Morante, editors, *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*.

Laura Kallmeyer. 2002. Using an enriched tag derivation structure as basis for semantics. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*.

Richard Montague, 1974. *The Proper Treatment of Quantification in Ordinary English*, chapter 1. In Portner and Partee (Portner and Partee, 2002).

Sylvain Pogodalla. 2004. Using and extending ACG technology: Endowing categorial grammars with an underspecified semantic representation. In *Proceedings of Categorial Grammars 2004, Montpellier*, June.

Paul Portner and Barbara H. Partee, editors. 2002. *Formal Semantics: The Essential Readings*. Blackwell Publishers.

Yves Schabes and Stuart M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Tree-Adjoining Grammars for Optimality Theory Syntax

Virginia Savova
Department of Cognitive Science
Johns Hopkins University
savova@jhu.edu

Robert Frank
Department of Cognitive Science
Johns Hopkins University
rfrank@jhu.edu

Abstract

This paper explores an optimality-theoretic approach to syntax based on Tree-Adjoining Grammars (TAG), where two separate optimizations are responsible for the construction of local pieces of tree structure (elementary trees) and the combination of these pieces of structure. The *local* optimization takes a non-recursive predicate-argument structure (PA-chunk) as an underlying representation and chooses the best tree structure realizing it. The *linking* optimization takes as an underlying representation a tree whose nodes are labeled by PA-chunks and chooses among a set of structurally isomorphic TAG derivation trees. We provide formal definitions of the OTAG system and prove equivalence in strong generative capacity between OTAG and TAG. Finally, we apply the mechanics of the formal system to the analysis of cross-serial dependencies in Swiss-German.

1 Introduction

Optimality Theory (OT) claims that linguistic expressions are restricted by a set of universal, mutually inconsistent and violable constraints (Prince and Smolensky, 1993). Conflicts result in the satisfaction of higher ranked constraints at the expense of their lower ranked adversaries. The variations among languages are attributed to differences in the constraint rankings. In OT, a grammatical linguistic expression is a winner of an optimization. Given an underlying representation (UR), a generator function (Gen) produces a (potentially infinite) set of surface realizations (SRs), and a process of optimization picks the SRs that minimally violate the constraints according to a language-particular ranking.

OT is a general framework that can give rise to a variety of specific formal instantiations depending on the types of representations and constraints invoked, but it is a largely unresolved question just what sort of formalism is appropriate for OT syntax. Since natural language syntax permits recursively embedded structures, this suggests that the OT optimizations ought to apply to unbounded domains. However, optimization over such structures can give rise to a system with excessive generative capacity, if the number of violations of a constraint can grow without bound as well (Frank and Satta, 1998; Wartena, 2000). Moreover, if we look at the properties of natural language syntax, it appears that the structural tradeoffs that arise from the resolution of constraint conflict take place over local domains.

We therefore propose an OT formalism based on Tree Adjoining Grammar, which we call Optimality Tree Adjoining Grammar (OTAG), where separate optimizations are responsible for the construction of local pieces of tree structure (elementary trees) and the combination of these pieces of structure.The first optimization (which we call *local optimization*) takes as UR a non-recursive predicate argument structure (PA-chunk) and chooses among a set of local trees generated by Gen as candidate SRs of this PA-chunk. The local optimization yields a finite tree language which serves as a set of elementary trees. The second type of optimization (which we refer to as *linking optimization*) takes as UR a tree whose nodes are labeled by PA-chunks (a derivation tree of sorts) and chooses among a set of structurally isomorphic TAG derivation trees, where each node in these trees is labeled by an elementary tree that is among the locally optimal outputs for the corresponding PA-chunk.

2 Definitions

Let us begin with a formal definition of an OT system, adapted from (Frank and Satta, 1998).

Def. 1 *An* optimality system *is a 4-tuple* $OS =$

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 72-79.

$\{\Sigma, \Gamma, Gen, C\}$ *where* Σ *and* Γ *are the finite input and output alphabets, Gen is a relation over* $\Sigma^* \times \Gamma^*$*, and* C *is a finite set of total functions from* $\Sigma^* \times \Gamma^*$ *to* N.

As seen in this definition, Gen maps a UR to a set of SRs, while a constraint is a function from a candidate UR-SR pair to a natural number, which we take to represent the degree of violation incurred by that candidate on that constraint. An OS gives rise to a set of optimality grammars (OG), defined in (2):

Def. 2 *An* optimality grammar *OG is an OS together with a total ordering* R *on* C*, called a* ranking.

Frank and Satta's definition is not directly applicable to OT syntax because it defines the URs and the SRs as strings. We assume that in syntax, the SRs are trees, while the URs are predicate-argument (PA) structures in tree form. A PA structure may contain simple and nested predicates. A simple predicate is a predicate applied over atomic arguments, i.e., arguments that do not contain predicates, as in example (1).

(1) loves(John, Mary)

A nested predicate is a predicate applied to other predicates, like *says* in example (2).

(2) says(Bill, (loves(John, Mary)))

We postulate a grammatical component, the $\mathbf{PA-chunker}$, which breaks down a complex PA structure into simple PA structures by substituting non-atomic arguments with predicate labels, which are treated as atomic arguments in the local optimization.

Def. 3 *A* PA-chunker *is a function from a nested PA structure* P *to a set of pairs containing a simple PA-structure (PA-chunks)* S *and a label* l *for that structure, such that*

i. each predicate in P *is a predicate in exactly one of the PA-chunks in* S*;*

ii. the atomic arguments of each predicate in P *are the same as the arguments of that predicate in corresponding PA-chunk in* S*; and*

iii. each complex argument A *of a predicate* π *in* P *is replaced in the PA-chunk containing* π *in* S *by the label uniquely associated with the simple PA-chunk in* S *corresponding to* A.

For example, the nested PA structure in (2) will give rise to the set of simple PA structures in (3) (where X and Y are predicate labels).

(3) {([says (Bill, X)], Y), ([loves (John, Mary)], X)}

In our setting, PA-chunks are the URs for optimizations over bounded domains whose outputs are local trees. The URs for optimizations over unbounded domains are tree structures over nodes labeled by a PA-chunk and all winning surface realizations of that PA-chunk (in the form of syntactic trees).

With this in mind, we define Optimality Tree Adjoining Systems (OTAS) as follows:

Def. 4 *An* Optimality Tree Adjoining System *is a 9-tuple* $OTAS = \{\Sigma, \Gamma, \Pi, Chunk, Loc, Gen_C, Gen_K, C, K\}$ *where*

- Σ *and* Γ *are finite input and output alphabets;*
- Π *is a set of predicate labels;*
- $Chunk$ *and* Loc *are finite sets of finite trees labeled by* $\Sigma \cup \Pi$ *and* Γ *respectively;*
- Gen_C *is a relation over* $Chunk \times Loc$*;*
- Gen_K *is a relation over* $\Psi \times \Xi$*, where*
  - *i.* Ψ *is the set of finite trees each of whose nodes are labeled by members of* $Chunk \times \Pi \times 2^{Loc}$ *where for each* $\tau \in \Psi$*, a node labeled* (σ, π, γ) *is a daughter of node* (σ', π', γ') *iff* σ' *contains label* π*;*
  - *ii.* Ξ *is the set of finite trees labeled by* Γ*;*
- C *is a finite set of total functions from* $Chunk \times Loc$ *to* N*;*
- K *is a finite set of total functions from* $\Psi \times \Xi$ *to* N *(with* Ψ *and* Ξ *defined as above).*

The alphabets Σ and Γ are the sets of symbols in the representations making up the UR and SR, respectively. In our current conception, Σ consists of the set of predicate and argument symbols, while Γ contains the set of terminal and non-terminal symbols.[1] $Chunk$ will contain the set of URs that feed the local optimization, the set of PA-chunks, while Loc contains the SRs that can be the output of this process, the possible syntactic realizations of the PA-chunks. Gen_C maps a PA-chunk $\sigma \in Chunk$ to corresponding SR $\gamma \in Loc$. Gen_K maps any tree structure whose nodes are labeled by (local-UR, pred-label, locally-optimal-SRs) triples to a recursive surface tree realization. C is the set of constraints on local trees, while K is the constraints over recursive trees.[2] According to definition (2), an OT grammar is obtained by imposing a unique ranking on the set of constraints. In OTAG, a ranking must be specified for each type of optimization.

Def. 5 *An* OTAG Grammar *(OTG) is an OTAS with a pair of rankings* R_C, R_K *on* C *and* K.

[1] To keep things relatively simple, our definition neither enforces the arity requirements of predicate symbols nor the proper placement of predicate labels, terminal and non-terminal symbols in building members of $Chunk$ or Loc.

[2] Note that we are assuming that set of possible realizations of a member of $Chunk$ is finite. This is reasonable under the assumption that there is a finite set of winners for each optimization.

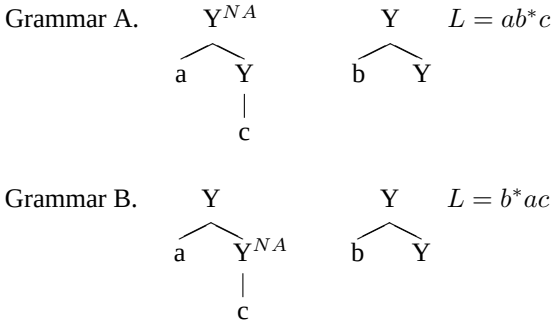


Figure 1: Related TAG grammars.

With these definitions in place, we can now define the notion of optimization in an OTG. Let us begin with local optimization:

Def. 6 *The* local optimum, $LOpt(p)$, *associated with a simple predicate argument structure* p *is defined recursively, as in (Frank and Satta, 1998):*

$$LOpt^i(p) = \begin{cases} Gen_C(p) & \text{if } i = 0; \\ argmin_{c_i}(LOpt^{i-1}(p)) & \text{if } i \geq 1 \end{cases}$$

$$LOpt(p) = LOpt^m(p) \text{ where } m = |C|$$

Given such a set of local optima, we can now define the linking optimization process. Assume that we have a recursive predicate argument structure Π. The input to the linking optimization is a tree whose labels are taken from the following set of locally optimal pairings:

$$\Lambda = \{(p, \pi, LOpt(p))|(p, \pi) \in \text{ PA-Chunk}(p)\}$$

Given such a Λ, there will be a unique tree τ such that (p, π, γ) is a daughter of node (p', π', γ') iff p' contains predicate label π. Linking optimization is now defined over this τ as in definition 6, using Gen_K and constraint set K.

3 Substitution, adjoining and the Linking Optimization

In traditional TAG, grammars sharing the same set of local trees can generate different languages. An example of this situation is depicted in Figure 1, where we see two grammars that differ only in the locus of adjoining constraints and generate distinct languages. Since the linking optimization in OTAG constrains how the elementary trees that result from the local optimization are put together, the languages of these grammars could also generated by two OTAGs derived from the same OTAG system with different constraint rankings (Figure 2).

The constraints on adjoining are implemented in the set of violable constraints K, which prohibit or require adjoining at a set of nodes. In the grammar illustrated here, C_1 requires some adjoining to take place, C_2 forbids adjoining at the root Y node of the *ac* elementary tree, and C_3 forbids adjoining at the lower Y node of the same tree. When C_1 is ranked above either or both of C_2 or C_3, the higher ranked of this latter pair of constraints determines where adjoining applies, whereas when C_1 is lowest ranked, no adjoining takes place at all. Constraint reranking, then, achieves the effect of altering the loci of adjoining constraints. In principle, the linking optimization may apply globally, evaluating the whole UR against a derivation, but that would lead to the possibility of conditioning an adjunction at high levels on lower level adjunctions. In order to limit the generative power of OTAG, we require that the linking optimization apply cyclically. Each cycle adjoins a set of auxiliary trees into a single local tree, and these cycles proceed in a bottom-up fashion through the PA-chunk structure that is the input to the linking optimization. The result of a linking optimization may be used for a subsequent cycle, when a derived auxiliary is adjoined. This constraint enforces a strong parallelism between the OTAG derivation and the TAG derivation. They differ only by the presence of an optimization step in OTAG, which determines where the auxiliary tree is adjoined into another elementary tree. In other words, an OTAG derivation tree represents a series of optimal adjoining operations.

With this restriction in place, it turns out that the resulting formalism is exactly as powerful as the TAG formalism. Specifically, we can prove the following theorems (see appendix for proofs):

Theorem 1 *For any TAG G, there is a OTAG G' such that T(G) = T(G').*

Theorem 2 *For any OTAG G', there is a TAG G such that T(G') = T(G).*

4 OTAG in action: An illustrative example

To illustrate the practical application of the formalism, we will go through the steps of a derivation of the Swiss-German cross-serial construction, and the corresponding

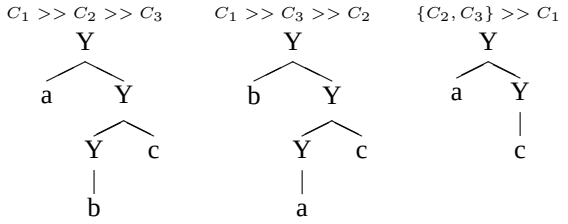


Figure 2: Output of OTAG grammars that differ only in constraint ranking.

German and English constructions. Swiss-German exhibits cross-serial dependencies that can be modeled by the language $LCross = a^n b^m c^n d^m | m, n \in N$ (Shieber, 1985).

(4) De Jan säit, dass mer em Hans es huus
John-NOM says that we Hans-DAT the house
hälfed aastriiche. (Swiss German)
helped paint

'John says that we helped Hans paint the house.'

Compare this to the English and German equivalents.

(5) John says that we helped Hans paint the house.

(6) Jan sagt, daß wir Hans das Haus
John says that we Hans the house-Acc
anstreichen hilften. (German)
paint helped

'John says that we help Hans paint the house.'

The German sentence exhibits center embedding - the innermost verb case-marking the innermost noun, the outermost verb case-marking the outermost noun. In the English case, there is no embedding at all: verbs always immediately precede their associated arguments.

Let us consider the necessary steps in an OTAG analysis of these data. First, we must isolate the local winners. As we know, they are SRs corresponding to PA-chunks. Table 4 shows the simple predicates and the corresponding yield of the local winners in English, German, and Swiss-German. The symbol _ marks the insertion site for the other SR. The question we need to tackle is what kind of trees yield these strings. We notice that the German and Swiss-German cases differ from the English case by the position of the verb with respect to its arguments. One way to account for this difference would be to invoke a Headedness constraint on the local trees, Head-Left, and a counter-constraint, e.g., Head-Right. We also invoke a local Markedness constraint such as "Move V" which conflicts with a Faithfulness constraint "*trace" (a.k.a. "Stay!", cf. Grimshaw1977). These constraints are defined as follows:

- Move V: Raise V to T.
- *trace: No traces.

In German, unlike English, "*trace" is ranked lower than "Move V". Note that the overt difference between English and German can be explained by assuming the verb *help* raises to node Y, without assuming anything about the verb *paint*. However, our OTAG analysis forces us to make a theoretical commitment that *paint* also raises, since the tree it is part of is a winner of a local optimization under the same constraint hierarchy.

We can now characterize the Swiss-German case in a way consistent with our theory of the English and German cases. At this point, we are going to make use of the linking optimization to distinguish German from Swiss-German in particular. Descriptively, Swiss-German differs from German by the fact that *help* intervenes between *paint* and its argument. This is exactly what we expect if we assume that adjoining in Swiss-German takes place at a lower node than adjoining in German. In the analysis of English and German, the node X_3 was the adjoining site. By supposing that instead, the adjoining site for Swiss-German is X_4, we obtain the desired cross-serial dependency. To enforce this difference in adjoining sites, we need to postulate two constraints that play a role in the linking optimization by favoring nodes X_3 and X_4, respectively. A linguistically motivated constraint favoring X_3 may be related to the relationship between Hans and PRO resulting from the adjoining. In English and German, but not in Swiss-German, Hans c-commands PRO in the output of the linking optimization. Another plausible constraint is a subcategorization constraint on the adjoining tree. Suppose the adjoining tree is of type A and node X_3 is of a particular type N. Thus, the linking optimization may involve a constraint "C-PRO: PRO must be c-commanded" and a constraint "A-to-N: Adjoin trees of type A to nodes of type N" ranked differently with respect to each other. In our case, let us suppose "trees of type A" means "Auxiliary trees of type VP" and "Nodes of type N" means "Highest VP node of initial tree." To recount, here is how our model analysis would play out. Table 4 presents the local optimizations with candidate structures, including the winners for English (E), German (G) and Swiss-German (SG).

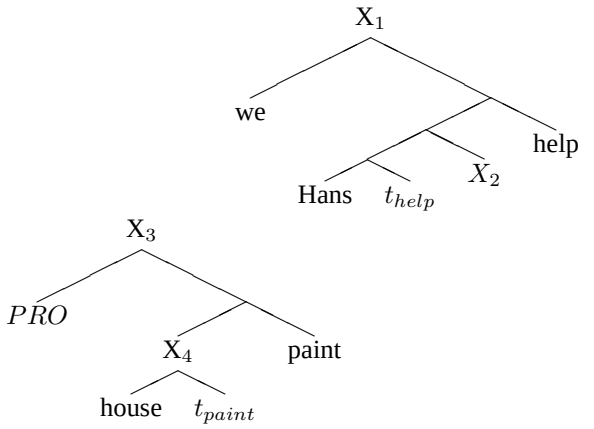


Figure 3: Adjoining occurs at X_4 in Swiss-German, X_3 in German

Note that at this point the local optimization contains two constraints more than necessary to account for the data. We can prune the analysis by removing any pair of constraints that favor opposite candidates. For example,

PA-chunks	English	German	Swiss-German
$([paint(Hans, house)], X)$	paint the house	das Haus anstreichen	es huus aastriiche
helped(we, Hans, X)	We helped Hans _	wir Hans _ hilften	mer em Hans_ hälfed _

Table 1: PA-chunks

(paint(Hans, house)],X)	Head-Left	Head-Right	*trace	Move V
E: [PRO [paint house]]		*		*
G, SG:[PRO [[t_{paint} house] paint]]	*		*	
help(we, Hans, X)				
E: [we [[help Hans]]]		*		*
G, SG:[we [[[t_{help} Hans]] help]]	*		*	

Table 2: Local optimizations

paint(Hans, house)	Head-Left	Head-Right
E: [PRO [paint [t_{paint} house]]]	*	
G, SG: [PRO [[t_{paint} house] paint]]		*
help(we, Hans, X)		
E: [we [help [t_{help} Hans]]]	*	
G, SG: [we [[[t_{help} Hans]] help]]		*

Table 3: Local optimization simplified

we have the option of scrapping either the pair Head-Left, Move V or the pair Head-Right, *trace from the constraint set. If we get rid of the former pair, we will essentially be claiming that movement of the verb happens in order to position the head to the right of the verb phrase. Alternatively, if we remove the latter constraint pair, we will be suggesting that movement of the verb can only happen to the right and hence necessarily violates Head-Left. There is no reason to dismiss either scenario right away. On the other hand, some new data might discredit either alternative and persuade us to keep all constraints in the set. Finally, a third scenario may involve obligatory verb movement in both English and German/Swiss-German. In this case, the only relevant players in the constraint set are Head-Left and Head-Right, which force the movement to take the preferred direction. The optimization would include only candidate representations in which movement has occurred (i.e. Loc would be restricted to such structures, Table 3).

Another issue in the local optimization is the realization of the argument "Hans" as PRO in one sentence, but as *Hans* in the other. This issue can only be solved by exploiting the possibility of multiple winners in the local optimizations. In other words PRO and the full argument must be indistinguishable from the point of view of the local optimization, but one or the other must be preferred in the linking optimization. The argument is simple. By virtue of our definition of the PA chunker, the predicate argument structure *paint(Hans, house)* is independent of the larger complex predicate it was embedded in. Consequently, the same predicate argument structure would qualify as an UR of *Hans paints the house* since the latter is a grammatical structure, Hans may equally surface as PRO or simply *Hans*. We need to update our Table once again by adding two more competitors, as shown in Table 4. This competition is resolved in the subsequent linking optimizations as seen in Table 5. The constraint "*Repeat" penalizes the repetition of a nominal element. Admittedly, this is a very crude way of enforcing the presence of PRO in the final structure. A more sophisticated way of defining *Repeat could refer to the relationship between trees with argument Arg in SpecVP on one hand, and trees with the same argument Arg in a complement position on the other. For example: *Repeat: Do not adjoin trees with complement Arg to trees with Arg in SpecVP This formulation is a better match for the type of constraints we have used in our formal treatment of OTAG so far.

The role of *Repeat here is to show how multiple winners in the local optimization allow us to sneak in solutions to differences in the form of main versus embedded clauses. Recall that, if the PA-chunker is only given the simple predicate argument structure to start with, the linking optimization will involve adjoining of the null tree. Consequently, "*Repeat" will not play a role, as shown in Table 6. At the same time, any constraint related to PRO would disadvantage PRO in this setting and the full argument would surface. This completes our illustrative

paint(Hans, house)	Head-Left	Head-Right
E: [PRO [paint [[t_{paint} house]]]	*	
G, SG: [PRO [[t_{paint} house] paint]]		*
E: [Hans [paint [t_{paint} house]]]	*	
G, SG: [Hans [[t_{paint} house] paint]]		*

Table 4: Full NP and PRO are tied in the local optimization

help(we, Hans, X). paint(Hans, house)	$C-PRO$	A-to-N	*Repeat
E: [we [[help Hans] [PRO [paint house]]]]		*	
G: [we [[[Hans t help] [PRO [[house t_{paint}] paint]]help]]		*	
SG: [PRO [[we [[[Hans t_{help}] [house t_{paint}]] help]]paint]]	*		
* [we [[help Hans] [Hans [paint house]]]]		*	*
* [we [[[Hans t_{help}] [Hans [[house t_{paint}] paint]]help]]		*	*
* [Hans [[we [[[Hans thelp] [house t_{paint}]]help]]paint]]		*	*

Table 5: Linking optimization licenses PRO in subordinate clause

$\emptyset$. paint(Hans, house)	$C-PRO$	*Repeat
* [PRO [paint house]]	*	
*[PRO [[house t_{paint}] paint]]	*	
E: [Hans [paint house]]		
G, SG: [Hans [[house t_{paint}] paint]]		

Table 6: Linking Optimization eliminates PRO in main clause

analysis of the Swiss-German construction and its cross-linguistic counterparts. The important points to remember are:

- When analyzing a complex structure, complex PA structures are broken into chunks.
- Predicate labels in the PA chunks constrain what adjoins into what in the linking optimization.
- Adjustments in the ranking among constraints in the local optimization permit different structural variants to win.
- Both main clause and the embedded clause variants of a PA chunk must be possible winners in the local optimization.
- If the embedded clause is not grammatical as a main clause, the linking optimization must include a constraint that favors the embedded clause over the main clause.

5 Conclusion

Our proposal is a step towards a restrictive and adequate framework for handling syntactic phenomena in the spirit of OT. We have demonstrated that the generative power of any grammar specified within the framework is limited to the class of MCSLs, which many believe is the complexity class of natural languages. The main theoretical advantage of the OTAG formalism is the locality imposed by the optimization over simple predicates in the first stage of the derivation of an arbitrarily complex structure. Another, more practical advantage stems from the relative transparency of the components of the framework. Our formalism relies on a specific kind of underlying representation, a specific way to handle recursion, and a general template for constraints. Clearly, further work is needed to test the viability of this framework for a broader range of empirical phenomena.

Acknowledgments

Special thanks to Paul Smolensky for helpful comments on an earlier draft. The research in this paper was supported in part by NSF grant SBR-9972807.

References

Robert Frank and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. *Comput. Linguist.*, 24(2):307–315.

Alan Prince and Paul Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar. Technical report, Rutgers University Center for Cognitive Science.

Stuart Shieber. 1985. Evidence against the context-

freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Christian Wartena. 2000. A note on the complexity of optimality systems. Ms. Universität Potsdam, Rutgers Optimality Archive (ROA-385-03100).

Appendix: Proofs of theorems

We define a TAG G as a tuple (A, I, R), where A is the set of auxiliary trees, I is the set of initial trees, and R is the set of adjoining constraints associated with nodes of $A \cup I$. We require that A contain a distinguished null auxiliary tree ϵ, capable of adjoining at any node. With such an ϵ tree, we can assume without loss of generality that every legal TAG/OTAG derivation involves adjoining to every node of every tree involved in the derivation. An adjoining constraint $r \in R$ specifies a set of trees S and a node d such that S cannot adjoin at d ($r = *S@d$). Such a constraint corresponds to the usual notion of selective adjoining constraint. Obligatory adjoining constraints can be modeled as a constraint which forbids adjoining of ϵ. Null adjoining constraints permit adjoining of only the tree ϵ. On the OTAG side, we will assume a constraint *NIL that penalizes SRs in the linking optimization in which trees present in the UR do not participate in the TAG derivation yielding the surface tree. Finally, we use the notation $T(G)$ to refer to the set of well-formed derivation trees in a TAG or OTAG.

Theorem 1. For any TAG G, there is a OTAG G' such that T(G) = T(G').

Given a TAG $G = (I, A, R)$, we define OTAG $G' = (\Sigma, \Gamma, \Pi, Chunk, Loc, C, K)$, such that $Loc = A \cup I$ and $K_{G'} = \{*NIL\} \cup \{k_r | r \in R\}$ where k_r penalizes a candidate if it involves an adjoining that would violate TAG adjoining constraint r.[3]

Claim 1 $D \in T(G) \rightarrow D \in T(G')$

Proof by induction on the depth of D (represented (D)):

Base case Let $(D) = 0$.D consists of a node t whose only children are instances of the empty tree ϵ. Let t be a tree with nodes {1...n}. $D \notin T(G')$ iff one of the following is true:

1. $\{t, \epsilon\} \notin Loc$. But ϵ is always in A. Moreover, $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $Loc = A \cup I, t \in Loc$.
2. $\exists k_1 ... k_n \in K | k_i = *\{\epsilon\}@i$, for i a node $\in t$. This is true only if $\exists \{r_1 ... r_n\} \in R | r_i = *\{\epsilon\}@i$, i a node $\in t$. But if $\{r_1 ... r_n\} \in R$ was true $D \in T(G)$ would be false.

Hence $\{k_1 ... k_n\}$ do not exist and $D \in T(G')$

Induction hypothesis Suppose Claim 1 is true for all $(D) \leq k$. Let t be the root of D and $\{1...n\}$ the set of nodes in t. Let $\{D_1 ... D_n\}$ be a set of derivations with roots $\{a_1 ... a_n\} \in A$ such that a_i is adjoined to node i in t. Observe that $(D_i) \leq k$ for $1 \leq i \leq n$. $D \notin T(G')$ iff one of the following is true:

1. $t \notin Loc$. But $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $Loc = A \cup I, t \in Loc$;
2. $\{D_1 ... D_n\} \notin T(G')$.But $\{D_1 ... D_n\} \in T(G')$ by the induction hypothesis;
3. $\exists k_i \in K | k_i = *a_i@i$. This is true only if $\exists r_i \in R | r_i = *a_i@i$. But if this were true, $D \in T(G)$ would be false.

Hence k_i do not exist and $D \in T(G')$

Claim 2 $W \in T(G') \rightarrow W \in T(G)$

Proof by induction on the depth of W:

Base case (W) = 0. W consists of one optimization adjoining the empty tree ϵ into some $w \in Loc$. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $Loc = A \cup I$ and $w \in Loc$. Hence $w \in A \cup I$.
2. $\exists r_1 ... r_n \in R | r_i = *\{\epsilon\}@i$, for i a node $\in t$. This is true only if $\exists \{k_1 ... k_n\} \in K | k_i = *\{\epsilon\}@i$, for i a node $\in t$. But if $\{k_1 ... k_n\} \in K$ was true, $W \in T(G')$ would be false.

Hence $\{r_1 ... r_n\}$ do not exist and $W \in T(G)$

Induction hypothesis Suppose Claim 2 is true for any derivation W, $(W) \leq k$. Let w be the root of W and $\{1...n\}$ the set of nodes in w. Let $\{W_1 ... W_n\}$ be a set of derivations with roots $\{z_1 ... z_n\} \in Loc$ such that z_i is adjoined at node i. Observe that $(W_i) \leq k$ for all $1 \leq i \leq n$. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $w \in T(G')$ by hypothesis, which is true only if $w \in Loc$. Since $Loc = A \cup I, w \in A \cup I$;
2. $\{W_1 ... W_n\} \notin T(G)$.But $\{W_1 ... W_n\} \in T(G)$ by hypothesis;
3. $\exists r_i \in R | r_i = *z_i@i$. This is true only if $\exists k_i \in K | k_i = *z_i@i$. But if $k_i \in K$ was true $W \in T(G')$ would be false.

Hence r_i do not exist and $W \in T(G)$. ■

Theorem 2. For any OTAG G', there is a TAG G such that T(G') = T(G).

Here, we will also give a general procedure for converting a OTAG into an equivalent TAG. Before we proceed, it would be useful to informally consider the two cases that cause complications in this conversion. Both cases are easily illustrated with a minimal OTAG. Suppose Loc contains only two trees: the initial tree t and the auxiliary tree a. In addition, let t contain only two nonterminal nodes (n1, n2). Case 1: Now suppose that the constraint set K of our OTAG G contains two OA constraints, k_1 and k_2, such that k_1 and k_2 require the adjoining of the same tree a at different nodes (n_1, n_2) of the tree t $(k_1 = *(A - a)@n_1; k_2 = *(A - a)@n_2;)$.

[3] We do not define $\Sigma, \Pi, Chunk$, or C since the there is no counterpart to the local optimization in TAG. Since the set of elementary trees is finite, we can assume the existence of some set of constraint C that will produce this set of trees from appropriate URs.

Furthermore, suppose $*Null >> k_1 >> k_2$. This constraint ranking would enforce the adjoining of a into $n_2(t)$ only if another instance of a is adjoined at $n_1(t)$. Case 2 is similar: Suppose that the constraint set K of our OTAG G contains two NA constraints, k_1 and k_2 against adjoining any auxiliary tree a at either one of two different nodes (n_1, n_2) of the same tree t. Furthermore, suppose $*Null >> k_1 >> k_2$. This constraint ranking would allow adjoining into $n_1(t)$ only if adjoining has taken place already at $n_2(t)$. It is clear from these cases that a simple translation of constraints into adjoining constraints is not sufficient. The violated OA constraint k_2 cannot be emulated by an OA constraint forcing a to adjoin at n_2 (because the adjoining fails when a is not adjoined at n_1); nor does it correspond to a SA constraint that merely allows adjoining of a at n_2 (because it the adjoining is obligatory whenever an instance of a is already adjoined at n_1). Thus, instead of picking a single type of constraint to place on each elementary tree, we need to multiply out the trees in Loc affected by problematic constraint sets of this type. The tree t corresponds to a subset of two trees in the elementary tree set of the corresponding TAG: One tree has an OA constraint on node n_1. The other has an NA constraint on node n_2. Similarly, the violated NA constraint k_2 cannot be emulated by a NA constraint against a on n_1 (because the adjoining could occur if an instance of a is already adjoined at n_2). Neither can it be completely disregarded, because it prevents a from adjoining into n_1 if a has not adjoined to n_2 beforehand. The tree t maps to a subset of two trees in the elementary tree set of the corresponding TAG: One tree has an NA constraint on n_1, the other has an OA constraint on n_2. Let G' be a $OTAG = \{\Sigma, \Gamma, \Pi, Chunk, Loc, Gen_C, Gen_K, C, K\}$ with rankings R_C and R_K. Then TAG $G = \{A, I, R\}$, obtained based on the outcome of all linking optimizations involving the adjoining of a set S of trees from Loc into some tree t in Loc (note that $|S| \leq$ the number of non-terminals in t).

Conversion algorithm:

Step 1: Create a table T_t of size $n \times p$ associated with each tree t in Loc, where n is the number of nodes in t, and p is the number of possible multisets of trees Z drawn from Loc of cardinality n. In each cell (j, k), enter all trees $z \in Z$ adjoined to node j in some linking optimization over Υ, where Υ is a UR tree whose nodes are labeled with triples $(\sigma_i, \pi, \gamma_i)$ and $\cup(\gamma_i) = k$.

Step 2: For every tree $t \in Loc$, create a set of elementary trees E_t containing distinct copies of t for each cell of T_t. For each such $t_{(i,j)} \in E_t$, create adjoining constraints $r = *A - T_t(i, j)@h$, where h is the name of the copy of node i in $t_{(i,j)}$.

Claim 1: $W \in T(G') \rightarrow W \in T(G)$

Proof by induction on depth of W.

Base case Let $(W) = 0$. W involves one optimization adjoining of only instances of the empty tree ϵ into some $w \in Loc$. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $A \cup I \supseteq Loc$ and $w \in Loc$. Hence $w \in A \cup I$.
2. $\exists r_1 ... r_n \in R | r_i = *\{\epsilon\}@i$, i a node $\in t$. This is true only if ϵ never adjoins into w in the linking optimization of G'. But if this were the case, $W \in T(G')$ would be false.

Hence $\{r_1 ... r_n\}$ do not exist and $W \in T(G)$

Induction hypothesis Suppose Claim 1 is true for any derivation $(W) \leq k$. Let w be the root of W and $\{1...n\}$ the set of nodes in w. Let $\{W_1 ... W_n\}$ be a set of derivations,$(W_i) \leq k$ with roots $\{z_1 ... z_n\} \in Loc$ such that z_i is adjoined at node i. $W \notin T(G)$ iff one of the following is true:

1. $w \notin A \cup I$. But $w \in T(G')$ by hypothesis, which is true only if $w \in Loc$. Since $A \cup I$ contains copies of all the trees in $Loc, w \in A \cup I$;
2. $\{W_1 ... W_n\} \notin T(G)$. But $\{W_1 ... W_n\} \in T(G)$ by hypothesis;
3. $\exists r_i \in R | r_i = *z_i@i$. This is true only if G' disallows adjoining of z_i to i, in which case $W \in T(G')$ would be false.

Hence r_i do not exist and $W \in T(G)$.

Claim 2: $D \in T(G) \rightarrow D \in T(G')$

Proof by induction on depth of D:

Base case Let $(D) = 0$. D consists of a node t whose only children are the empty tree ϵ. Let t be a tree with nodes {1...n}. $D \notin T(G')$ iff one of the following is true:

1. $\{t, \epsilon\} \notin Loc$. But ϵ is always in A. Moreover, $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $A \cup I$ contains only copies of trees in $Loc, t \in Loc$.
2. $\exists k_1 ... k_n \in K | k_i = *\{\epsilon\}@i$, i a node $\in t$. This is true only if $\exists \{r_1 ... r_n\} \in R | r_i = *\{\epsilon\}@i$, i a node $\in t$. But if $\{r_1 ... r_n\} \in R$ was true $D \in T(G)$ would be false.

Hence $\{k_1 ... k_n\}$ do not exist and $D \in T(G')$

Induction hypothesis Suppose Claim 1 is true for any $(D) \leq k$. Let t be the root of D and $\{1...n\}$ the set of nodes in t. Let $\{D_1 ... D_n\}$ be a set of derivations,$(D_i) \leq k$ with roots $\{a_1 ... a_n\} \in A$ such that a_i is adjoined to node i. $D \notin T(G')$ iff one of the following is true:

1. $t \notin Loc$. But $D \in T(G)$ by hypothesis, which is true only if $t \in A \cup I$. Since $A \cup I$ contains only copies of trees in Loc, $t \in Loc$;
2. $\{D_1 ... D_n\} \notin T(G')$.But $\{D_1 ... D_n\} \in T(G')$ by hypothesis;
3. $\exists k_i \in K | k_i = *a_i@i$. This is true only if $\exists r_i \in R | r_i = *a_i@i$. But if $r_i \in R$ was true $D \in T(G)$ would be false.

Hence k_i do not exist and $D \in T(G')$. ■

Semantic Reconstruction for *how many*-Questions in LTAG

Tatjana Scheffler
Department of Linguistics
619 Williams Hall
University of Pennsylvania
Philadelphia, PA 19104–6305
tatjana@ling.upenn.edu

Abstract

In this paper, we show how to formalize reconstruction effects in an LTAG semantics. We derive a lexical entry and semantic specification for *how many*, which introduces two quantificational elements. We also show how they interact compositionally with other scopal items, e.g. modal and attitude verbs in a question. The use of an underspecified semantics allows the compact representation of scope ambiguities. We demonstrate how this also enables us to obtain the correct readings in embedded questions.

1 Introduction

Semantic *reconstruction* is an effect that is appealed to if a scopal element seems to be interpreted "further down" in the syntactic tree than it actually occurs. One example are complex wh-questions, in which a part of the wh-phrase sometimes must be interpreted as if it occurred in the approximate position of its trace (in a transformation-based analysis).

How many-questions are such complex wh-questions, because *how many* introduces two quantifiers (basically, *what n* and *n-many*). Thus, sentence (1) is ambiguous with respect to whether reconstruction of the second quantifier (*n-many*) into the object position occurs or not.[1]

(1) How many students did Mary interview?
For what n: there are n-many people y_i, such that Mary interviewed y_i.
$\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{interview}(k, y, w)))]$[2,3]

This ambiguity is made apparent if other scopal elements, like modal verbs, adjoin to the sentence. Example (2) has two separate meanings, with different relative scope of *n-many* and *should*.

(2) How many students should Mary interview?

(a) For what n: it should be the case that there are n-many students y_i such that Mary interviewed y_i.
$\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{should}(\mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x))))]$

(b) For what n: there are n-many students y_i such that it should be the case that Mary interviewed y_i.
$\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{should}(\mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x))))]$

The first meaning might be intended when Mary is known to make a representative survey among students, and the speaker wants to know how many students (no matter who they are) have to be interviewed in order for Mary to be able to make valid judgments. Meaning (b) is more salient if Mary has been assigned to ask certain students (e.g., Bill, Bob, and Susan), and the speaker wants to know how big the group of people whom Mary has to interview is exactly.

In earlier approaches to such semantics, the effect is accounted for by postulating a trace in the canonical position of the wh-element (Cresti, 1995). A part of the

[1] Note that reconstruction of a quantifier into a lower position in the tree does not deny that quantifier the possibility to raise by normal quantifier raising. In fact, in the case of *how many*, the *what n* is a wh quantifier which has to take the widest possible scope. The *n-many* quantifier is a normal non-wh quantifier which can be interpreted in the usual "scope window" for NP quantifiers such as "some" and "every". Alternatively, by way of appearing together in one word with the wh quantifier, *n-many* can take the higher wh-scope here.

[2] We loosely follow the view of (Karttunen, 2003) on the meaning of questions, which analyses a question denotation as a set of propositions, namely all those propositions that answer the question.

[3] stud^* means "a plurality of students".

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 80-87.

wh-phrase is then said to be reconstructed in that position, from which it can optionally raise across other, higher scopal elements. Thus, an ambiguity arises with respect to the relative scopings of scopal elements in the sentence.

These phenomena seem to pose problems for a semantics interface on top of a syntactic theory which, like TAG, does not make use of traces or movement. However, we demonstrate here that the use of feature structures not only makes an account possible, but also provides us with a compact underspecified representation of scope ambiguities that arise due to the optionality of reconstruction.

2 LTAG Semantics

It is commonly argued that semantic composition in TAG should be done with respect to the derivation tree, not the derived tree. This is possible because each elementary tree is associated with its appropriate semantic representation, and the semantics of the sentence is composed incrementally in parallel with the syntactic composition (see e.g. Kallmeyer and Joshi, 2003; Joshi et al., 2003; Gardent and Kallmeyer, 2003).

In this paper we use the framework presented in Kallmeyer and Romero (2004): We use a flat semantic representation with unification variables (similar to MRS, Copestake et al., 1999). In addition to predications, the semantics contain propositional metavariables. Constraints on the relative scope of the metavariables and propositional labels are used to provide underspecified representations of scope ambiguities. The semantic representation is stored in semantic feature structures that are part of the lexical entry, together with the elementary tree. To keep track of the necessary variable unifications, semantic features are associated with each node position in the elementary tree.[4] The values of these features are feature structures that consist of a T and a B feature (top and bottom) whose values are feature structures with features I for individual variables, P for propositional labels etc.

The semantic composition follows the usual definitions for unification in Feature-Based TAG syntax: For each edge in the derivation tree from elementary tree γ_1 to γ_2 with position p: (1) the T feature of position p in γ_1 and the T feature of the root of γ_2 are identified, and (2) if γ_2 is an auxiliary tree, then the B feature of the foot node of γ_2 and the B feature of position p in γ_1 are identified. Furthermore, at the end of a syntactic derivation, the top and bottom feature structures at each node are unified. By these unifications, some of the variables in the semantic representations get values. Then, the union of all semantic representations is built which yields an underspecified representation with scope constraints.

To obtain the different possible scopings of the sentence, all possible *disambiguations*, i.e. injective functions from the remaining propositional variables to labels, must be found. The disambiguated representations are interpreted conjunctively.

Quantifiers Following Joshi and Vijay-Shanker (1999); Kallmeyer and Joshi (2003) and in particular Romero et al. (2004), we assume that quantificational NPs as *every* in (3) and also *who* in (4) are syntactically split into two parts of one multicomponent set. One tree is substituted into the appropriate NP node and provides the predicate-argument information; the other tree is a degenerate auxiliary tree that consists only of a single S node, and which contributes the scope part. Figure 1 shows the syntax for sentence (3).

(3) Every dog barks.

(4) Who laughs?

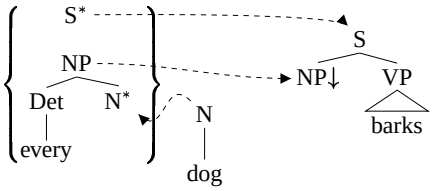


Figure 1: Syntax of (3) *Every dog barks*.

The semantic derivation for the simple quantified sentence (3) is shown in figure 2. The unifications lead to the following feature identities: $\boxed{1} = \boxed{6}$ (adjunction of the scope part), $\boxed{3} = x$ and $\boxed{7} = l_3$ (substitution of *dog* into determiner), $\boxed{2} = x$ and $\boxed{8} = l_1$ (substitution of the NP into *barks*). Replacing the variables by their values and building then the union of all semantic representations leads to (5):

(5) $$\boxed{\begin{array}{l} l_1 : \mathsf{bark}(x), l_2 : \mathsf{every}(x, \boxed{4}, \boxed{5}), l_3 : \mathsf{dog}(x) \\ \boxed{1} \geq l_1, \boxed{4} \geq l_3, \boxed{5} \geq l_1, \boxed{1} \geq l_2 \end{array}}$$

There is only one disambiguation, $\boxed{1} \rightarrow l_2, \boxed{4} \rightarrow l_3, \boxed{5} \rightarrow l_1$, which leads to the final semantic representation: $\mathsf{every}(x, \mathsf{dog}(x), \mathsf{bark}(x))$.

Questions The feature *maximal scope* (MAXS) is needed to provide the correct maximal scope of quantifiers. This is important in questions, as we will see later. Furthermore, MAXS is also used to make sure that quantifiers embedded under attitude verbs such as *think* cannot scope over the embedding verb (see Kallmeyer and Romero, 2004, for further discussion).

[4] For the sake of readability, we use names np, vp, ... for the node positions instead of the usual Gorn adresses.

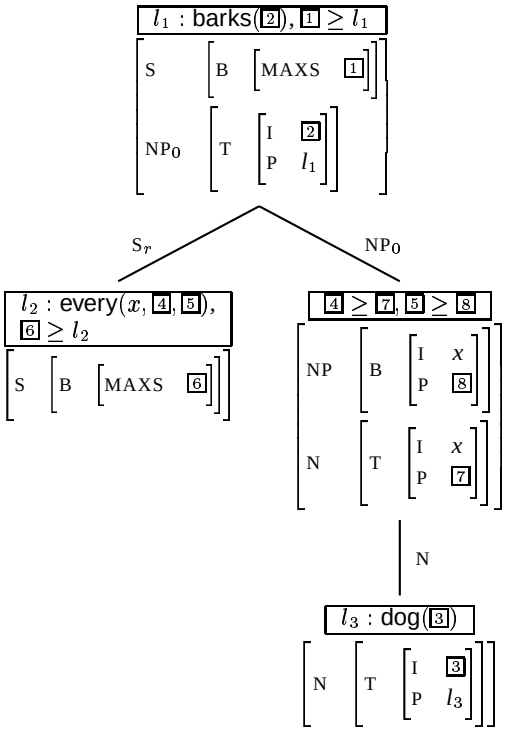


Figure 2: Semantic derivation of *Every dog barks.*

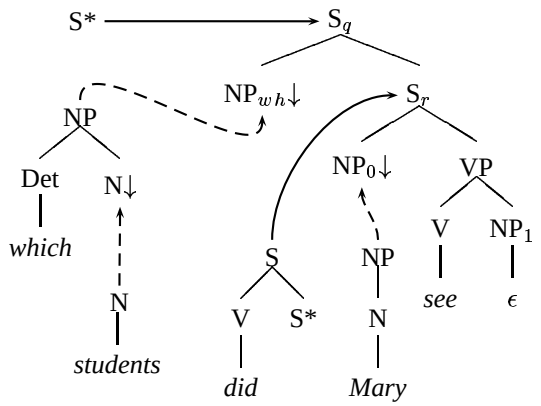


Figure 3: Syntactic derivation of *Which students did Mary see?*

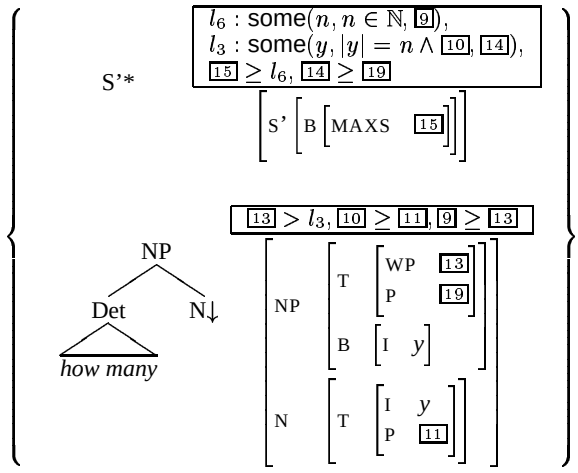


Figure 5: Lexical entry for *how many*.

Following Romero et al. (2004), we assume that wh-operators, like quantifiers, also have a separate scope part and they also have a MAXS scope limit. But their scope limit is provided by the S' node, not the S node. For an analysis of the question *Which students did Mary see?*, see figures 3 and 4.

The MAXS features together with the semantics of the question verb make sure that all wh-operators have scope over the question proposition (here l_2) and all quantifiers scope below this proposition. The minimal nuclear scope of the wh-operator (variable 2) is provided by the question proposition l_2.

3 A Lexical Entry for *how many*

In this section, we give Multicomponent-TAG elementary trees and appropriate semantic representations that show how to derive the meaning of *how many* sentences in TAG.

As noted above, the phrase *how many* introduces two existential quantifiers. Both appear together in the semantic representation. As for all (wh-)quantifiers, the contribution is split up into a predicate-argument and a scope part. Here, the predicate-argument part is empty and contains only some constraints. This makes *how many* analogous to *which* (see the derivation in figure 4 above), in that the restriction is provided by the noun that substitutes into the quantifier. The lexical entry we propose for *how many* is shown in figure 5.

The additional complication of this lexical item is that the two quantifiers it contributes do not have exactly the same scope. One (l_6) is a wh-quantifier that needs to take scope over the question proposition in the verbal tree. The constraint 9 ≥ 13 guarantees that the wh-quantifier itself must stay on top of the tree and not be reconstructed.

The other quantifier is a "normal" one whose minimal scope is the elementary predication of the verbal tree. Thus, it is not enough to have one single feature P in the root node of the predicate-argument part to provide the minimal scope for both quantifiers (as was still sufficient in the case of *which* above). We introduce a feature WP

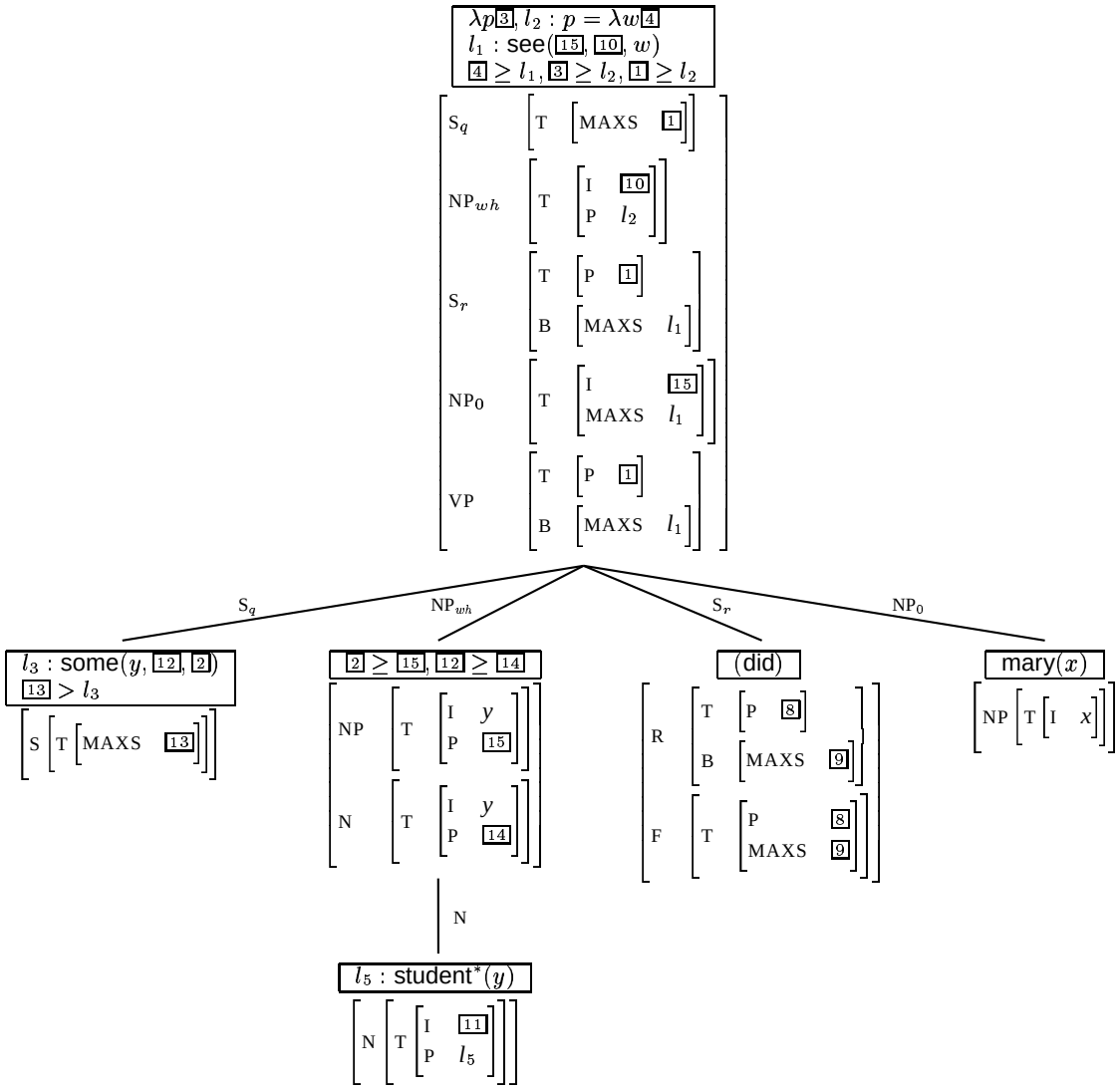


Figure 4: Semantic derivation of *Which students did Mary see?*

for this purpose, which provides the minimal scope for the wh-quantifier. Feature P is kept for the non-wh minimal scope. 19 will unify with the verb's basic predicate.

On the other hand, non-wh quantifiers are usually restricted by the MAXS feature of the S node their scope part adjoins into, which in turn is used during embedding under attitude verbs: In *Mary thinks John likes everybody*, the universal quantifier cannot scope over *thinks*. For the non-wh part of *how many*, however, this restriction does not seem to hold: *How many students does Mary think John likes?* is ambiguous between *many* scoping over *think*, or *think* over *many*.[5] This fact is captured in the proposed lexical entry by not giving a maximal scope restriction for the non-wh quantifier l_3. Of course, the constraints $\boxed{9} \geq \boxed{13}$ and $\boxed{13} > l_3$ ensure that l_3 is in the nuclear scope of the wh-quantifier l_6.

4 Interaction with other Scopal Elements

The interesting problem of scopal reconstruction is to obtain the two possible readings of a sentence like (2). The meaning in (b) is easily derivable, because no reconstruction occurs. Reading (a), however, must be obtained by reconstructing $\mathsf{some}(y, stud^*(y) \wedge |y| = n, ...)$ under $\mathsf{should}(...)$.[6] Figure 6 shows the semantic derivation for sentence (2).

[5]This was also pointed out by one reviewer.

[6]For simplicity, an abbreviated notation for the semantics of should is used in this paper. More accurately, the modal verb should introduce a universal quantifier over situations. We will not deal with the computations related to situations here.

Scope underspecification is obtained in the following way: both the *many*-quantifier and *should*'s minimal scopes are restricted by constraints ($\boxed{14} \geq \boxed{19}$ and $\boxed{16} \geq \boxed{17}$, respectively), which makes them both scope over l_1 eventually. Furthermore, the two scopal elements are maximally restricted to be in the scope of the question proposition. Their relative scope is left undetermined.

The feature identities that are derived during the semantic computation of (2) are $\boxed{15} = \boxed{1}, \boxed{13} = l_2, \boxed{19} = l_1, \boxed{4} = y, \boxed{12} = y, \boxed{11} = l_5, \boxed{2} = \boxed{18} = \boxed{6}, \boxed{17} = \boxed{7}, \boxed{3} = x, \boxed{7} = l_1$. Building the union of all semantic representations and substituting values for metavariables as possible leads to the underspecified semantic representation (6):

(6)
$\lambda p\boxed{1}, l_2 : p = \lambda w\boxed{2}, l_1 : \mathsf{intv}(x, y, w),$
$l_6 : \mathsf{some}(n, n \in \mathbb{N}, \boxed{9}),$
$l_3 : \mathsf{some}(y, |y| = n \wedge \boxed{10}, \boxed{14}), l_5 : \mathsf{student}^*(y),$
$l_7 : \mathsf{should}(\boxed{16}), \mathsf{mary}(x)$
$\boxed{1} \geq l_2, \boxed{2} \geq l_1, \boxed{1} \geq l_6, \boxed{14} \geq l_1, l_2 > l_3,$
$\boxed{10} \geq l_5, \boxed{9} \geq l_2, \boxed{16} \geq l_1, \boxed{2} \geq l_7$

There is are two possible disambiguations, namely:

(a)				(b)			
	$\boxed{1}$	$\rightarrow$	l_6		$\boxed{1}$	$\rightarrow$	l_6
	$\boxed{9}$	$\rightarrow$	l_2		$\boxed{9}$	$\rightarrow$	l_2
	$\boxed{2}$	$\rightarrow$	l_7		$\boxed{2}$	$\rightarrow$	l_3
	$\boxed{16}$	$\rightarrow$	l_3		$\boxed{10}$	$\rightarrow$	l_5
	$\boxed{10}$	$\rightarrow$	l_5		$\boxed{14}$	$\rightarrow$	l_7
	$\boxed{14}$	$\rightarrow$	l_1		$\boxed{16}$	$\rightarrow$	l_1

which result in the two appropriate readings for the sentence:

(a) $\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{should}(\mathsf{some}(y, |y| = n \wedge \mathsf{stud}^*(y), \mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x))))]$

(b) $\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{some}(y, |y| = n \wedge \mathsf{stud}^*(y), \mathsf{should}(\mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x))))]$

Attitude Verbs In TAG, predicates that take clausal complements anchor auxiliary trees that adjoin into their embedded sentences. Figure 7 shows the lexical entry for the verb *think*[7].

A verb like *think* functions as a boundary for MAXS by projecting a different variable upwards. However, as we have seen above, the maximal scope of the non-wh quantifier of *how many* is not restricted by the MAXS feature of the S node. This ensures that even if a how-many question is embedded under an attitude verb, there is some freedom for the quantifier's scope with respect to other scopal elements, e.g., *should* and *think*. Therefore, sentence (7) still has at least the two meanings given along with it in (a) and (b). In addition, one meaning should be

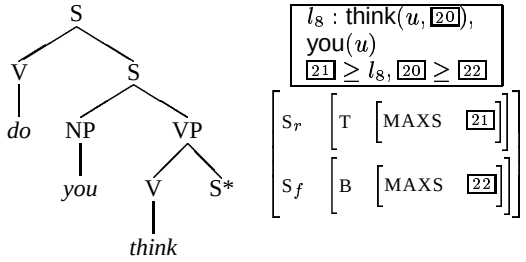


Figure 7: Lexical entry for *think*.

obtainable where *many* scopes over both *think* and *should* (c). This reading shall not concern us here.

(7) How many students do you think Mary should interview?

(a) $\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{think}(u, \mathsf{should}(\mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x)))))]$

(b) $\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{think}(u, \mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{should}(\mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x)))))]$

(c) $\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{think}(u, \mathsf{should}(\mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x)))))]$

The syntactic analysis of example (7) is depicted in figure 8. The semantic derivation for the sentence is very similar to the non-embedded sentence (2), shown in figure 6. The only difference is the additional adjuntion of the semantic representation as shown in figure 6 with the semantic formulae and feature structure shown in figure 7, at the S_r node of the *interview* tree.

The feature unifications triggered by the semantic derivation are: $\boxed{15} = \boxed{1}, \boxed{13} = l_2, \boxed{19} = l_1, \boxed{4} = y, \boxed{12} = y, \boxed{11} = l_5, \boxed{18} = \boxed{6}, \boxed{17} = \boxed{7}, \boxed{3} = x, \boxed{7} = l_1, \boxed{21} = \boxed{2}, \boxed{22} = \boxed{6}$. (Note that because of the adjunction, some previous unifications are not carried out any more: $\boxed{2} \neq \boxed{6}$.) This yields the following semantic representation for the complete sentence *How many students do you think Mary should interview?*:

(8)
$\lambda p\boxed{1}, l_2 : p = \lambda w\boxed{2}, l_1 : \mathsf{intv}(x, y, w),$
$l_6 : \mathsf{some}(n, n \in \mathbb{N}, \boxed{9}),$
$l_3 : \mathsf{some}(y, |y| = n \wedge \boxed{10}, \boxed{14}), l_5 : \mathsf{student}^*(y),$
$l_7 : \mathsf{should}(\boxed{16}), \mathsf{mary}(x),$
$l_8 : \mathsf{think}(u, \boxed{20}), \mathsf{you}(u)$
$\boxed{1} \geq l_2, \boxed{2} \geq l_1, \boxed{1} \geq l_6, \boxed{14} \geq l_1, l_2 > l_3,$
$\boxed{10} \geq l_5, \boxed{9} \geq l_2, \boxed{16} \geq l_1, \boxed{6} \geq l_7, \boxed{2} \geq l_8, \boxed{20} \geq \boxed{6}$

The representation accounts for the fact that *think* necessarily scopes over *should*, but the *many*-quantifier can scope out of it.

[7] For simplicity, we have already combined *think* with *do* and *you* in this figure. So for all practical purposes, this would not be a lexical entry for any broad TAG-grammar, although nothing in the theory prohibits such lexical items.

Two of the possible disambiguations (where *think* has widest scope) are shown below, and they represent the two readings (a) and (b):

(a)
$$\begin{array}{lcl} \boxed{1} & \rightarrow & l_6 \\ \boxed{9} & \rightarrow & l_2 \\ \boxed{2} & \rightarrow & l_8 \\ \boxed{20} & \rightarrow & l_7 \\ \boxed{16} & \rightarrow & l_3 \\ \boxed{10} & \rightarrow & l_5 \\ \boxed{14} & \rightarrow & l_1 \end{array}$$

(b)
$$\begin{array}{lcl} \boxed{1} & \rightarrow & l_6 \\ \boxed{9} & \rightarrow & l_2 \\ \boxed{2} & \rightarrow & l_8 \\ \boxed{20} & \rightarrow & l_3 \\ \boxed{10} & \rightarrow & l_5 \\ \boxed{14} & \rightarrow & l_7 \\ \boxed{16} & \rightarrow & l_1 \end{array}$$

Islands Reconstruction is not always possible. In examples such as (9) with extraction out of weak islands (Ross, 1967), only the non-reconstructed reading (where Mary should interview specific students) is possible for *how many*.

(9) How many students do you wonder whether Mary should interview?

(b) $\lambda p.[\mathsf{some}(n, n \in \mathbb{N}, p = \lambda w.\mathsf{wonder}(u, \mathsf{some}(y, \mathsf{stud}^*(y) \wedge |y| = n, \mathsf{should}(\mathsf{intv}(x, y, w) \wedge \mathsf{mary}(x)))))]$

The status of weak islands is not completely clear. Many studies suggest that the factor that prohibits one of the possible interpretations in sentences such as (9), and which is traditionally attributed to the failure of *students* to reconstruct across a weak island barrier (see Cresti, 1995), is really a pragmatic rather than syntactic or semantic phenomenon.

The issue whether this effect can be accounted for compositionally with LTAG or whether it has to be resolved by a pragmatic process is left for further work.

5 Conclusion

In this paper we showed that using recently developed frameworks for representing semantics in LTAG, we can account for ambiguities that arise in *how many* questions in an elegant way. The use of underspecified semantics and the feature unification process as employed also in the syntactic composition in TAG together allow the reconstruction of non-wh quantifier lower in the tree.

We proposed a lexical entry and semantic specification for *how many* which introduces two quantifiers, one of the wh type, and one non-wh quantifier. We presented how these quantifiers obtain exactly the right scopal possibilities in simple and embedded questions. Furthermore, we showed how the proposed lexical entry interacts compositionally with other scopal elements in questions, such as modal verbs, and how two readings are obtained from a single semantic representation.

An account for weak island constraints is left for future work. We propose that weak island barriers in these contexts may actually be a pragmatic effect that should not affect our semantic analysis.

Acknowledgments

I would like to thank Maribel Romero, Aravind K. Joshi and the members of the XTAG Group at the University of Pennsylvania for many fruitful discussions of the analyses presented in this paper. Furthermore, I am indebted to two anonymous reviewers for their detailed and helpful comments.

References

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 1999. Minimal Recursion Semantics. An Introduction. Draft, Stanford University.

Diana Cresti. 1995. Extraction and reconstruction. *Natural Language Semantics*, 3:79–122.

Claire Gardent and Laura Kallmeyer. 2003. Semantic construction in Feature-Based TAG. In *Proceedings of the 10th EACL*, Budapest, Hungary.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible composition in LTAG, quantifier scope and inverse linking. In *Proceedings of the 5th IWCS*, pages 179–194, Tilburg, NL.

Aravind K. Joshi and K. Vijay-Shanker. 1999. Compositional Semantics with Lexicalized Tree-Adjoining Grammar (LTAG): How Much Underspecification is Necessary? In H. C. Blunt and E. G. C. Thijsse, editors, *Proceedings of the Third International Workshop on Computational Semantics (IWCS-3)*, pages 131–145, Tilburg.

Laura Kallmeyer. 1999. *Tree Description Grammars and Underspecified Representations*. Ph.D. thesis, Universität Tübingen. Technical Report IRCS-99-08 at the Institute for Research in Cognitive Science, Philadelphia.

Laura Kallmeyer and Aravind K. Joshi. 2003. Factoring predicate argument and scope semantics: Underspecified semantics with LTAG. *Research on Language and Computation*, 1:3–58.

Laura Kallmeyer and Maribel Romero. 2004. LTAG Semantics with Semantic Unification. In *Proceedings of TAG+7*, Vancouver, Canada.

Lauri Karttunen. 2003. Syntax and semantics of questions. In Paul Portner and Barbara H. Partee, editors, *Formal Semantics. The Essential Readings*, pages 382–420. Blackwell.

Maribel Romero, Laura Kallmeyer, and Olga Babko-Malaya. 2004. LTAG Semantics for Questions. In *Proceedings of TAG+7*, Vancouver, Canada.

John Robert Ross. 1967. *Constraints on variables in Syntax*. Ph.D. thesis, MIT.

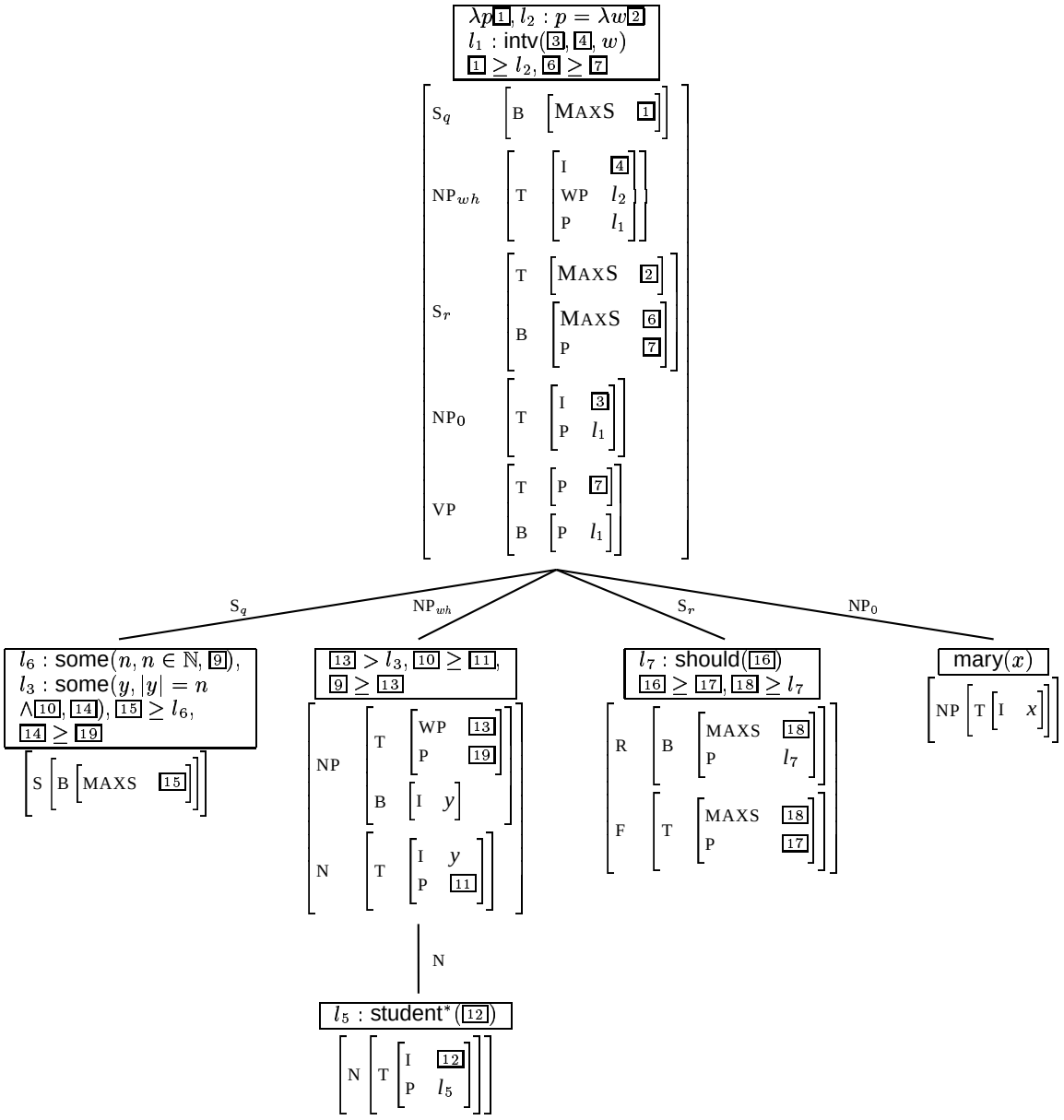


Figure 6: Semantic derivation tree for (2) *How many students should Mary interview?*

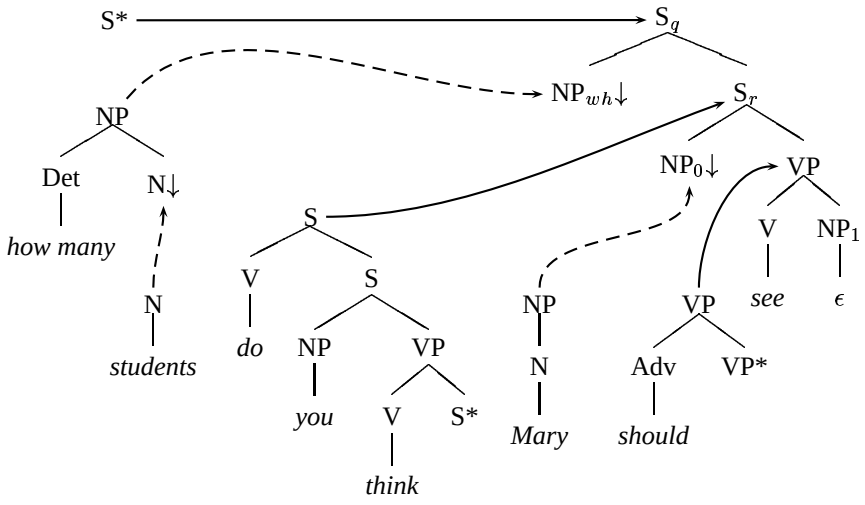


Figure 8: Syntactic derivation of *How many students do you think Mary should see?*

Synchronous Grammars as Tree Transducers

Stuart M. Shieber
Division of Engineering and Applied Sciences
Harvard University
shieber@deas.harvard.edu

Abstract

Tree transducer formalisms were developed in the formal language theory community as generalizations of finite-state transducers from strings to trees. Independently, synchronous tree-substitution and -adjoining grammars arose in the computational linguistics community as a means to augment strictly syntactic formalisms to provide for parallel semantics. We present the first synthesis of these two independently developed approaches to specifying tree relations, unifying their respective literatures for the first time, by using the framework of bimorphisms as the generalizing formalism in which all can be embedded. The central result is that synchronous tree-substitution grammars are equivalent to bimorphisms where the component homomorphisms are linear and complete.

1 Motivation

The typical natural-language pipeline can be thought of as proceeding by successive transformation of various data structures, especially strings and trees. For instance, low-level speech processing can be viewed as transduction of strings of speech samples into phoneme strings, then into triphone strings, finally into words strings. (Because of nondeterminism in the process, the nondeterministic string possibilities may be represented as a single lattice. Nonetheless, the underlying abstract operation is one of string transduction.) Morphological processes can similarly be modeled as character string transductions. For this reason, weighted finite-state transducers (WFST), a general formalism for string-to-string transduction, can serve as a kind of universal formalism for representing low-level natural-language processes (Mohri, 1997).

Higher-level natural-language processes can also be thought of as transductions, but on more highly structured representations, for instance trees. Semantic interpretation can be viewed as a transduction from a syntactic parse tree to a tree of semantic operations whose simplification to logical form can be viewed as a further transduction. This raises the question as to whether there is a universal formalism for NL tree transductions that can play the same role there that WFST plays for string transduction.

In this paper, we investigate the formal properties of synchronous tree-substitution and -adjoining grammars (STSG and STAG) from this perspective. In particular, we look at where the formalisms sit in the pantheon of tree transduction formalisms. As a particular result, we show that, contra previous conjecture, STSG is not equivalent to simple nondeterministic tree transducers, and place for the first time STSG and STAG into the tree transducer family. Essential to this unification of the two types of formalisms is the bimorphism characterization of tree transducers, little known outside the formal language theory community.

We begin by recalling the definitions of nondeterministic top-down tree transducers ($\downarrow TT$), and their description in terms of bimorphisms, and also provide a definition of STSG and STAG. We show that $\downarrow TT$ and STSG differ in their expressive properties; these differences argue in favor of the synchronous formalisms for NL use. Finally, we prove the equivalence between STSG and a new kind of bimorphism, which characterization makes some of the properties of STSG trivial. This view of STSG generalizes to provide a bimorphism characterization of STAG as well.

This work makes several contributions to our understanding of tree transducers and the synchronous formalisms. First, it provides the first unification of the two, placing both in a consistent framework, that of bimorphisms. Second, it provides intuition about appropriate properties of such formalisms for the purpose of natural-language processing applications, which may help inform the search for a universal NL tree transduction formalism.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 88-95.

2 Preliminaries

We start by defining the terminology and notations that we will use for strings, trees, and the like.

We will notate sequences with angle brackets, e.g., $\langle a, b, c\rangle$, with the empty string written ϵ. The number of elements in a set or sequence x will be notated $|x|$.

Trees will have nodes labeled with elements of a RANKED ALPHABET, a set of symbols $\mathcal{F}$, each with a non-negative integer RANK or ARITY assigned to it, say by a function $arity$, determining the number of children for nodes so labeled. Symbols with arity zero are called NULLARY symbols; with arity one, UNARY; with arity two, BINARY. We write $\mathcal{F}_n$ for the set of symbols in $\mathcal{F}$ with arity n. To express incomplete trees, trees with "holes" waiting to be filled, we will allow leaves to be labeled with variables, in addition to nullary symbols.

The set of TREES OVER A RANKED ALPHABET $\mathcal{F}$ AND VARIABLES $\mathcal{X}$, notated $\mathcal{T}(\mathcal{F}, \mathcal{X})$, is the smallest set such that

Nullary symbols at leaves $f \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}_0$;

Variables at leaves $x \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $x \in \mathcal{X}$;

Internal nodes $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for all $f \in \mathcal{F}_n$, $n \geq 1$, and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

We abbreviate $\mathcal{T}(\mathcal{F}, \emptyset)$, where the set of variables is empty, as $\mathcal{T}(\mathcal{F})$, the set of GROUND TREES over $\mathcal{F}$. We will also make use of the set of n numerically ordered variables $\mathcal{X}_n = \{x_1, \ldots, x_n\}$, and write x, y, z as synonyms for x_1, x_2, x_3, respectively.

Trees can also be viewed as mappings from TREE ADDRESSES, sequences of integers, to the labels of nodes at those addresses. The address ϵ is the address of the root, $\langle 1\rangle$ the address of the first child, $\langle 1, 2\rangle$ the address of the second child of the first child, and so forth. We will use the notation $t@p$ to pick out the label of the node at address p in the tree t, that is, (using $\cdot$ for the insertion of an element on a list)

$$\begin{aligned} f(t_1, \ldots, t_n)@\epsilon &= f \\ f(t_1, \ldots, t_n)@(i \cdot p) &= t_i@p \\ &\text{for } 1 \leq i \leq n \quad . \end{aligned}$$

We can use trees with variables as CONTEXTS in which to place other trees. A tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ will be called a context, typically denoted with the symbol C. The notation $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ denotes the tree in $\mathcal{T}(\mathcal{F})$ obtained by substituting for each x_i the corresponding t_i.

For a context $C \in \mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ and a sequence of n trees $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, the SUBSTITUTION OF $t_1, \ldots, t_n$ INTO C, notated $C[t_1, \ldots, t_n]$, is defined inductively as follows:

$$\begin{aligned} &(f(u_1, \ldots, u_m))[t_1, \ldots, t_n] \\ &\quad = f(u_1[t_1, \ldots, t_n], \ldots, u_m[t_1, \ldots, t_n]) \\ &x_i[t_1, \ldots, t_n] = t_i \quad . \end{aligned}$$

A tree $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is LINEAR if and only if no variable in $\mathcal{X}$ occurs more than once in t.

3 Tree Transducers and Bimorphisms

The variation in tree transducer formalisms is extraordinarily wide and the literature vast. For the purpose of this paper, we restrict attention to simple nondeterministic tree transducers operating top-down, which transform trees by replacing each node with a subtree as specified by the label of the node and the state of the transduction at that node.

A NONDETERMINISTIC TOP-DOWN TREE TRANSDUCER ($\downarrow TT$) is a tuple $\langle Q, \mathcal{F}_{in}, \mathcal{F}_{out}, \Delta, q_0\rangle$ where

- Q is a finite set of STATES;
- $\mathcal{F}_{in}$ is a ranked alphabet of INPUT SYMBOLS;
- $\mathcal{F}_{out}$ is a ranked alphabet of OUTPUT SYMBOLS;
- Δ is a set of TRANSITIONS each of the form

  $$q(f(x_1, \ldots, x_n)) \to C[q_1(x_1), \ldots, q_n(x_n)]$$

  for some $f \in \mathcal{F}_{in}$ of arity n, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n \in \mathcal{X}_n$, and $C \in \mathcal{T}(\mathcal{F}_{out}, \mathcal{X}_n)$;
- $q_0 \in Q$ is a distinguished INITIAL STATE.

Given a tree transducer $\langle Q, \mathcal{F}_{in}, \mathcal{F}_{out}, \Delta, q_0\rangle$ and two trees $t \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out} \cup Q)$ and $t' \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out} \cup Q)$, tree t DERIVES t' IN ONE STEP, notated $t \vdash t'$ if and only if there is a transition $u \to u' \in \Delta$ with $u \in \mathcal{T}(\mathcal{F}_{in} \cup Q, \mathcal{X}_n)$ and $u' \in \mathcal{T}(\mathcal{F}_{out} \cup Q, \mathcal{X}_n)$ and trees $C \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out} \cup Q, \mathcal{X}_1)$ and $u_1, \ldots, u_n \in \mathcal{T}(\mathcal{F}_{in} \cup \mathcal{F}_{out})$, such that

$$t = C[u[u_1, \ldots, u_n]]$$

and

$$t' = C[u'[u_1, \ldots, u_n]] \quad .$$

The TREE RELATION defined by a $\downarrow TT$ $\langle Q, \mathcal{F}_{in}, \mathcal{F}_{out}, \Delta, q_0\rangle$ is the set of all tree pairs $\langle s, t\rangle \in \mathcal{T}(\mathcal{F}_{in}) \times \mathcal{T}(\mathcal{F}_{out})$ such that $q_0(s) \vdash^* t$.

For instance, the following rules specify a transducer that "rotates" subtrees of the form $f(t_1, f(t_2, t_3))$ to the tree $f(f(t_1, t_2), t_3)$. (By convention, we take the left-hand state of the first rule as the start state for the transducer.)

$$\begin{aligned} q(f(x, y)) &\to f(f(q(x), q_1(y)), q_2(y)) \\ q_1(f(x, y)) &\to q(x) \\ q_2(f(x, y)) &\to q(y) \\ q(a) &\to a \\ q(b) &\to b \end{aligned}$$

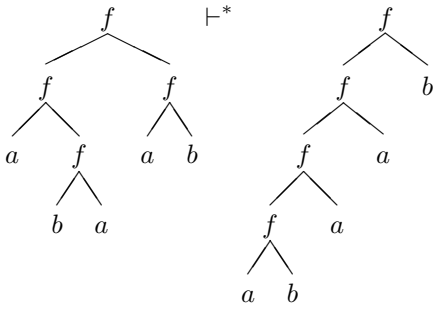


Figure 1: Local rotation computed by a nonlinear tree transducer

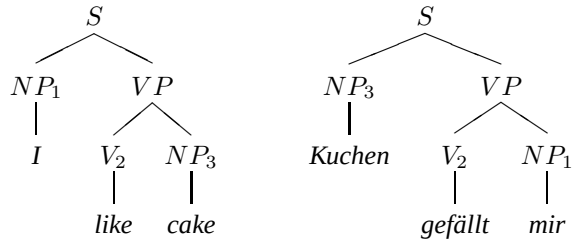


Figure 2: Example of local rotation in language translation divergence. Corresponding nodes are marked with matched subscripts.

The tree $f(f(a, f(b, a)), f(a, b))$ is transduced to $f(f(f(f(a, b), a), a), b)$ (as depicted graphically in figure 1) according to the following derivation:

$$
\begin{aligned}
& q(f(f(a, f(b, a)), f(a, b))) \\
& \quad \vdash \quad f(f(q(f(a, f(b, a))), q_1(f(a, b))), q_2(f(a, b))) \\
& \quad \vdash \quad f(f(f(\ f(q(a), q_1(f(b, a))), \\
& \qquad\qquad\quad q_2(f(b, a)) \ ), q(a)), q(b)) \\
& \quad \vdash \quad f(f(f(f(a, q(b)), q(a)), a), b) \\
& \quad \vdash \quad f(f(f(f(a, b), a), a), b)
\end{aligned}
$$

3.1 Nonlinearity Deprecated

Note that intrinsic use is made in this example of the ability to duplicate variables on the right-hand sides of rewrite rules. Transducers without such duplication are *linear*. Linear tree transducers are incapable of performing local rotations of this sort.

Local rotations are typical of natural-language applications. For instance, many of the kinds of translation divergences between languages, such as that exemplified in Figure 2, manifest such rotations. Similarly, semantic bracketing paradoxes can be viewed as necessitating rotations. Thus, linear tree transducers are insufficient for NL modeling purposes.

Nonlinearity per se, the ability to make copies during transduction, is not the kind of operation that is characteristic of natural-language phenomena. Furthermore, nonlinear transducers are computationally problematic. The following nonlinear transducer generates a perfect binary tree whose height is identical to that of its single-strand input.

$$
\begin{array}{lcl}
q(f(x)) & \rightarrow & g(q(x), q(x)) \\
q(a) & \rightarrow & a
\end{array}
$$

For instance, the tree of height and size four, $f(f(f(a)))$, transduces to $g(g(g(a, a), g(a, a)), g(g(a, a), g(a, a)))$, of height four but with fifteen symbols. The size of this transducer's output is exponential in the size of its input. (The existence of such a transducer constitutes a simple proof of the lack of composition closure of tree transducers, as the exponential of an exponential grows faster than exponential.)

In summary, nonlinearity seems inappropriate on computational and linguistic grounds, yet is required for tree transducers to express the kinds of simple local rotations that are typical of natural-language transductions. By contrast, STSG, as described below, is intrinsically a linear formalism but can express rotations straightforwardly.

3.2 Tree Automata and Homomorphisms

Two subcases of tree transducers are especially important. First, tree transducers that implement the identity relation over their domain are TREE AUTOMATA. A tree is in the language specified by a tree automaton if it is transduced to itself by the automaton. The tree languages so recognized are the regular tree languages (or recognizable tree languages), and are coextensive with those definable by context-free grammars. We take tree automata to be quadruples by dropping one of the redundant alphabets from the corresponding tree transducer quintuple.

Second, TREE HOMOMORPHISMS are essentially tree transducers with only a single state, so that the replacement of a node by a subtree proceeds independently of its context. A homomorphism $h : \mathcal{T}(\mathcal{F}_{in}) \rightarrow \mathcal{T}(\mathcal{F}_{out})$ is specified by its kernel, a function $\hat{h} : \mathcal{F}_{in} \rightarrow \mathcal{T}(\mathcal{F}_{out}, \mathcal{X}_{\infty})$ such that $\hat{h}(f)$ is a tree in $\mathcal{T}(\mathcal{F}_{out}, \mathcal{X}_{arity(f)})$ for each symbol $f \in \mathcal{F}_{in}$. The kernel $\hat{h}$ is extended to the homomorphism h by the following recurrence:

$$h(f(t_1, \ldots, t_n)) = \hat{h}(f)[h(t_1), \ldots, h(t_n)]$$

that is, $\hat{h}(f)$ acts as a context in which the homomorphic images of the subtrees are substituted. Further restrictions can be imposed: A tree homomorphism h is LINEAR if $\hat{h}(f)$ is linear for all $f \in \mathcal{F}_{in}$; is COMPLETE if $\hat{h}(f)$ contains every variable in $\mathcal{X}_{arity(f)}$ for all $f \in \mathcal{F}_{in}$; is ϵ-FREE if $\hat{h}(f) \notin \mathcal{X}_{arity(f)}$ for all $f \in \mathcal{F}_{in}$;

is SYMBOL-TO-SYMBOL if $\hat{h}(f)$ has exactly one symbol, for all $f \in \mathcal{F}_{in}$; and is a DELABELING if h is complete, linear, and symbol-to-symbol.

The import of these two subcases of tree transducers lies in the fact that the tree relations definable by tree transducers have been shown also to be characterizable by composition from these simplified forms, via an alternate quite distinct formalization based on bimorphisms. A BIMORPHISM is a triple $\langle L, h_{in}, h_{out}\rangle$ consisting of a regular tree language and two tree homomorphisms. The tree relation defined by a bimorphism consists of all pairs of trees generable by applying the homomorphisms to elements of the tree language, that is, $\{\langle h_{in}(t), h_{out}(t)\rangle \mid t \in L\}$. Depending on the type of tree homomorphisms used in the bimorphism, different classes of tree relations are defined. In particular, if we restrict h_{in} to be a delabeling, the tree relations defined are exactly those definable by $\uparrow TT$. As a convenient notation for bimorphisms, we write $B(X, Y)$ for the class of bimorphisms where h_{in} is restricted to have property X and h_{out} to have property Y. We use the following abbrevations for the properties: L[inear], C[omplete], [ϵ-]F[ree], S[ymbol-to-symbol], D[elabeling], M[orphism without restriction]. Thus the tree relations $B(D, M)$ are exactly those definable by $\uparrow TT$. (See the survey by Comon et al. (1997) and works cited therein.) Though many classes of bimorphisms have been studied, to our knowledge, the class $B(LC, LC)$ investigated below has not.

4 Synchronous Grammars and Bimorphisms

Tree-substitution grammars are composed of a set of elementary trees over a nonterminal and terminal vocabulary, allowing for nonterminal nodes at the leaves at which substitution of other elementary trees can occur (SUBSTITUTION NODES). They can be thought of as tree-adjoining grammars with substitution but no adjunction (hence no auxiliary trees). A synchronous tree-substitution grammar extends a tree-substitution grammar with the synchronization idea presented by Shieber (1992). In particular, grammars are composed of pairs of elementary trees, and pairs of substitution nodes, one from each tree in a pair, are linked to indicate that substitution of trees from a single elementary pair must occur at the linked nodes.

4.1 Tree-Substitution Grammars

A TREE-SUBSTITUTION GRAMMAR (TSG) comprises a set of ELEMENTARY TREES over a ranked alphabet $\mathcal{F}$, where certain frontier nonterminal (non-zero arity) nodes are marked as sites of substitution. The ability to have such nonterminal nodes with no children means that we must augment the definition of well-formed trees. We define the set of SUBSTITUTABLE TREES OVER A RANKED ALPHABET $\mathcal{F}$, notated $\mathcal{T}_{\downarrow}(\mathcal{F})$ as the smallest set such that

Nullary symbols at leaves $f \in \mathcal{T}_{\downarrow}(\mathcal{F})$ for all $f \in \mathcal{F}_0$;

Substitution nodes at leaves $f_{\downarrow} \in \mathcal{T}_{\downarrow}(\mathcal{F})$ for all $f \in F_n, n > 0$;

Internal nodes $f(t_1, \ldots, t_n) \in \mathcal{T}_{\downarrow}(\mathcal{F})$ for all $f \in \mathcal{F}_n$, $n \geq 1$, and $t_1, \ldots, t_n \in \mathcal{T}_{\downarrow}(\mathcal{F})$.

The marker $\downarrow$ marks the substitution nodes. In order to refer to the substitution nodes of a substitutable tree, we define the substitution paths of a tree t, $\downarrow\mathit{paths}(t)$ to comprise the paths to substitution nodes in t.

A tree-substitution grammar, then, is a triple, $\langle \mathcal{F}, P, S\rangle$ where $\mathcal{F}$ is a ranked alphabet comprising the vocabulary of the grammar, $S \in \mathcal{F}$ is the start symbol of the grammar, and $P \subseteq \mathcal{T}_{\downarrow}(\mathcal{F})$ is a set of elementary trees. In order to allow reference to a particular tree in the set P, we associate with each tree in P a unique index, conventionally notated with a subscripted α. This further allows us to have multiple instances of a tree in P, distinguished by their index. (We will abuse notation by using the index and the tree that it names interchangably.) Furthermore, we will assume that each grammar comes with an arbitrary ordering on the substitution node paths of a tree α_i, notating this permutation of $\downarrow\mathit{paths}(\alpha_i)$ by $\overline{\downarrow\mathit{paths}}(\alpha_i)$. We use this to mandate the child ordering of the children in derivation trees.

As a simple example, we consider the grammar with three elementary trees

$$\begin{array}{ll} \alpha_1 & S(NP_{\downarrow}, VP(V(like), NP_{\downarrow})) \\ \alpha_2 & NP(I) \\ \alpha_3 & NP(cake) \end{array}$$

and start symbol S. The arities of the symbols should be clear from their usage.

A DERIVATION for a grammar $G = \langle \mathcal{F}, P, S\rangle$ is a tree whose nodes are labeled with (indexes of) elementary trees, that is, a tree D in $\mathcal{T}(P)$, satisfying the following conditions:

1. For each node α in the tree D with substitution paths $\overline{\downarrow\mathit{paths}}(\alpha) = \langle p_1, \ldots, p_n\rangle$, the node must have n immediate children $\alpha_1, \ldots, \alpha_n$.

2. The root node of each child tree must match the corresponding substitution node in the parent, that is,

$$\alpha @ p_i = (\alpha_i @ \epsilon)_{\downarrow} \tag{1}$$

for all i, $1 \leq i \leq n$.

3. The tree α_r at the root of the derivation tree must be labeled at its root by the start symbol, that is, $\alpha_r @ \epsilon = S$.

For example, the derivation tree $\alpha_1(\alpha_3, \alpha_2)$ is a well-formed derivation tree for the sample grammar above, assuming that $\overline{\downarrow \mathit{paths}}(\alpha_1) = \langle\langle 2,2\rangle, \langle 1\rangle\rangle$. Note, for instance, that $\alpha_1 @ \langle 2,2\rangle = NP = \alpha_3 @ \epsilon$.

The derived tree for a derivation tree D is generated by performing all of the requisite substitutions. This can be defined directly, but to highlight the relationship with homomorphisms, we define it by mapping the substitutable trees into contexts, using a homomorphism kernel $\hat{h}_D$. For each tree $\alpha \in P$, with $\overline{\downarrow \mathit{paths}}(\alpha) = \langle p_1, \ldots, p_n\rangle$, $\hat{h}_D(\alpha)$ is the tree generated by replacing each node at address p_i by the variable x_i. For example, the context corresponding to the elementary tree $S(NP_\downarrow, VP(V(like), NP_\downarrow))$ with respect to the assumed substitution path ordering $\langle\langle 2,2\rangle, \langle 1\rangle\rangle$ is $S(x_2, VP(V(like), x_1))$. Because the substitution nodes of a tree all occur at its frontier, $\hat{h}_D(\alpha)$ is always a tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$, and by construction is linear and complete. Hence, the associated homomorphism h_D is also linear and complete.

We define the derived tree corresponding to a derivation tree D as the application of this homomorphism to D, that is $h_D(D)$. For the example above, the derived tree is that shown in Figure 2(a):

$$\begin{aligned} & h_D(\alpha_1(\alpha_3, \alpha_2)) \\ &= \hat{h}_D(\alpha_1)[h_D(\alpha_3), h_D(\alpha_2)] \\ &= S(x_2, VP(V(like), x_1))[\alpha_3, \alpha_2] \\ &= S(NP(I), VP(V(like), NP(cake))) \end{aligned}$$

4.2 Synchronous Tree-Substitution Grammars

We perform synchronization of tree-substitution grammars as per the approach taken for synchronizing tree-adjoining grammars in earlier work (Shieber, 1992). Synchronous grammars consist of pairs of elementary trees with a linking relation between nodes in one tree and nodes in the other. Simultaneous composition operations occur at linked nodes. In the case of synchronous tree-substitution grammars, the composition operation is substitution, so the linked nodes are substitution nodes.

We define a synchronous tree-substitution grammar, then, as a quintuple $G = \langle \mathcal{F}_{in}, \mathcal{F}_{out}, P, S_{in}, S_{out}\rangle$, where

- $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ are the input and output ranked alphabets, respectively,
- $S_{in} \in \mathcal{F}_{in}$ and $S_{out} \in \mathcal{F}_{out}$ are the input and output start symbols, and
- P is a set of elementary linked tree pairs, each of the form $\langle t, t', \frown\rangle$, where $t \in \mathcal{T}_\downarrow(\mathcal{F}_{in})$ and $t' \in \mathcal{T}_\downarrow(\mathcal{F}_{out})$ are input and output substitutable trees and $\frown \subseteq \downarrow\mathit{paths}(t) \times \downarrow\mathit{paths}(t')$ is a relation over substitution nodes from the two trees.

In order to guarantee that derivations for the synchronized grammars are isomorphic, we need to impose consistent orderings on the substitution nodes for paired trees. We therefore choose an arbitrary ordering $\langle p_{in,1} \frown p_{out,1}, \ldots, p_{in,n} \frown p_{out,n}\rangle$ over the linked pairs, and take $\overline{\downarrow \mathit{paths}}(t) = \langle p_{in,1}, \ldots, p_{in,n}\rangle$ and $\overline{\downarrow \mathit{paths}}(t') = \langle p_{out,1}, \ldots, p_{out,n}\rangle$.

We define $G_{in} = \langle \mathcal{F}_{in}, P_{in}, S_{in}\rangle$ where $P_{in} = \{t \mid \langle t, t', \frown\rangle \in P\}$; this is the left projection of the synchronous grammar onto a simple TSG. The right projection G_{out} can be defined similarly.

A synchronous derivation was originally defined as a pair $\langle D_{in}, D_{out}\rangle$ where (following Shieber (1992)):[1]

1. D_{in} is a well-formed derivation tree for G_{in}, and D_{out} is a well-formed derivation tree for G_{out}.
2. D_{in} and D_{out} are isomorphic.

The derived tree pair for a derivation $\langle D_{in}, D_{out}\rangle$ is then $\langle h_D(D_{in}), h_D(D_{out})\rangle$.

5 The Bimorphism Characterization of STSG

The central result we provide relating STSG to tree transducers is this: STSG is equivalent to $B(LC, LC)$. To show this, we must demonstrate that any STSG is reducible to a bimorphism, and vice versa.

5.1 Reducing STSG to $B(LC, LC)$

Given an STSG $G = \langle \mathcal{F}_{in}, \mathcal{F}_{out}, P, S_{in}, S_{out}\rangle$, we need to construct a bimorphism characterizing the same tree relation. All the parts are in place to do this. We start by recasting derivations as single derivation trees from which the left and right derivation trees can be projected via homomorphisms. Rather than taking a derivation to be a pair of isomorphic trees D_{in} and D_{out}, we take it to be the single tree D isomorphic to both, whose element at address p is $D @ p = \langle D_{in} @ p, D_{out} @ p\rangle$. Condition (2) on the well-formedness of a synchronous derivation thus being trivially satisfied, we simply need to require that the trees obtained by projecting this new derivation tree on its first and second elements are well-formed derivation trees in the projected TSGs. These projections D_{in} and D_{out} can be reconstructed by homomorphisms extending h_{in}

[1] In the earlier version, a third condition required that the isomorphic operations are sanctioned by links in tree pairs. This condition can be dropped here, as it follows from the previous definitions. In particular, since the substitution path orderings are chosen to be compatible, it follows that the isomorphic children of isomorphic nodes are substituted at linked paths.

that projects on the first component and h_{out} that projects on the second, respectively. These homomorphisms are trivially linear and complete (indeed, they are mere delabelings). Then the paired derived trees can be constructed as $h_D(h_{in}(D))$ and $h_D(h_{out}(D))$, respectively. Thus the mappings from the derivation tree to the derived trees are the compositions of two linear complete homomorphisms, hence linear complete homomorphisms themselves. We take the bimorphism characterizing the STSG tree relation to be $\langle L_D, h_D \circ h_{in}, h_D \circ h_{out} \rangle$ where L_D is the language of well-formed synchronous derivation trees.

To show that the language L_D is a regular tree language, we construct a top-down nondeterministic automaton $\langle Q_G, \mathcal{F}_G, \Delta_G, q_G \rangle$ recognizing it. The states of the automaton Q_G are elements of $\mathcal{F}_{in} \times \mathcal{F}_{out}$, expressing the allowable pair of symbols labeling the roots of the tree pair dominated by the state. The start state is $q_0 = \langle S_{in}, S_{out} \rangle$. The alphabet $\mathcal{F}_G$ of the trees is composed of pairs $\langle \alpha_{in}, \alpha_{out} \rangle$ of elementary trees, such that $\langle \alpha_{in}, \alpha_{out}, \frown \rangle \in P$, the arity of which is the number of substitution nodes in each tree, or equivalently, $|\frown|$. For each elementary tree pair $\langle \alpha_{in}, \alpha_{out}, \frown \rangle \in P$, where $\overline{\downarrow \mathit{paths}}(\alpha_{in}) = \langle p_1, \ldots, p_n \rangle$ and $\overline{\downarrow \mathit{paths}}(\alpha_{out}) = \langle r_1, \ldots, r_n \rangle$, there is a single transition in Δ_G of the form:

$$\begin{aligned}\langle \alpha_{in}@\epsilon, \alpha_{out}@\epsilon \rangle(\langle \alpha_{in}, \alpha_{out} \rangle(x_1, \ldots, x_n)) \\ \quad \rightarrow \langle \alpha_{in}, \alpha_{out} \rangle(\ \langle \alpha_{in}@p_1, \alpha_{out}@r_1 \rangle(x_1), \ldots, \\ \langle \alpha_{in}@p_n, \alpha_{out}@r_n \rangle(x_n)\ )\end{aligned}$$

We must verify that for any tree D recognized by this automaton $h_{in}(D)$ and $h_{out}(D)$ are well-formed derivation trees for their respective TSGs.

To show that $h_{in}(D)$ is a well-formed derivation tree (and symmetrically, for $h_{out}(D)$), we must demonstrate that the three definitional conditions hold. Consider a node in the tree of the form $\langle \alpha_{in}, \alpha_{out} \rangle$. This node must have been admitted by virtue of some transition of the form above.

1. By construction, there must be an elementary tree pair $\langle \alpha_{in}, \alpha_{out}, \frown \rangle \in P$, and the node must have n immediate children corresponding to $\overline{\downarrow \mathit{paths}}(\alpha_{in}) = \langle p_1, \ldots, p_n \rangle$.

2. Each child node, say the i-th, which we can notate $\langle \alpha_{in,i}, \alpha_{out,i} \rangle$, again by construction, must be admitted by a transition of the form $\langle \alpha_{in}@p_i, \alpha_{out}@r_i \rangle(\langle \alpha_{in,i}, \alpha_{out,i} \rangle(\cdots))$. Any matching transition enforces the requirement that $\langle \alpha_{in}@p_i, \alpha_{out}@r_i \rangle = \langle \alpha_{in,i}@\epsilon, \alpha_{out,i}@\epsilon \rangle$ hence that $\alpha_{in}@p_i = (\alpha_{in,i}@\epsilon)_{\downarrow}$ and $\alpha_{out}@r_i = (\alpha_{out,i}@\epsilon)_{\downarrow}$, as required.

3. Since the start state is $\langle S_{in}, S_{out} \rangle$, the root of the derivation tree must be a node $\langle \alpha_{in,r}, \alpha_{out,r} \rangle$ such that $\alpha_{in,r}@\epsilon = S_{in}$ and $\alpha_{out,r}@\epsilon = S_{out}$.

Thus, each of the two projection trees $h_{in}(D)$ and $h_{out}(D)$ are well-formed derivation trees for their respective grammars, and the tree relation defined by the STSG is in $B(LC, LC)$.

5.2 Reducing $B(LC, LC)$ to STSG

The other direction is somewhat trickier to prove, but can be done. Given a bimorphism $\langle L, h_{in}, h_{out} \rangle$ over input and output alphabets $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$, respectively, we construct a corresponding STSG $G = \langle \mathcal{F}'_{in}, \mathcal{F}'_{out}, P, S_{in}, S_{out} \rangle$. By "corresponding", we mean that the tree relation defined by the bimorphism is obtainable from the tree relation defined by the STSG via delabelings of the input and output that map $\mathcal{F}'_{in}$ to $\mathcal{F}_{in}$ and $\mathcal{F}'_{out}$ to $\mathcal{F}_{out}$. (Recall that delabelings are just many-to-one renamings of the symbols.)

As the language L is a regular tree language, it is generable by a nondeterministic top-down tree automaton $\langle Q, \mathcal{F}_d, \Delta, q_0 \rangle$. We use the states of this automaton in the input and output alphabets of the STSG. The input alphabet of the STSG is $\mathcal{F}'_{in} = \mathcal{F}_{in} \cup (Q \times \mathcal{F}_{in})$, composed of the input symbols of the bimorphism, along with some special symbols that pair states with the input symbols, and similarly for the output alphabet. The pair symbols mark the places in the tree where substitutions occur, allowing control for appropriate substitutions. In order to generate the trees actually related by the original bimorphism, the nodes labeled with such pairs can be projected on their second component by a simple delabeling.

The basic idea of the STSG construction is to construct an elementary tree pair for certain *sequences* of transitions from Δ. However, it is easiest understood by starting with the construction for the special case in which the homomorphisms are ϵ-free. In this case, as we will see, the pertinent sequences are just the single transitions. For the nonce, then, we assume h_{in} and h_{out} to be ϵ-free, relaxing this assumption later.

We define a simple nondeterministic transformation on trees in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$ controlled by a sequence of $n+1$ states in Q:

$$\begin{aligned}\mathcal{C}(f(t_1, \ldots, t_k), q, q_1, \ldots, q_n) \\ = \{\langle q, f \rangle(t_1, \ldots, t_k)[\langle q_1, N_1 \rangle_{\downarrow}, \ldots, \langle q_n, N_n \rangle_{\downarrow}] \\ \mid N_1, \ldots, N_n \in \mathcal{F}\}\end{aligned}$$

In essence, the transformation replaces the root symbol by pairing it with the state q, and replaces the n variables with new pairs of a state q_i and an arbitrarily chosen symbol N_i. (The nondeterminism arises in the choice of the N_i.) These latter symbols are taken to be substitution

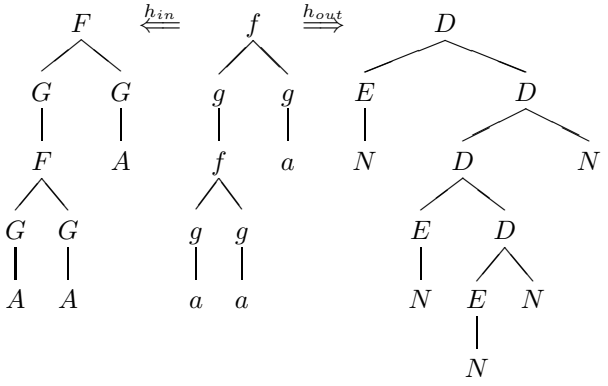


Figure 3: Example of bimorphism construction

nodes in the generated tree. Importantly, this transformation is partial; it applies to any tree in $\mathcal{T}(\mathcal{F}, \mathcal{X}_n)$, with the exception of those trees that consist of a variable alone.

We use the transformation $\mathcal{C}$ to generate elementary tree pairs corresponding to transitions in Δ. For each transition $q(f(x_1, \ldots, x_n)) \rightarrow f(q_1(x_1), \ldots, q_n(x_n)) \in \Delta$, we construct the elementary tree pairs $\langle t_{in}, t_{out}, \frown \rangle$, where $t_{in} \in \mathcal{C}(\hat{h}_{in}(f), q, q_1, \ldots, q_n)$ and $t_{out} \in \mathcal{C}(\hat{h}_{out}(f), q, q_1, \ldots, q_n)$ and $\frown$ links the corresponding paths in the two trees, that is, the paths at which corresponding variables occur in the trees $\hat{h}_{in}(f)$ and $\hat{h}_{out}(f)$. Since h_{in} and h_{out} are linear and complete, this notion is well-defined. The applications of $\mathcal{C}$ are well-defined only when $\hat{h}_{in}(f)$ and $\hat{h}_{out}(f)$ are in the domain of $\mathcal{C}$, that is, it is not a lone variable, hence the requirement that h_{in} and h_{out} be ϵ-free.

An example may clarify the construction. Take the language of the bimorphism to be defined by the following two-state automaton:

$$\begin{array}{rcl} q(f(x,y)) & \rightarrow & f(q'(x), q'(y)) \\ q(a) & \rightarrow & a \\ q'(g(x)) & \rightarrow & g(q(x)) \end{array}$$

This automaton uses the states to alternate g's with f's and a's level by level. For instance, it admits the middle tree in Figure 3. With input and output homomorphisms defined by

$$\begin{array}{rclrcl} \hat{h}_{in}(f) & = & F(x,y) & \hat{h}_{out}(f) & = & D(y, D(x,N)) \\ \hat{h}_{in}(g) & = & G(x) & \hat{h}_{out}(g) & = & E(x) \\ \hat{h}_{in}(a) & = & A & \hat{h}_{out}(a) & = & N \end{array}$$

the bimorphism so defined generates the tree relation instance exemplified in the figure.

The construction given above generates the schematic elementary tree pairs in Figure 4 for this bimorphism. (The tree pairs are schematic in that we use a $*$ to stand for an arbitrary symbol in the appropriate alphabet.) The reader can verify that the grammar generates a tree pair whose delabeling is that shown in Figure 3 generated by the bimorphism.

Now, we turn to the considerably more subtle considerations of non-ϵ-free homomorphisms. In a linear complete homomorphism, the only possible case of non-ϵ-freeness that is possible is for unary function symbols, that is $\hat{h}(f) = x$, so that $h(f(x)) = h(x)$. Intuitively speaking, such cases in bimorphisms should (and will) correspond to STSG elementary trees that have just a single node, so that they contribute no structure to the derived trees.

If, for some symbol f, both h_{in} and h_{out} are non-ϵ-free, then any tree rooted in such a symbol, $f(t)$, is mapped, respectively, to $h_{in}(t)$ and $h_{out}(t)$. But in that case, we can eliminate the unary symbol f, eliminating transitions in the automaton of the form $q(f(x)) \rightarrow f(q'(x))$ by adding, for all transitions with q' on the left hand side, identical transitions with q on the left-hand side. We then construct the STSG for the simplified automaton.

The situation is more complicated if only one of the two homomorphisms, say h_{in}, is non-ϵ-free. In this case, we have that $h_{in}(f(x)) = h_{in}(x)$ but $h_{out}(f(x)) = C[h_{out}(x)]$ for nontrivial context C, thus introducing structure on the output with no corresponding structure on the input. We will call such a unary symbol ASYMMETRIC. A sequence of asymmetric symbols can introduce unbounded amounts of material on the output with no corresponding material on the input (or vice versa). The key is thus to construct all possible such sequences of asymmetric symbols and chop them into a bounded set of minimal cycles, using these to generate single elementary tree pairs. We arrange that in such cycles, the state and symbol at the root will be identical to the state and symbol at the end of the sequence. For example, suppose we have asymmetric symbols f and g and an ϵ-free symbol k with the following automaton transitions:

$$\begin{array}{rcl} q(k(x)) & \rightarrow & k(q(x)) \\ q(f(x)) & \rightarrow & f(q(x)) \\ q(g(x)) & \rightarrow & g(q'(x)) \\ q'(f(x)) & \rightarrow & f(q'(x)) \\ q'(f(x)) & \rightarrow & f(q''(x)) \\ q'(g(x)) & \rightarrow & g(q'(x)) \\ q''(k(x)) & \rightarrow & k(\ldots) \end{array}$$

There is a minimal cycle such that $q'(f(g(f(x)))) = f(g(q'(f(x))))$. Note that the state q' and symbol f at the root are duplicated at the bottom. There is a similar cycle of the form $q'(f(f(x))) = f(q'(f(x)))$. For each such cycle, we construct a linked tree pair with a trivial input tree labeled with a pair of the state and an arbitrary symbol N from the input alphabet—$\langle q', N \rangle$ in

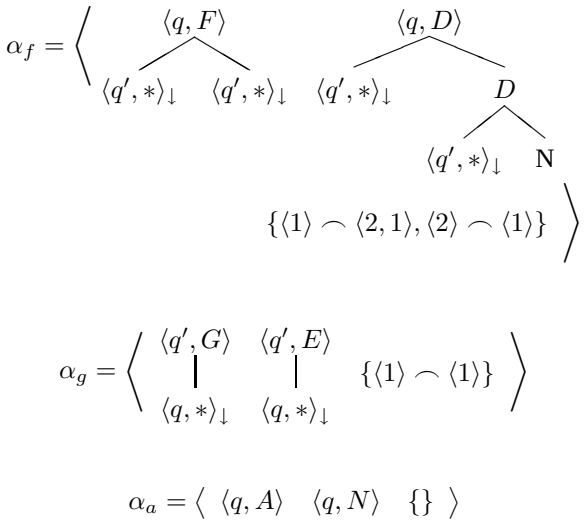


Figure 4: Generated STSG for example bimorphism

the example. The corresponding output tree is generated by composing the nontrivial output trees and applying $\mathcal{C}$ to this compound tree in the obvious way. Since the path language in the tree language of a tree automaton is regular, a decomposition of the paths into a bounded number of bounded-length cycles can always be done, leading to a finite number of elementary tree pairs. Note that since the label of the root for the appropriate input tree $\langle q', f\rangle$ is identical to the label to replace the (single) variable, the tree pair is constructed in a way consistent with $\mathcal{C}$, hence the workings of the rest of the STSG.

In addition, for each minimal sequence starting with a symbol that is non-ϵ-free on the input and leading to such a cyclic state/symbol pair, a tree pair is similarly generated. In the example, the sequence corresponding to the automaton subderivation $q(k(f(g(f(x))))) = k(f(g(q'(f(x)))))$ would lead us to generate a tree pair with $\langle \mathcal{C}(\hat{h}_{in}(k), q, q'), \mathcal{C}(\hat{h}_{in}(k)[\hat{h}_{out}(f)[\hat{h}_{out}(g)]], q, q'), \frown\rangle$ where $\frown$ links the two leaf nodes labeled with state/symbol pairs.

Similarly, we require elementary tree pairs corresponding to minimal tails of sequences of asymmetric symbols starting in a cyclic state/symbol pair and ending in a symbol non-ϵ-free on the input. These three types of sequences can be pieced together to form any possible sequence of unary symbols admitted by the automaton, and the corresponding tree pairs correspond to the compositions of the homomorphism trees.

6 Discussion

By placing STSG in the class of bimorphisms, which have already been used to characterize tree transducers, we provide the first synthesis of these two independently developed approaches to specifying tree relations, unifying their respective literatures for the first time. The relation between a TAG derivation tree and its derived tree is not a mere homomorphism. The appropriate morphism generalizing linear complete homomorphisms to allow adjunction can presumably be used to provide a bimorphism characterization of STAG as well, further unifying these strands of research.

The bimorphism characterization of STSG has immediate application. First, the symmetry of the tree relations defined by an STSG is a trivial corollary. Second, it has been claimed in passing that synchronous tree-substitution grammars are "equivalent to top-down tree transducers." (Eisner, 2003). This is clearly contravened by the distinction between $B(LC, LC)$ and $B(D, M)$. Third, the bimorphism characterization of tree transducers has led to a series of composition closure results. Similar techniques may now be applicable to synchronous formalisms, where no composition results are known. For instance, the argument for the lack of composition closure in $B(LCF, LCF)$ (Arnold and Dauchet, 1982) may be directly applicable to a similar proof for $B(LC, LC)$, hence for STSG; the conjecture remains for future work.

References

A. Arnold and M. Dauchet. 1982. Morphismes et bimorphismes d'arbres. *Theoretical Computer Science*, 20(1):33–93, March.

H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 1997. Tree automata techniques and applications. Available at: `http://www.grappa.univ-lille3.fr/tata`. release of October 1, 2002.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, July.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Stuart M. Shieber. 1992. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. In *Proceedings of the Second TAG Workshop*, University of Pennsylvania, Philadelphia, Pennsylvania, June 24–26.

Why Supertagging Is Hard

François Toussenel
Lattice – UMR 8094
University Paris 7
francois.toussenel@linguist.jussieu.fr

Abstract

Tree adjoining grammar parsers can use a supertagger as a preprocessor to help disambiguate the category[1] of words and thus speed up the parsing phase dramatically. However, since the errors in supertagging propagate to this phase, it is vital to keep the error rate of the supertagger phase reasonably low. With very large tagsets coming from extracted grammars, this error rate can be of almost 20%, using standard Hidden Markov Model techniques. To combat this problem, we can trade a higher precision for increased ambiguity in the supertagger output. I propose a new approach to introduce ambiguity in the supertags, looking for a suitable trade-off. The method is based on a representation of the supertags as a feature structure and consists in grouping the values, or a subset of the values, of certain features, generally those hardest to predict.

1 Introduction

This paper deals with supertagging[2] as a preprocessing step before full parsing.

A TAG parser has too many elementary trees to choose from if they are not at least partially disambiguated beforehand (Joshi and Bangalore, August 1994): the combinatorics at the parsing level are huge. As suggested in Srinivas Bangalore's Ph.D. thesis (Bangalore, 1997), supertagging may be used to reduce the high number of trees associated with each word. But to tag and parse real-world text, we need a sufficiently sized grammar. One convenient way to constitute a large TAG is to extract it from a hand-corrected treebank. Naturally, the resulting tagset for supertagging is also large. The problem thus becomes the fact that when the tagset is very large (e.g. about 5,000 different trees), the precision of the supertagger output is so low (about 80%) that the parser fails on most sentences.

The supertagger we use is based on a Hidden Markov Model (HMM) tagger trained on a grammar extracted (Chen, 2001) from the Wall Street Journal part of the Penn Treebank (Marcus et al., 1993) and the parser is the one described in (Nasr et al., 2002).

2 Supertagging and Very Large Tagsets

If HMM part of speech tagging has been proven quite successful, supertagging is more problematic for two main reasons.

- (A) The large number of categories which characterizes supertagging entails statistical problems, but for the result to be useful in helping parse real-world texts, a medium-sized or small grammar (with e.g. 300 or 400 different elementary trees) seems insufficient.
- (B) The non-local nature of the information included in the supertag clashes with the local vision of the HMM tagger (e.g. a three-word window). Indeed, supertags locally represent dependencies not represented in parts of speech. For instance, the supertag assigned to the verb *brought* in *I brought their children my son's old bicycle* will include a slot for each of the two complements, the second of which (*my son's old bicycle*) is beyond the three-word window in this sentence.

With a tagset of about 5,000 trees, HMM tagging techniques suffer from severe training data sparseness. Statistical problems arise that are little or not encountered in a regular part of speech tagging context. Indeed, various types of events are never seen in the training corpus. The simplest type is the supertag itself. Some supertags are new in the test corpus. Obviously, standard techniques cannot guess them.

[1] Specifically, a rich description of the syntactic properties of words.

[2] Supertagging consists in assigning an elementary tree (of a TAG) to each word of a sentence.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 96-103.

A more frequent type of unseen events is the association of known words with known supertags that did not occur together in the training corpus. About 5% of the word-supertag pairs are new, these pairs being involved in about a quarter of the errors[3]. John Chen (Chen, 2001) has addressed this problem and has designed tree families to automatically extend the grammar. In (Hockenmaier and Steedman, 2002), a Combinatory Categorial Grammar is extracted from the Penn Treebank and the authors have found a 26% reduction of "unseen pairings of seen words and seen categories" (from 3% to 2.2%) thanks to a reduction of the category inventory, and a 50% reduction (from 3% to 1.5%) when combining the reduced category inventory with a more elaborate treatment of unknown words (using the part of speech instead of a single token for unknown words).

Other existing solutions include reranking (Chen et al., 2002) and class tagging (Chen et al., 1999) (Chen, 2001), but either they are applied to smaller grammars (between 300 and 500 different trees) or they face problems similar to ours.

The reranking technique notably is not bound to a limited context and is thus complementary with an n-gram tagger.

3 Ambiguous Supertags

Failing to find the correct supertag often enough for the parse to succeed, we resort to allowing some ambiguity in the supertagger output. The main idea is to relieve the supertagger from a part of its disambiguating duty, deferring it to the parser which will make the final decisions (given that it has information about the whole sentence). The key point is finding a good trade-off between precision rate (for successful parses) and ambiguity (to keep the parsing phase tractable).

With the n-best tagging technique (Bangalore and Joshi, 1999), the supertagger outputs several trees (the most probable n supertags) and the parser chooses among them. One drawback is that the output consists in the same number of supertags for each word, regardless of its type (e.g. verb or adjective), whereas it seems attractive to keep more possible supertags for a verb than for less ambiguous words, for instance.

Previously we tested a kind of n-best supertagging on our grammar, but failed to achieve an error rate below 9.5%, which was unsatisfactory and led us to imagine harder ways to produce an ambiguous output.

[3]The results presented here have been computed from a supertagged portion of the Penn Treebank consisting of 1,939 sentences (about 50K words), the training corpus consisting of 37,858 sentences (about 980K words).

3.1 Underspecification Using a Feature Structure

The solution I propose introduces underspecification at the supertag level. In other words, the supertag conveys less information, but still more than in mere parts of speech. To do this I represent the trees as a feature structure in which the salient characteristics of a supertag are encoded, as was initially suggested in John Chen's Ph.D. (for another purpose) (Chen, 2001)[4].

The results presented here are from experiments using a structure of 18 features, among which are:

- the part of speech of the root node (26 possible values),
- the subcategorization (more than one hundred possible values),
- several transformational features,
- the two ordered lists of the nodes on the left and right frontiers,
- the list of internal nodes (neither the root nor the nodes on the frontier),
- the list of co-anchors (more than one hundred possible values),
- the part of speech of the modified word if this is a modifier,
- the direction of the modification if this is a modifier.

3.2 More on Two Features

Two features are of particular interest (both are pertinent only for modifier trees): one specifies the part of speech of the modified word and the other specifies the direction of the modification. It must be noted that both these features have an extra value (*NIL*) which means non-pertinent, for the case of a non-modifier word: thus predicting this feature involves predicting whether the word is a modifier. These are the most difficult features to predict (error rates of about 12.6% for the first with 38 possible values and almost 9% for the second with only 3 possible values). Moreover, predicting them makes the supertagging process much longer. However, as is shown below, knowing their values for a given supertag helps predict other features, including the part of speech.

3.3 Neutralizing Features

By neutralizing certain features describing the trees (i.e. not specifying the value for those features), we obtain an underspecified supertag (the tagset is therefore reduced), which is thus ambiguous but easier to predict.

[4]For my experiments I used John Chen's feature structures but my plans for future work involve the use of others.

This approach allows us to control the amount of information we are able and willing to supply the parser with[5]. This is particularly interesting since the error comes from a relatively small number of features each time (but the features which are incorrectly predicted are not always the same). Table 1 shows that 42% of the errors on trees[6] involve only up to two features.

Table 1: Cumulated proportion or errors due to n features incorrectly predicted (the remaining 6.3% is due to co-anchors).

# of features	% of errors (cumulated)
1	19.972
2	42.038
3	54.429
4	62.934
5	74.584
6	82.122
7	85.440
8	89.660
9	91.485
10	92.290
11	93.386
12	93.611
13 to 18	93.697

It is important to state that the feature neutralization must take place only after training and supertagging. Indeed, if the supertagger is trained on an "underspecified" annotated corpus, it gives worse results than if it is trained on a corpus annotated with regular supertags, its output then being modified to change the regular supertags into their underspecified versions. For instance, there is a 15% relative reduction of the error rate for the part of speech feature when we tag the whole supertag. This is due to the dependencies between the features: learning on more features helps predict one particular feature. Of course, if it is just to tag with part of speech, the whole process takes much more time than regular part of speech tagging. On the other hand, the precision is higher (the two features mentioned above are in a large part responsible for this).

3.4 Experiments on (Almost) All the Combinations

As a first trial in this direction I conducted experiments consisting in neutralizing series of sets of features to study the coordinated behavior of both the error rate and ambiguity according to the features neutralized. The combinatorics are rather large[7], but evaluating the error rate and ambiguity of the supertagged output with a given set of neutralized features takes very little time (about one or two seconds on a personal computer). Indeed, since the input text is tagged with the full supertag, there is no need to supertag the text for each set of neutralized features. We only have to extract the reduced information from the supertags in both the hypothesis and the reference and run the evaluation on it.

I decided never to neutralize the part of speech feature (numbered 0); I thus gathered the resulting 131,072 error rate/ambiguity pairs. To find a good trade-off between error rate and ambiguity, one needs to consider some candidate sets of neutralized features; a simple method to preselect some candidates is to search for the set of neutralized features yielding the lowest ambiguity for a (small) number of given maximum error rates. Figure 1 and Table 2 show the lowest ambiguity for each given maximum entire error rate, from 19% to 4%. These boundaries come from the error rate associated with no neutralization at all (18.64%) and the one associated with all the features neutralized except part of speech (3.67%).

The ambiguity figures are the average number of supertags (from the original tagset) represented by the underspecified tag for each word in the test corpus. Thus it depends on the tags chosen for each word, it is not just the ambiguity of the simplified grammar with regard to the original grammar.

The lowest error rate is a bit under 4%, associated with an average ambiguity above 450, and corresponds to a tag representing (a little more than) part of speech. We hope to find at least one set of neutralized features allowing for acceptable error rate and ambiguity. Error rates of 6% or 5% would seem suitable, but they are associated with an ambiguity of about 212 and 306 respectively. It is not sure that such high ambiguity can be handled by a processor with such input; in the case of a statistical parser, the resulting combinatorics would make it necessary to use an appropriate beam search. However, the accuracy of the parser is not guaranteed to be preserved with such a beam search.

3.5 The Incremental Method

Exploring the whole set of possible combinations is affordable when each test mainly involves translating tags. However, the performance of the supertagger in itself is not the only relevant measure when it comes to use its output as an input for a parser. To find the best trade-off between error rate and ambiguity, the most natural test is the performance of the parser. We would need both the accuracy of its output and the average time it takes to parse a sentence. These experiments are yet to be done,

[5] To do this we can neutralize certain features altogether or tag with a set of values for certain features instead of only one value for those features.

[6] not including errors on a co-anchor.

[7] A structure of 18 features entails $2^{18} = 262144$ possible sets of neutralizations.

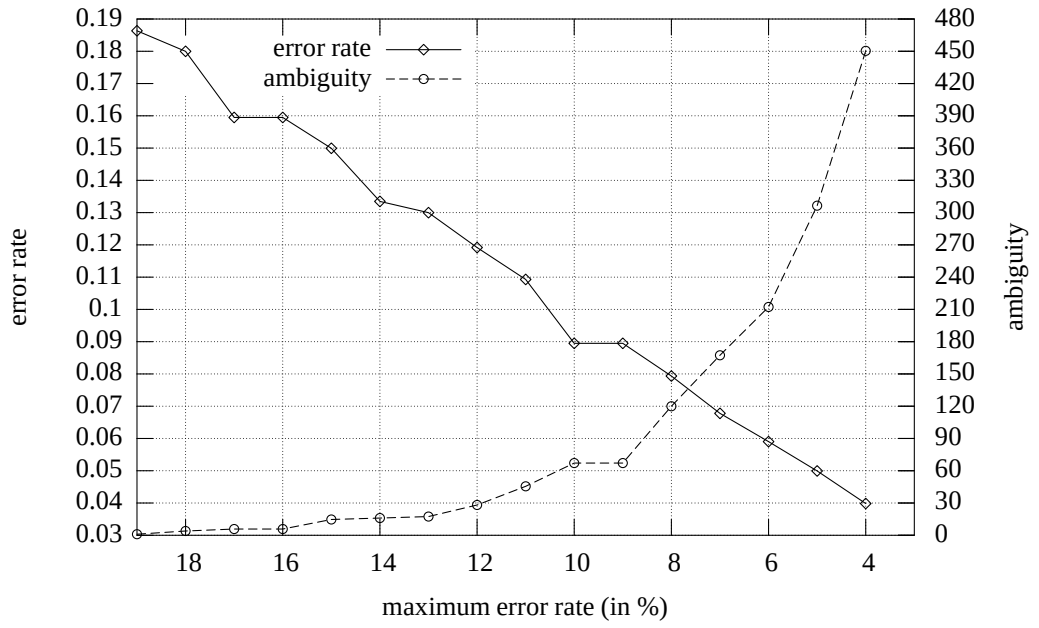


Figure 1: Lowest ambiguity per WER (from full set of combinations).

Table 2: Lowest ambiguity per WER (from full set of combinations). The third column (#) represents the number of neutralized features for the given set (detailed in the fourth column). The word error rate is given in %.

WER	Ambig.	#	Neutralized features
18.635	1.014	1	9
17.998	3.944	5	1 2 6 11 15
15.948	5.810	1	16
15.948	5.810	1	16
14.993	14.699	2	11 16
13.347	16.012	2	16 17
12.999	17.394	10	3 4 6 7 8 12 13 15 16 17
11.917	28.162	10	2 3 5 7 8 12 14 15 16 17
10.928	45.532	3	11 16 17
8.950	67.129	6	1 2 11 13 16 17
8.950	67.129	6	1 2 11 13 16 17
7.935	119.928	8	1 2 11 13 14 15 16 17
6.778	167.370	9	1 2 3 11 12 13 14 16 17
5.903	212.301	10	1 2 3 11 12 13 14 15 16 17
4.994	306.446	13	1 2 3 4 7 9 11 12 13 14 15 16 17
3.984	450.421	15	1 2 3 4 5 6 7 9 11 12 13 14 15 16 17

but one can already guess that the same thorough series of tests will probably not be tractable when testing the parser every time. Not only the tests will take the time of parsing, but this time will increase exponentially with the ambiguity. As a matter of fact, achieving a test in a limited time is a result in itself: it means the parser can handle the ambiguity.

Consequently, the tests must be run in the order of fastest to slowest. Also, we will not run all the tests but only those that have the best chances to reveal interesting results. That is, we need a method to select the combinations.

With this objective in mind, I designed an incremental method to choose which features are to be neutralized in order to minimize the error rate. I applied this method to the supertagger output, which gives a preview of its usefulness (I hope it will be as useful with the parser evaluation). There again, the result is a graduated trade-off between precision and ambiguity. We will compare it to our previous combinations.

I now describe the incremental method as I applied it on the supertagger output. The goal is to construct a number of sets of neutralized features, from a set of one feature to a set of 17 features (for a structure of 18 features). The main idea is follow the optimum "path" by selecting the most interesting feature to neutralize at each step, adding it to the previous set. Let S be the current set of neutralized features. I first decided to always keep the feature representing the part of speech of the anchor of the tree. So the second step was to add one of the 17

remaining features to the (yet empty) set S. To choose this feature, each of the candidate features is temporarily added to S and the corresponding error rate and ambiguity are computed. The feature leading to the best result is then selected and permanently added to S. The process is repeated with the remaining features until there are no more features to neutralize and only the part of speech (which is our baseline) remains.

The number of tests required by this method is only $\sum_{i=1}^{17} i = 154$ instead of $2^{17} = 131,072$ for the full set of combinations.

The search for the next feature to neutralize can be driven by three types of criteria: the error rate, the ambiguity, or a combination of the two. I tried the first two criteria, which selected different features but yielded similar trade-offs.

3.6 Experiments on the Incremental Method Applied to the Supertagger

Figure 3 shows the linked progression of the error rate and ambiguity, using the error rate criterium to select each feature to neutralize. Here the relevant curves are those marked as *feature*. We will see *values* below.

It is interesting to see how the incremental method behaves compared with the full set of combinations. Figure 2 compares the two corresponding curves (features being either fully neutralized or not at all). Let us first note that the incremental method's curve is very similar to the curve obtained by selecting the lowest error rate per number of neutralized features, as opposed to the average error rates per number of neutralized features. Indeed, only 4 out of 17 sets of neutralized features are different in the two curves.

Let's take a closer look at those four couples of sets in Table 3. What happened is that for the lowest error rate from the full set of combinations, the set of 6 neutralized features (1 2 11 13 16 17) has only 3 features (11 16 17) in common with the set of 5 neutralized features (11 12 14 16 17). Of course, the incremental method cannot compete with this performance because it keeps the whole previous set by design.

What's more, both the error rate and the ambiguity are lower for the set of 6 features than for the set of 5 features. For the other sets, the balance between error rate and ambiguity is regular again, and new features are just added to the previous set, just like in the incremental method.

The combinations of error rate and ambiguity drop from 18.64%/1 with no neutralization at all to 3.67%/509 for just part of speech. The point of the method is to choose an intermediate value (the best trade-off). For example, with 11 neutralized features, we have 5.17%/284, and for 10 neutralized features, 5.9%/212.

3.7 Refinement

A slight improvement of this method can be achieved by neutralizing only part of the features as opposed to the features altogether. In other words, instead of grouping all the values for a given feature, we can group some values. For instance, consider the three-value feature *direction of modification*. The three possible values are *left*, *right* or *NIL* (in case this is not a modifier). We could group the first two values, which would result in a binary feature simply indicating whether this is a modifier. The selection of values to group can be done according to error analysis. We group the values which the supertagger most often confuses.

To evaluate the power of this improved method, I used the same test corpus for both the error analysis and the new evaluation with grouped values, which one cannot do when (super)tagging new text but this shows the maximum gain we can get thanks to this refined method.

On Figure 3, the relevant curves are those marked as *values*. The error rate curve is the same as the old one, since all values which were confused by the supertagger on this test corpus, and only these values, were grouped. Only ambiguity is different, and naturally always lower or equal, since the supertags represented by the underspecified tags have all their features present, only with ambiguous values for some of them.

To compare with the previous results, with 11 neutralized features, ambiguity drops from 284 down to 248, and for 10 features, from 212 down to 185.

As we can see, the ambiguity associated with acceptable error rates is still quite large, even with the refined method. This seems to indicate that this kind of approach is not sufficient. Replacing the error rate criterium with the parser's accuracy, as was explained above, will probably highlight better trade-offs, but it seems likely that the improvement will be limited.

3.8 Feature Structures

All my experiments were based on John Chen's various feature structures. I believe the choice of a feature structure must have a noticeable (but somewhat limited) influence on the results one can get from playing with ambiguity in the way described in this paper. But there is more to it: the extracted grammar itself is determined by the feature structure. The grammar I used is very close to what was seen in the training corpus. A good deal of generalization can be done, though, and this would probably entail lower error rates for the same amount of ambiguity. Metagrammars (Candito, 1999) and their associated feature structures are designed in this spirit. First, features are drawn, then a generalization phase takes place, and finally the grammar is extracted. Thus unseen supertags are less likely to appear in a test corpus.

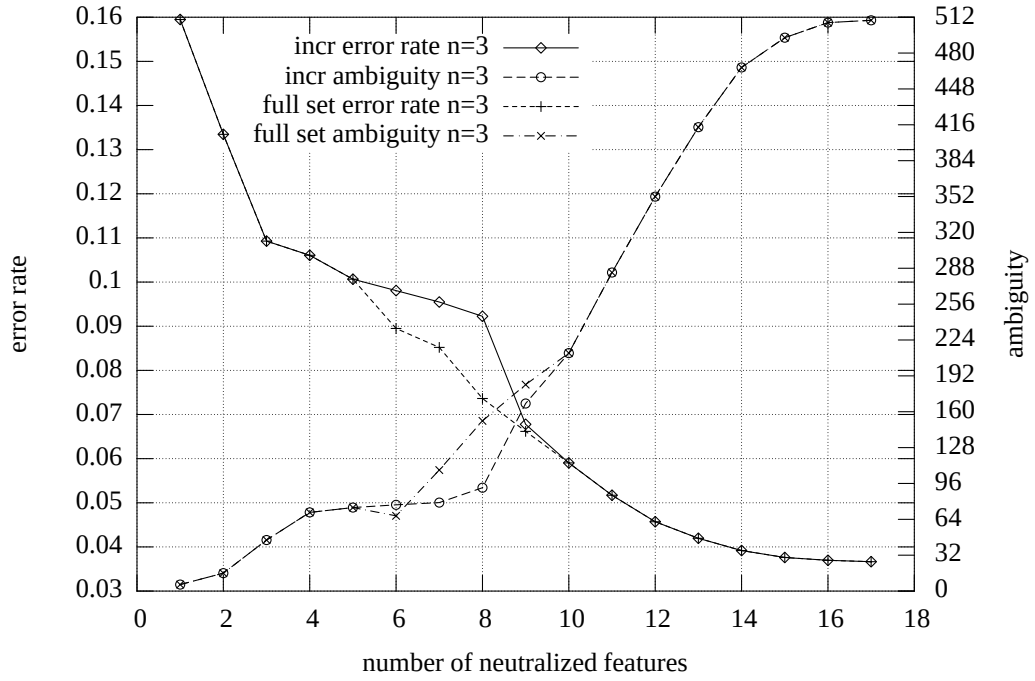


Figure 2: Linked progression of error rate and ambiguity, for whole *feature* neutralization, to compare the incremental method with the full set.

Table 3: Different sets between incremental method and whole combination (lowest error rate).

	Whole combination			Incremental		
#	WER	Ambig.	Neutralized features	WER	Ambig.	Neutralized features
6	8.95	67.129	1 2 11 13 16 17	9.81	76.959	16 17 11 14 12 3
7	8.52	107.972	1 2 11 13 14 16 17	9.55	78.925	16 17 11 14 12 3 2
8	7.36	151.844	1 2 11 12 13 14 16 17	9.22	92.238	16 17 11 14 12 3 2 1
9	6.62	184.199	1 2 11 12 13 14 15 16 17	6.78	167.370	16 17 11 14 12 3 2 1 13

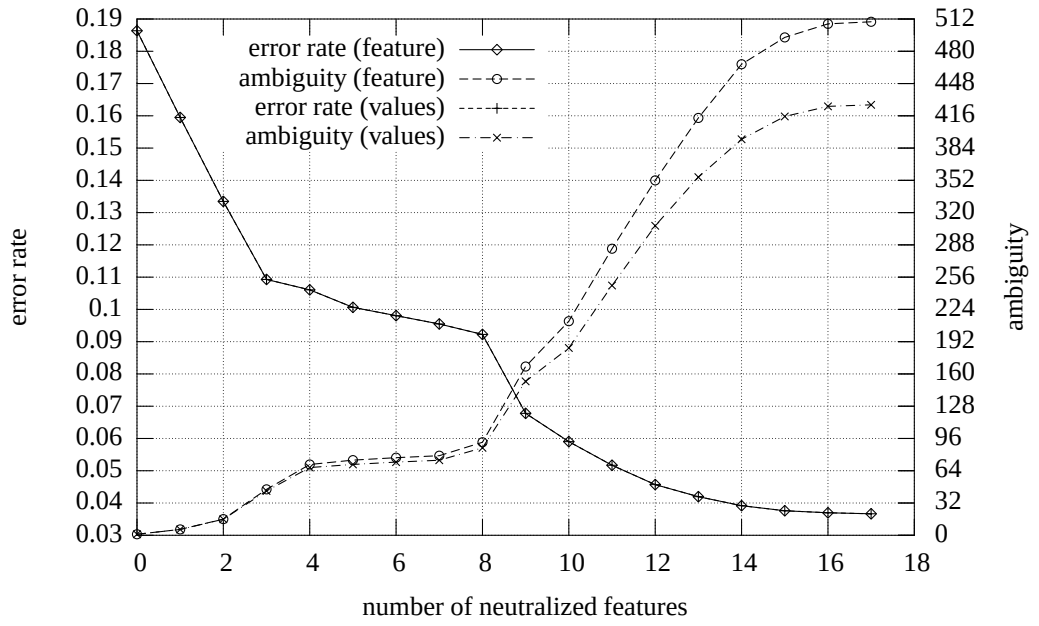


Figure 3: Linked progression of error rate and ambiguity, using only the incremental method, to compare whole *feature* neutralization and selected *values* neutralization. The error rate is the same in both cases. Here the selection of values is driven by error analysis on the same test corpus (to show the theoretical maximum gain we could get with this precise method).

4 Conclusion

The incremental method, while not being perfect, can offer a good approximation at a low cost.

Having applied various Hidden Markov-derived models on supertagging with large extracted grammars, I believe that with such a large tagset it is impossible to achieve a precision rate acceptable for parsing in a single process. Consequently, underspecification imposes itself as one of the most promising directions in this respect. Hopes for future work on this subject mainly lie in a grammar less dependent on the treebank from which it is extracted, in a feature structure better structured (using Metarules (Xia, 2001) or inspired by (Kinyon, 2000) which rely on a Metagrammar (Candito, 1999)), and more importantly in a shallow parsing phase eliminating supertags which would not fit in, thanks to a global consideration of the sentence.

In this last respect, it must be noted that many supertagged sequences are inconsistent: I have observed that a third of them contained at least a supertag which required a certain category before or after it that was not in the relevant part (either to the left or to the right) of the sequence. It is clear that a global vision of the sentence can help reduce the ambiguity of the supertags. The difficulty is to keep the computation simple and fast enough to be used efficiently before full parsing.

Acknowledgments

I wish to particularly thank Owen Rambow, Alexandra Kinyon and Laura Kallmeyer for their precious help with the abstract and the final version of this paper.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Srinivas Bangalore. 1997. *Complexity of lexical descriptions and its relevance for partial parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Marie-Hélène Candito. 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées*. Ph.D. thesis, University Paris 7.

John Chen, Srinivas Bangalore, and K. Vijay-Shanker. 1999. New models for improving supertag disambiguation. In *Proceedings of the Ninth Conference of the European Chapter of the Assocation for Computational Linguistics*, Bergen, Norway.

John Chen, Srinivas Bangalore, Michael Collins, and Owen Rambow. 2002. Reranking an n-gram supertagger. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Venice, Italy.

John Chen. 2001. *Towards Efficient Statistical Parsing using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

Julia Hockenmaier and Mark Steedman. 2002. Acquiring Compact Lexicalized Grammars from a Cleaner Treebank. *Proceedings of Third International Conference on Language Resources and Evaluation (LREC)*.

Aravind K. Joshi and Srinivas Bangalore. August 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *International Conference on Computational Linguistics (COLING 94)*, Kyoto University, Japan.

Alexandra Kinyon. 2000. Hypertags. In *Proceedings of COLING-00*, Saarbrücken, Germany.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, June.

Alexis Nasr, Owen Rambow, John Chen, and Srinivas Bangalore. 2002. Context-Free Parsing of a Tree Adjoining Grammar Using Finite State Machines. In *Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Venice, Italy.

Fei Xia. 2001. *Automatic grammar generation from two different perspectives*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Generalizing Subcategorization Frames Acquired from Corpora Using Lexicalized Grammars

Naoki Yoshinaga[†]
† University of Tokyo
7-3-1 Hongo, Bunkyo-ku,
Tokyo, 113-0033 Japan
yoshinag@is.s.u-tokyo.ac.jp

Jun'ichi Tsujii[†‡]
‡ CREST, JST
4-1-8, Honcho, Kawaguchi-shi,
Saitama, 332-0012 Japan
tsujii@is.s.u-tokyo.ac.jp

Abstract

This paper presents a method of improving the quality of subcategorization frames (SCFs) acquired from corpora in order to augment a lexicon of a lexicalized grammar. We first estimate a confidence value that a word can have each SCF, and create an SCF confidence-value vector for each word. Since the SCF confidence vectors obtained from the lexicon of the target grammar involve co-occurrence tendency among SCFs for words, we can improve the quality of the acquired SCFs by clustering vectors obtained from the acquired SCF lexicon and the lexicon of the target grammar. We apply our method to SCFs acquired from corpora by using a subset of the SCF lexicon of the XTAG English grammar. A comparison between the resulting SCF lexicon and the rest of the lexicon of the XTAG English grammar reveals that we can achieve higher precision and recall compared to naive frequency cut-off.

1 Introduction

Recently, a variety of methods have been proposed for automatic acquisition of subcategorization frames (SCFs) from corpora (Brent, 1993; Manning, 1993; Briscoe and Carroll, 1997; Sarkar and Zeman, 2000; Korhonen, 2002). Although these research efforts aimed at enhancing lexicon resources, there has been little work on evaluating the impact of acquired SCFs on grammar coverage using large-scale lexicalized grammars with the exception of (Carroll and Fang, 2004).

The problem when we combine acquired SCFs with existing lexicalized grammars is lower quality of the acquired SCFs, since they are acquired in an unsupervised manner, rather than being manually coded. If we attempt to compensate for the lack of recall by being less strict in filtering out less likely SCFs, then we will end up with a larger number of lexical entries. This is fatal for parsing with lexicalized grammars, because empirical parsing efficiency and syntactic ambiguity of lexicalized grammars are known to be proportional to the number of lexical entries used in parsing (Sarkar et al., 2000). We therefore need some method to improve the quality of the acquired SCFs.

Schulte im Walde and Brew (2002) and Korhonen (2003) employed clustering of verb SCF (probability) distributions to induce verb semantic classes. Their studies are based on the assumption that verb SCF distributions are closely related to verb semantic classes. Conversely, if we could induce word classes whose element words have the same set of SCFs, we can eliminate SCFs acquired in error from the corpora and predict plausible SCFs unseen in the corpora. This kind of generalization would be useful to improve the quality of the acquired SCFs.

In this paper, we present a method of generalizing SCFs acquired from corpora in order to augment a lexicon of a lexicalized grammar. For words in the acquired SCF lexicon and the lexicon of the target lexicalized grammar, we first estimate a confidence value that a word can have each SCF. We next perform clustering of SCF confidence-value vectors in order to make use of co-occurrence tendency among SCFs for words in the lexicon of the target lexicalized grammar. Since each centroid value of the obtained clusters indicate whether the words in that class have each SCF, we eliminate implausible SCFs and add unobserved but possible SCFs according to that value. In other words, we can generalize the acquired SCFs by the reliable lexicon of the target lexicalized grammar.

We applied our method to SCFs acquired from mobile phone news groups corpus by a method described in (Carroll and Fang, 2004), in order to generalize the acquired SCFs by using a training portion of the SCF lexicon of the XTAG English grammar (XTAG Research Group, 2001), a large-scale Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988). We evaluated the resulting SCF lexicon by comparing it to the rest of

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 104-110.

```
(#S(EPATTERN :TARGET |ftp|
             :SUBCAT (VSUBCAT NONE)
             :CLASSES (22 2985)
             :RELIABILITY 0
             :FREQSCORE 0.01640195
             :FREQCNT 2
             :TLTL (VVD VV0)
             :SLTL ((((|ssh| NN1)))
             :OLT1L NIL
             :OLT2L NIL
             :OLT3L NIL :LRL 0))
```

Figure 1: An acquired SCF for a verb "ftp"

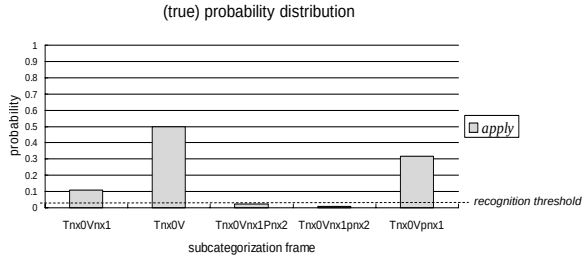


Figure 2: Probability distributions of SCFs for *apply*

the lexicon of the XTAG English grammar, and then compared the results with those obtained by naive frequency cut-off.

2 Background

2.1 Acquisition of SCFs for Lexicalized Grammars

We start by acquiring SCFs for a lexicalized grammar from corpora by the method described in (Carroll and Fang, 2004).

In their study, they first acquire fine-grained SCFs by the method proposed by (Briscoe and Carroll, 1997; Korhonen, 2002). Figure 1 shows an example of one acquired SCF entry for a verb "*ftp*." Each acquired SCF entry has several fields about the observed SCF. We explain here only its portion related to this study. The TARGET field is a word stem (|ftp| in Figure 1), the first number in the CLASSES field indicates an SCF ID (22 in Figure 1), and FREQCNT shows how often words derivable from the word stem had the SCF identified by the SCF ID (2 times in Figure 1) in the training corpus. The obtained SCFs comprise the total 163 types of relatively fine-grained SCFs, which are originally based on the SCFs in the ANLT (Boguraev and Briscoe, 1987) and COMLEX (Grishman et al., 1994) dictionaries. In this example, the SCF ID 22 corresponds to an SCF of intransitive verb.

They then obtain SCFs for the target lexicalized grammar (the LINGO English Resource Grammar (Flickinger, 2000) in their study) by using a handcrafted translation map from these 163 types to one of the types of SCFs in the target grammar. They report that they could achieve a coverage improvement of 4.5% (52.7% to 57.2%) with a parsing time double (9.78 sec. to 21.78 sec.).

This approach is easily extensible to any lexicalized grammars, if the grammars have an organized architecture of lexicon, which derive possible lexical entries from each SCF the grammar defines. Existing lexicalized grammars usually are equipped with this kind of organization, *e.g.*, lexical types in LINGO ERG and tree families in the XTAG English grammar.

2.2 Clustering of Verb SCF Distributions

There are some related work on clustering of SCF probability distributions (Schulte im Walde and Brew, 2002; Korhonen et al., 2003). These studies aim at obtaining verb semantic classes, which closely related to syntactic behavior of argument selection.

Schulte im Walde and Brew (2002) employed clustering of verb SCF distributions to induce verb semantic classes. They first represent a verb SCF distribution by an n-dimensional vector for each verb. Each element in the SCF distribution represents a probability that a verb appears with the corresponding SCF. They then perform k-Means clustering (Forgy, 1965) of these vectors in order to obtain verb semantic classes.

Korhonen et al. (2003) also conducted clustering of verb SCF distributions using a different clustering method including the nearest neighbors clustering and the Information Bottleneck clustering (Tishby et al., 1999). They investigated the effect of polysemic verbs on clustering.

Although these studies demonstrated that there is a certain classification of verbs by clustering of verb SCF distributions, they do not focus on the improvement of the quality of the SCF lexicon. In this paper, we focus on the problem to identify whether a word can have each SCF and try to obtain word classes whose element words have the same set of SCFs.

3 Method

The basic idea of our method is first to obtain word classes whose element words have the same set of SCFs, using not only acquired SCFs but also existing SCFs in the target grammar. We then eliminate implausible acquired SCFs and add plausible unseen SCFs according to the set of SCFs represented by the centroids of the resulting clusters.

3.1 Representation of Confidence Values for SCFs

We represent *an SCF confidence-value vector* of each word w_i with a vector v_i, an object for clustering. Each element v_{ij} in v_i represents the confidence value of SCF

s_j for w_i, which expresses how reliable a word w_i has SCF s_j. We should note that the confidence value is not the probability that a word w_i appears with SCF s_j but a probability of existence of SCF s_j for the word w_i. In this study, we assume that a word w_i can have each SCF s_j with a certain (non-zero) probability $\theta_{ij}(= p(s_{ij}|w_i) > 0$ where $\sum_j \theta_{ij} = 1)$, but only SCFs whose probabilities exceed a certain threshold are recognized as SCFs for the word in the lexicon. We hereafter call this threshold *recognition threshold*. Figure 2 exemplifies a probability distribution of SCFs for *apply*. In this context, we can regard a confidence value of each SCF as the possibility that a probability of a SCF exceeds the recognition threshold.

One intuitive way to estimate a confidence value is to assume an observed probability, *i.e.*, relative frequency, is equal to a probability θ_{ij} of SCF s_j for a word w_i ($\theta_{ij} = freq_{ij}/\sum_j freq_{ij}$ where $freq_{ij}$ is a frequency count that a word w_i have the SCF s_j in corpora[1]). We simply assign 1 to a confidence value $conf_{ij}$ when the relative frequency of s_j for a word w_i exceeds the recognition threshold, and otherwise assign 0 to a confidence value of $conf_{ij}$. However, an observed probability is totally unreliable for infrequent words. For example, when we use a confidence value derived from a relative frequency as above, we cannot distinguish cases where a word w_1 appears once with a SCF s_j and a word w_2 appears 100 times, always with the SCF s_j, which are both the relative frequency 1. Moreover, even when we would like to encode confidence values of reliable SCFs in the target lexicalized grammar, it is also problematic to distinguish the confidence value of those SCFs with confidence values of acquired SCFs.

The other promising way to estimate a true probability θ_{ij} is to regard it as a stochastic variable in the context of Bayesian statistics (Gelman et al., 1995). In this context, *a posteriori* distribution of the probability θ_{ij} of a SCF s_j for a word w_i is given by:

$$p(\theta_{ij}|D) = \frac{P(\theta_{ij})P(D|\theta_{ij})}{P(D)} = \frac{P(\theta_{ij})P(D|\theta_{ij})}{\int_0^1 P(\theta_{ij})P(D|\theta_{ij})d\theta_{ij}}, \quad (1)$$

where $P(\theta_{ij})$ is *a priori* distribution, and D is the data we have observed. Since every occurrence of SCFs in the data D is independent with each other, the data D can be regarded as Bernoulli trials in this case. When we observe the data D that a word w_i appears n times and has SCF s_j $x(\leq n)$ times, its conditional distribution is therefore represented by binominal distribution:

$$P(D|\theta_{ij}) = \binom{n}{x} \theta_{ij}^x (1-\theta_{ij})^{(n-x)}. \quad (2)$$

To calculate this *a posteriori* distribution, we need to define the *a priori* distribution $P(\theta_{ij})$. The question is which probability distribution of θ_{ij} can appropriately reflect prior knowledge. In other words, it should encode knowledge we use to estimate SCFs for an unknown word w_i. We simply determine it from distributions of probability values of s_j for known words. We use distributions of observed probability values of s_j for all words acquired from the corpus by using a method described in (Tsuruoka and Chikayama, 2001). In their study, they assume *a priori* distribution as the *beta* distribution defined as:

$$p(\theta_{ij}|\alpha,\beta) = \frac{\theta_{ij}^{\alpha-1}(1-\theta_{ij})^{\beta-1}}{B(\alpha,\beta)}, \quad (3)$$

where $B(\alpha,\beta) = \int_0^1 \theta_{ij}^{\alpha-1}(1-\theta_{ij})^{\beta-1}d\theta_{ij}$. The value of α and β is determined by moment estimation.[2] By substituting Equations 2 and 3 into Equation 1, we finally obtain the *a posteriori* distribution $p(\theta_{ij}|D)$ as:

$$p(\theta_{ij}|\alpha,\beta,D) = \frac{\frac{\theta_{ij}^{\alpha-1}(1-\theta_{ij})^{\beta-1}}{B(\alpha,\beta)}\binom{n}{x}\theta_{ij}^x(1-\theta_{ij})^{(n-x)}}{\int_0^1 P(\theta_{ij})P(D|\theta_{ij})d\theta_{ij}} = c\cdot\theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1} \quad (4)$$

where $c = \binom{n}{x}/(B(\alpha,\beta)\int_0^1 P(\theta_{ij})P(D|\theta_{ij})d\theta_{ij})$.

When we determine the value of the recognition threshold as t, we can calculate a confidence value $conf_{ij}$ that a word w_i can have s_j by integrating the *a posteriori* distribution $p(\theta_{ij}|D)$ from the threshold t to 1:

$$conf_{ij} = \int_t^1 c\cdot\theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij} \quad (5)$$

By using this confidence value, we can express an SCF confidence-value vector v_i for a word w_i in the acquired SCF lexicon ($v_{ij} = conf_{ij}$).[3]

In order to combine SCF confidence-value vectors for words acquired from corpora and those for words in the

[1] We used values of `FREQCNT` to obtain frequency counts of SCFs.

[2] The expectation value and variance of the beta distribution are made equal to those of the observed probability values.

[3] By using the fact that $\int_0^1 P(\theta_{ij}|\alpha,\beta) = 1$, we can calculate $conf_{ij}$ as follows.

$$conf_{ij} = \frac{\int_t^1 c\cdot\theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}}{\int_0^1 c\cdot\theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}} = \frac{\int_t^1 \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}}{\int_0^1 \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}} \quad (6)$$

```
Input:  a set of SCF confidence-value
          vectors V = {v_1, v_2, ..., v_n} ⊆ R^m
        a distance function d : R^m × Z^m → R
        a function to compute a centroid
          μ : {v_j1, v_j2, ..., v_jl} → R^m
Output: a set of clusters C_j

while cluster members are not stable do
  foreach cluster C_j
    C_j = {v_i | ∀c_l, d(v_i, c_j) ≤ d(v_i, c_l)}
  end foreach
  foreach clusters C_j
    c_j = μ(C_j)
  end foreach
end while

return C_j
```

Figure 3: Clustering algorithm for SCF confidence-value distributions

lexicon of the target grammar, we also represent SCF confidence-value vectors for the words in the target grammars. In this paper, we express SCF confidence-value vectors v'_i for words in the SCF lexicon of the target grammar by:

$$v'_{ij} = \begin{cases} 1-\varepsilon & w_i \textit{ has } s_j \textit{ in the lexicon} \\ \varepsilon & \textit{otherwise} \end{cases} \quad (7)$$

where ε expresses an unreliability of the lexicon. In this study, we simply set it to the machine epsilon. In other words, we trust the lexicon as much as possible.

3.2 Clustering Algorithm for SCF Confidence-Value Distributions

We next present a k-Means-like clustering algorithm for SCF confidence-value vectors, as shown in Figure 3. Given an initial assignment of data objects to k clusters, our algorithm computes a representative value of each cluster called *centroids*. Our algorithm then iteratively updates clusters by assigning each object to its closest centroid and recomputing centroids until cluster members become stable.

Although our algorithm is roughly based on the k-Means algorithm, it is different in an important respect. We define the elements of the centroid values of the obtained clusters as a discrete value of 0 or 1 because we want to obtain clusters which include words that have the exactly same set of SCFs. We then derive a distance function d to calculate the distance from a data object v_i to each centroid c_m. Since the distance function is used to determine the closest cluster for v_i, we define the function d to output the probability that v_i has the SCF set expressed by centroid c_m as follows:

$$d(v_i, c_m) = \prod_{c_{mj}=1} v_{ij} \cdot \prod_{c_{mj}=0} (1-v_{ij}). \quad (8)$$

By using this function, we can determine the closest cluster as $\underset{C_m}{\operatorname{argmax}}\, d(v_i, c_m)$.

After every assignment, we determine a next centroid c_m of each cluster C_m as follows:

$$c_{mj} = \begin{cases} 1 & \textit{when } \prod_{v_i \in C_m} v_{ij} > \prod_{v_i \in C_m} (1-v_{ij}) \\ 0 & \textit{otherwise} \end{cases} \quad (9)$$

We then address the way to determine the number of clusters and initial assignments of objects. In this paper, we assume that the most of the possible set of SCFs for words are included in the target lexicalized grammar, and make use of the existing sets of SCFs for the words in the lexicon of the target grammar to determine the possible set of SCFs for words out of the lexicon. We first extract SCF confidence-value vectors from the lexicon of the target grammar by regarding $\varepsilon = 0$ in Equation 7. By eliminating duplications from them, we obtain SCF centroid-value vectors c_m. We then initialize the number of clusters k to the number of c_m and use them as initial centroids.[4]

We finally update the acquired SCFs using each element's value in the centroid of each cluster and the confidence value of SCFs in this order. We first eliminate SCF s_j for w_i in a cluster m when the value c_{mj} of the centroid c_m is 0, and add SCF s_j for w_i in a cluster m when the value c_{mj} of the centroid c_m is 1. This is because c_{mj} represents whether the words in that class can have SCF s_j. We then eliminate implausible SCFs s_j for w_i from the resulting SCFs according to its corresponding confidence value $conf_{ij}$. We call this elimination *centroid cut-off*. In the following experiments, we compare this cut-off with naive *frequency cut-off*, which uses only relative frequencies to eliminate SCFs and *confidence cut-off*, which uses only confidence values to eliminate SCFs. Note that frequency cut-off and confidence cut-off use only corpus-based statistics to eliminate SCFs.

4 Experiments

We applied our method to an SCF lexicon acquired from 135,902 sentences of the mobile phone news group archived by Google.com, which is the same data used in (Carroll and Fang, 2004). The number of the resulting SCFs is 14,783 for 3,864 word stems. We then translated them to an SCF lexicon for the XTAG English grammar (XTAG Research Group, 2001) by using a translation map manually defined by Ted Briscoe. It defines a mapping from 23 out of 163 possible SCF types into 13 out of 57 XTAG SCFs called *tree families* listed in Table 1. The number of resulting SCFs for the XTAG English grammar was 6,742 for 2,860 word stems.

[4] When a lexicon of the grammar is not comprehensive or less accurate, we should determine the number of clusters using other algorithms (Bischof et al., 1999; Hamerly, 2003).

Table 1: Tree families of the XTAG English grammar mapped from 23 out of 163 SCF types

Tree family	Explanation
Tnx0Ax1	Adjective small clause
Tnx0Vnx1	Transitive
Tnx0Vs1	Sentential complement
Tnx0Vnx2nx1	Ditransitive
Tnx0Vnx1Pnx2	Multiple anchor ditransitive with PP
Tnx0Vnx1pnx2	Ditransitive with PP
Tnx0Vplnx1	Transitive verb Particle
Tnx0Vpl	Intransitive verb Particle
Tnx0Vnx1s2	Sentential complement with NP
Tnx0Vpnx1	Intransitive with PP
Ts0Vnx1	Transitive sentential subject
Tnx0Vax1	Intransitive with adjective
Tnx0Vplnx2nx1	Ditransitive verb Particle

In order to evaluate our method, we split the SCF lexicon of the XTAG English grammar into the training portion and the test portion. The training portion includes 9,427 SCFs for 8,399 words, while the test portion includes 433 SCFs for 280 words The test portion is selected from the SCF lexicon for words that are observed in the acquired SCF lexicon. We extract SCF confidence-value vectors from the training portion and combine them with the SCF confidence-value vectors obtained from the acquired SCFs. The number of the resulting data objects is 8,679.[5] We also make use of the SCF confidence-value vectors obtained from the training SCF lexicon as an initial centroid by regarding ε as 0. The total number of them was 35.[6] We then performed clustering of these 8,679 data objects into 35 clusters.

We finally evaluate precision and recall of the resulting SCFs by comparing them with the test SCF lexicon of the XTAG English grammar.

We first compare confidence cut-off with frequency cut-off to investigate effects of Bayesian estimation. Figure 4 shows precision and recall of the resulting SCF sets using confidence cut-off and frequency cut-off. We measured precision and recall of the SCF sets obtained using confidence cut-off whose recognition threshold $t = 0.01$ (confidence cut-off 0.01), 0.03 (confidence cut-off 0.03), and 0.05 (confidence cut-off 0.05) by varying threshold for the confidence value from 0 to 1. We also measured those for the SCF sets obtained using frequency cut-off by varying threshold for the relative frequency from 0 to 1. The graph apparently indicates that the confidence cut-offs outperformed the frequency cut-off. When we compare confidence cut-offs with different recognition thresholds, we can improve precision using higher recognition threshold while we can improve recall using lower recognition threshold. This result is quite consistent with our expectations.

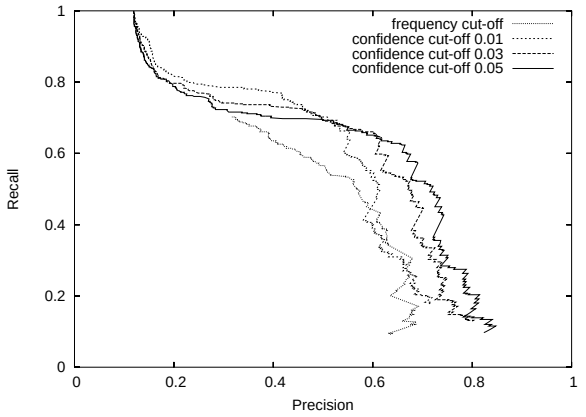


Figure 4: Precision and recall of the resulting SCFs using confidence cut-off and frequency cut-off

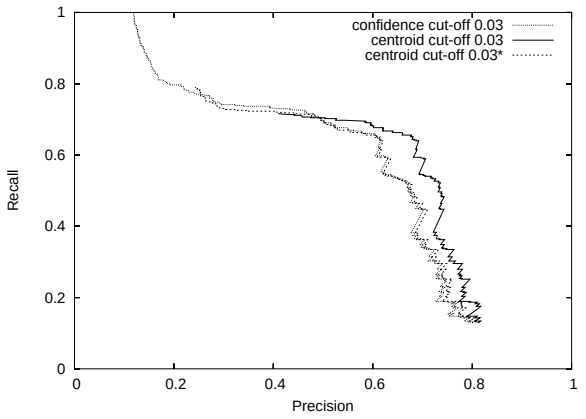


Figure 5: Precision and recall of the resulting SCFs using confidence cut-off and frequency cut-off

We then compare centroid cut-off with confidence cut-off to observe effects of clustering using information in the lexicon of the XTAG English grammar. Figure 5 shows precision and recall of the resulting SCF sets using centroid cut-off and confidence cut-off with the recognition threshold $t = 0.03$ by varying the threshold for the confidence value. In order to show the effects of information of the training SCF lexicon, centroid cut-off 0.03* is SCFs obtained by clustering of SCF confidence-value vectors in the acquired SCFs only with random initialization. The graph apparently shows that clustering is meaningful only when we make use of the reliable SCF confidence-value vectors obtained from the manually tai-

[5] We used the SCF confidence-value vectors for words which are included in the XTAG English grammar. When both the training SCF lexicon and the acquired SCF lexicon have the same words, we simply used an SCF confidence-value vector obtained from the acquired SCF lexicon.

[6] We used the SCF confidence-value vectors that appear with more than two words.

SCF	# SCFs	frequency cut-off				confidence cut-off 0.03				centroid cut-off 0.03			
		Precision		Recall		Precision		Recall		Precision		Recall	
Tnx0Ax1	12(1)	na	(0/0)	0.000	(0/12)	na	(0/0)	0.000	(0/12)	na	(0/0)	0.000	(0/12)
Tnx0Vnx1	267(222)	0.959	(212/221)	0.794	(212/267)	0.958	(253/264)	0.948	(253/267)	0.956	(260/272)	0.974	(260/267)
Tnx0Vs1	38(29)	0.357	(10/28)	0.263	(10/38)	0.381	(8/21)	0.211	(8/38)	0.323	(10/31)	0.263	(10/38)
Tnx0Vnx2nx1	21(16)	0.105	(6/57)	0.286	(6/21)	0.185	(10/54)	0.476	(10/21)	0.122	(9/74)	0.429	(9/21)
Tnx0Vnx1Pnx2	8(4)	0.200	(3/15)	0.375	(3/8)	0.200	(2/10)	0.250	(2/8)	0.250	(2/8)	0.250	(2/8)
Tnx0Vnx1pnx2	5(1)	0.024	(1/41)	0.200	(1/5)	0.029	(1/34)	0.200	(1/5)	na	(0/0)	0.000	(0/5)
Tnx0Vplnx1	40(23)	0.538	(7/13)	0.175	(7/40)	0.667	(6/9)	0.150	(6/40)	0.778	(7/9)	0.175	(7/40)
Tnx0Vpl	20(0)	na	(0/0)	0.000	(0/20)	na	(0/0)	0.000	(0/20)	na	(0/0)	0.000	(0/20)
Tnx0Vnx1s2	11(6)	0.083	(1/12)	0.091	(1/11)	0.200	(1/5)	0.091	(1/11)	0.200	(1/5)	0.091	(1/11)
Ts0Vnx1	8(1)	0.000	(0/2)	0.000	(0/8)	na	(0/0)	0.000	(0/8)	na	(0/0)	0.000	(0/8)
Tnx0Vax1	2(1)	0.000	(0/9)	0.000	(0/2)	0.000	(0/3)	0.000	(0/2)	0.000	(0/1)	0.000	(0/2)
Tnx0Vplnx2nx1	1(0)	0.000	(0/2)	0.000	(0/1)	na	(0/0)	0.000	(0/1)	na	(0/0)	0.000	(0/1)

Table 2: Precision and recall for 400 SCFs obtained from freqency cut-off, confidence cut-off 0.03, and centroid cut-off 0.03

lored lexicon. The centroid cut-off using the lexicon boosted precision and recall compared to the confidence cut-off and the centroid cut-off without the lexicon.

We finally investigate precision and recall of the resulting SCFs for every SCF type in order to evaluate effects of our method on each SCF. Table 2 shows precision and recall of the SCFs by using frequency cut-off (the threshold for the relative frequency 0.092), confidence cut-off 0.03 (the threshold for the confidence value 0.953), centroid cut-off 0.03 (the threshold for the confidence value 0.889)[7] by using thresholds for the relative frequency and the confidence value that preserve exactly 400 SCFs. The numbers in curly brackets in # of SCFs colum show the number of SCFs in the test SCF lexicon that are acquired from the training corpus. The left and right numbers in curly brackets in the precision columns show the number of correct SCFs against all SCFs in the resulting SCF lexicon while those in the recall columns show the number of correct SCFs against all SCFs in the test SCF lexicon. We can observe a tendency that the confidence cut-off and the centroid cut-off preserve more transitive (Tnx0Vnx1) SCF. This is because some SCFs of Tnx0Vnx1 in the test SCF lexicon are not observed in the training corpus but are predicted by *a priori* distribution for SCF Tnx0Vnx1. Also, the centroid cut-off tends to reduce implausible SCFs of Tnx0Vnx1Pnx2 and Tnx0Vax1. Since the threshold for the confidence value of the centroid cut-off 0.03 (0.889) is smaller than that of the confidence cut-off 0.03 (0.953), the clustering could eliminate implausible SCFs without reducing recall.

In short, one reason why the centroid cut-off outperforms the confidence cut-off (or the frequency cut-off) is due to the way how the centroid cut-off eliminate SCFs not existed in the lexicon. When we eliminate SCFs with lower relative frequency under the assumption that those SCFs tend to be wrongly acquired SCFs, it must also eliminate correct SCFs with low relative frequencies. By using co-occurrence tendency among SCFs as another criteria to judge the implausibility of the SCFs, we can eliminate more wrongly acquired SCFs because they tend to violate the co-occurrence tendency. Another reason why the centroid cut-off and the confidence cut-off outperform the the frequency cut-off is due to the way how those cut-offs add new unseen SCFs. We can add plausible SCFs from those SCFs which is reliable according to their *a priori* distribution. Furthermore, since the centroid cut-off makes use of the co-occurrence tendency among SCFs, it adds only SCFs which are plausible in terms of corpus-based statistics (confidence value) under the restriction provided by the co-occurrence tendency among SCFs in the lexicon of the target grammar.

[7] Since no word takes SCF Tnx0Vpnx1 in the test SCF lexicon, we omit it here.

5 Concluding Remarks and Future Work

In this paper, we presented a novel way to improve the quality of SCFs acquired from corpora in order to augment a lexicalized grammar with them. By applying our method to the acquired SCF lexicon using the XTAG English grammar, we showed that our method improved both precision and recall of the resulting SCFs compared to the naive frequency-based cut-off.

In future work, we are going to investigate the parsing performance of the XTAG English grammar augmented with SCFs obtained by our method. We will apply our method to lexicalized grammars with relatively smaller lexicon, *e.g.*, the LINGO English Resource Grammar (Flickinger, 2000).

Acknowledgment

The authors wish to thank Yoshimasa Tsuruoka and Takuya Matsuzaki for their advice on probabilistic modeling of the set of SCFs, and thank Alex Fang for his help in using SCFs acquired from the corpus. The authors are also indebted to Yusuke Miyao, John Carroll and the three anonymous reviewers for their valuable comments on this paper. The first author was supported in part by JSPS Research Fellowships for Young Scientists.

References

Horst Bischof, Ales Leonardis, and Alexander Selb. 1999. MDL principle for robust vector quantization. *Pattern Analysis and Applications*, 2(1):59–72.

Branimir Boguraev and Ted Briscoe. 1987. Large lexicons for natural language processing: utilising the grammar coding system of LDOCE. *Computational Linguistics*, 13(4):203–218.

Michael R. Brent. 1993. From grammar to lexicon. *Computational Linguistics*, 19(2):243–262.

Ted Briscoe and Jhon Carroll. 1997. Automatic extraction of subcategorization from corpora. In *Proc. of the fifth ANLP*, pages 356–363.

Jhon Carroll and Alex C. Fang. 2004. The automatic acquisition of verb subcategorizations and their impact on the performance of an HPSG parser. In *Proc. of the first ijc-NLP*, pages 107–114.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28.

Edward W. Forgy. 1965. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. *Biometrics*, 21:768–780.

Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin, editors. 1995. *Bayesian Data Analysis*. Chapman and Hall.

Ralph Grishman, Catherine Macleod, and Adam Meyers. 1994. Comlex syntax: Building a computational lexicon. In *Proc. of the 15th COLING*, pages 268–272.

Greg Hamerly. 2003. *Learning structure and concepts in data through data clustering*. Ph.D. thesis, University of Californila, San Diego.

Anna Korhonen, Yuval Krymolowski, and Zvika Marx. 2003. Clustering polysemic subcategorization frame distributions semantically. In *Proc. of the 41st ACL*, pages 64–71.

Anna Korhonen. 2002. *Subcategorization Acquisition*. Ph.D. thesis, University of Cambridge.

Christopher D. Manning. 1993. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proc. of the 31st ACL*, pages 235–242.

Anoop Sarkar and Daniel Zeman. 2000. Automatic extraction of subcategorization frames for Czech. In *Proc. of 18th COLING*, pages 691–697.

Anoop Sarkar, Fei Xia, and Aravind K. Joshi. 2000. Some experiments on indicators of parsing complexity for lexicalized grammars. In *Proc. of the 18th COLING workshop*, pages 37–42.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In *Proc. of the 12th COLING*, pages 578–583.

Sabine Schulte im Walde and Chris Brew. 2002. Inducing German semantic verb classes from purely syntactic subcategorisation information. In *Proc. of the 41st ACL*, pages 223–230.

Naftali Tishby, Fernand C. Pereira, and William Bialek. 1999. The information bottleneck method. In *Proc. of the 37th ACL*, pages 368–377.

Yoshimasa Tsuruoka and Takashi Chikayama. 2001. Estimating reliability of contextual evidences in decision-list classifiers under bayesian learning. In *Proc. of the sixth NLPRS*, pages 701–707.

XTAG Research Group. 2001. A lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

LTAG Semantics of NP-Coordination*

Olga Babko-Malaya
Department of Computer and Information Science, University of Pennsylvania
3330 Walnut Street, Philadelphia, PA 19104-6389
malayao@linc.cis.upenn.edu

Abstract

A central component of Kallmeyer and Joshi 2003 is the idea that the contribution of a quantifier is separated into a scope and a predicate argument part. Quantified NPs are analyzed as multi-component TAGs, where the scope part of the quantifier introduces the proposition containing the quantifier, and the predicate-argument part introduces the restrictive clause. This paper shows that this assumption presents difficulties for the compositional interpretation of NP coordination structures, and proposes an analysis which is based on LTAG semantics with semantic unification, developed in Kallmeyer and Romero 2004.

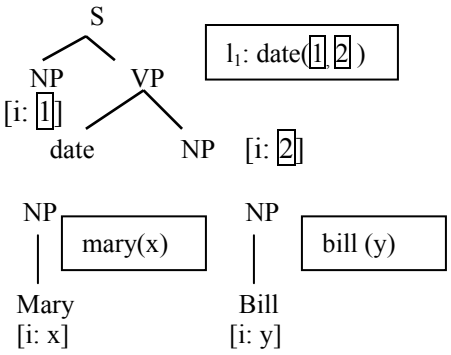
1 LTAG Semantics with Semantic Unification.

In LTAG framework (Joshi and Schabes 1997), the basic units are (elementary) trees, which can be combined into bigger trees by substitution or adjunction. LTAG derivations are represented by derivation trees that record the history of how the elementary trees are put together. Given that derivation steps in LTAG correspond to predicate-argument applications, it is usually assumed that LTAG semantics is based on the derivation tree, rather than the derived tree (Kallmeyer and Joshi 2003).

Semantic composition which we adopt is based on LTAG-semantics with Semantic Unification (Kallmeyer and Romero 2004). In the derivation tree, elementary trees are replaced by their semantic representations and corresponding feature structures. Semantic representations are as defined in Kallmeyer and Joshi 2003, except that they do not have argument variables. These representations consist of a set of formulas (typed λ-expressions with labels) and a set of scope constraints. The scope constraints x ≤ y are as in Kallmeyer and Joshi 2003, except that both x and y are propositional labels or propositional variables.

Each semantic representation is linked to a feature structure. Feature structures, as illustrated by different examples below, include a feature i, whose values are individual variables, and features p and MaxS, whose values are propositional labels. Semantic composition consists of feature unification. After having performed all unifications, the union of all semantic representations is built. Consider, for example, semantic representations and feature structures associated with the elementary trees of the sentence shown in (1).

(1) Mary dates Bill

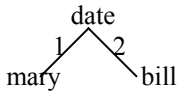


The derivation tree that records the history of how elementary trees are put together is shown in (2):

* I would like to thank Maria-Isabel Romero, Aravind Joshi, Laura Kallmeyer and all participants of the XTAG meetings for discussions and numerous suggestions, as well as anonymous reviewers for their valuable comments. All remaining errors are mine

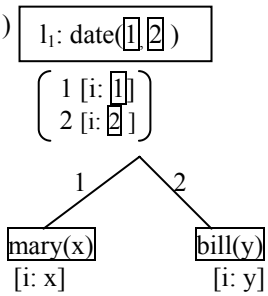
TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 111-117.

(2)



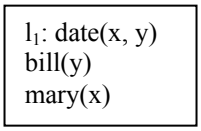
Semantic composition proceeds on the derivation tree and consists of feature unification[2]:

(3)



Performing two unifications, 1=x, 2=y, we arrive at the final interpretation of this sentence:

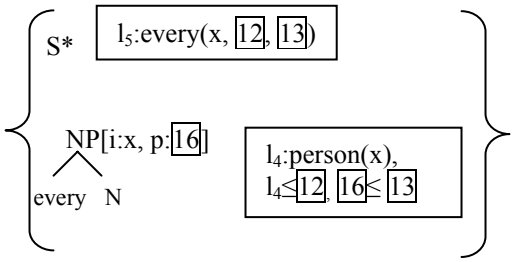
(4)



This representation is interpreted conjunctively, with free variables being existentially bound.

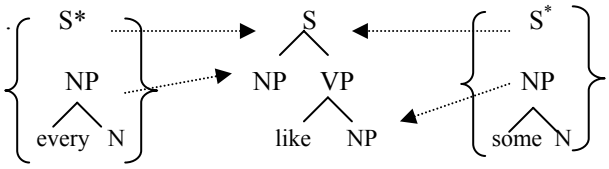
Quantificational NPs are analyzed as multi-component TAGs, where the scope part of the quantifier introduces the proposition containing the quantifier, and the predicate-argument part introduces the restrictive clause. The multi-component representation of the quantifier 'everybody', for example, and its semantics, is shown in (5):

(5)

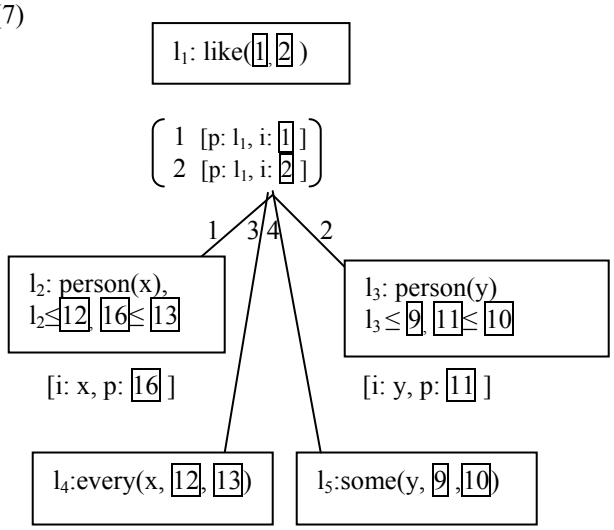


The use of multi-component representations for quantifiers in Kallmeyer and Joshi 2003 is motivated by the desire to generate underspecified representations for scope ambiguities. Consider, for example, compositional interpretation of the sentence in (6), shown in (7).

(6) Everybody likes someone.

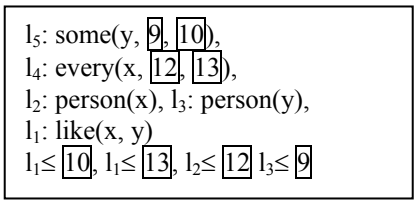


(7)



Performing unifications leads to the feature identities 1=x, 2=y, 11=l_1, 16=l_1 and the following final representation of this sentence:

(8)



The semantic representation in (8) is underspecified for scope, and there are two possible disambiguations of scope constraints (i.e. functions from propositional variables to propositional labels that respect the scope constraints in the sense of Kallmeyer and Joshi 2003), shown in (9a) and (9b).

(9) a. 10 -> l_1, 13 -> l_5
b. 13 -> l_1, 10 -> l_4

In (9a), the proposition l_1 is identified with the nuclear scope of the quantifier 'some', and the proposition l_5 with the nuclear scope of 'every'. The quantifier 'every' has a wide scope interpretation in this case. In (9b), the quantifier 'every' is identified with the nuclear scope of 'some', and thus has a narrow scope interpretation.

[2] For simplification, top-bottom feature distinction is omitted.

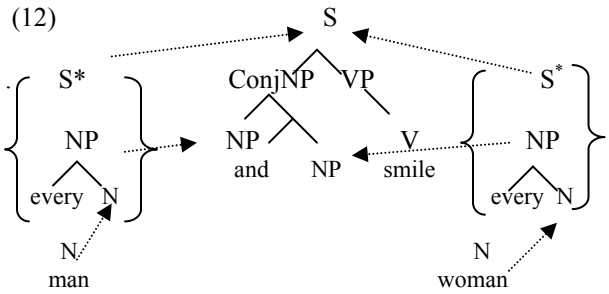
2 Problems for NP-Coordination

Structures with conjoined quantified NPs, of the type illustrated in (10) and (11), present difficulties for this analysis.

(10) Every man and every woman smiled.
(11) Every man and every woman solved a puzzle.

First, separating scope part and predicate-argument part presents a challenge for a compositional interpretation of conjoined structures, since the conjunction 'and' is composed with the NP-parts of the quantified NPs, which specify the restrictive clause (as the derivation tree in (12) illustrates[3]). On the other hand, the desired interpretation of this sentence is 'every man smiled and every woman smiled', where the two quantifiers are conjoined, rather than just their restrictive parts. Furthermore, under the analysis presented above these structures are expected to show scope ambiguities, whereas it is well known that conjoined structures are islands for quantifier scope (Ross 1967, Morrill 1994, among others).

(12)



The second problem concerns the fact that the interpretation of this sentence involves two 'copies' of the proposition introduced by the verb:

(13) Every(x, man(x), smile(x)) ∧
every(y, woman(y), smile(y))

In LTAG semantics, as developed in Kallmeyer and Joshi 2003, the representation of each elementary tree is a proposition. The semantic representation of a tree for 'smile', for example, denotes a proposition smile(1), where 1 is identified with a variable introduced by the NP. In order to derive a compositional interpretation of the sentence in (10), on the other hand, S-tree should denote a property, which can be predicated of either x or y (as has been proposed for the analysis of this type of constructions in Montague-style semantic frameworks, (e.g. Partee and Rooth 1983), as well as Categorial Grammars (e.g. CCG, Steedman 1996)). This option, however, is not directly available in the LTAG semantics, given that the nuclear scope of quantifiers which are adjoined to S should be unified with a proposition supplied by the S-tree.

This problem becomes more apparent when we try to analyze the sentence in (11). This sentence has two possible interpretations:

(14) Every man and every woman solved a puzzle.

a. every(x, man(x), some(z, puzzle(z), solve(x, z)))
∧every(y, woman(y), some(z,puzzle(z),solve(y, z))))
b. some(z, puzzle(z), every(x, man(x), solve(x, z)))
∧every(y, woman(y), solve(y, z))))

In the interpretation in (14a), the nuclear scope of both quantifiers 'every' has to be identified with the quantifier 'some'. However, since quantifiers are introduced as propositions, we cannot identify the same proposition with the nuclear scopes 4 and 6 of both quantifiers every in every(x, man(x), 4) and every(y, woman(y), 6). The proposition 'some' has to be 'copied' at some point of compositional interpretation, so that 4 and 6 will be identified with different copies of 'some'. In the interpretation (14b), on the other hand, what is being 'copied' is the proposition introduced by the verb, i.e. solve(1,2).

Let us consider possible assignments for nuclear scopes of the three quantifiers:

(15) every (x, 3, 4):
a. some(z, puzzle(z), solve(x, z))=4 or
b. solve(x, z) = 4

every (y, 5, 6),
a. some(z, puzzle(z), solve(y, z))=6 or
b solve(y, z) = 6

some(z, 7, 8):
a. solve(v, z) = 8,
where v can be either x or y, or
b. every(x, man(x), solve(x, z) ∧
every(x, man(x), solve(x, z)) = 8

As (15) shows, it does not seem possible to find a single proposition which could be viewed as 'being in the nu-

[3] The tree in (12) represents shorthand for the derivation tree of this sentence. ConjNP is a separate elementary tree, and in order for the derivation to be local, the NP tree should be first composed with the ConjNP, then the derived tree is combined with the second NP-tree, and then the resulting multi-component TAG is combined with the S-tree (as described in flexible composition approach in Joshi et al 2003). The order of syntactic derivation is not relevant for the semantic analysis and therefore is not represented here.

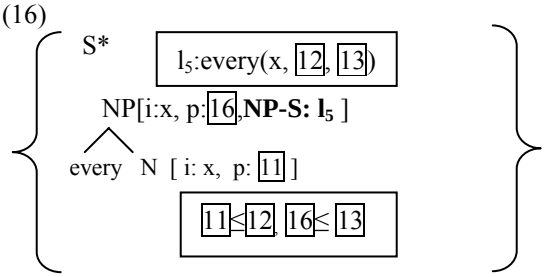
clear scope' of the three quantifiers. Furthermore, in the case of the (a) reading of the quantifier 'some', we need to account for the clause 'where v can be either x or y', which given the present framework implies that we should be able to map the same variable 22 to two different propositions, specifically: solve(x, z) and solve(y, z). This is undesirable, given that disambiguations are viewed as functions from propositional variables to propositional labels.

The question which arises therefore is what kind of underspecified representation and copying mechanism can we use to achieve the desired scope ambiguities?

3 Coordination of Quantified NPs

To solve the first problem, we propose that the NP part of a quantifier has an additional feature (called NP-S below), which is identified with the proposition in the scope part.[4]

(16)



Given this modification, the NP parts of the quantifiers can now compose with conjunction 'and' in such a way that the conjoined expressions are identified with propositions in the scope part of the quantifier.

Compositional interpretation of the sentence in (10) is shown in (17). The semantic representation of the conjunction 'and' includes a proposition l_1, which is a conjunction of propositional variables 2 and 3. In the case of quantificational NPs, as illustrated by the example above, the variables 2 and 3 are identified with l_5 and l_6, which are provided by using feature NP-S.

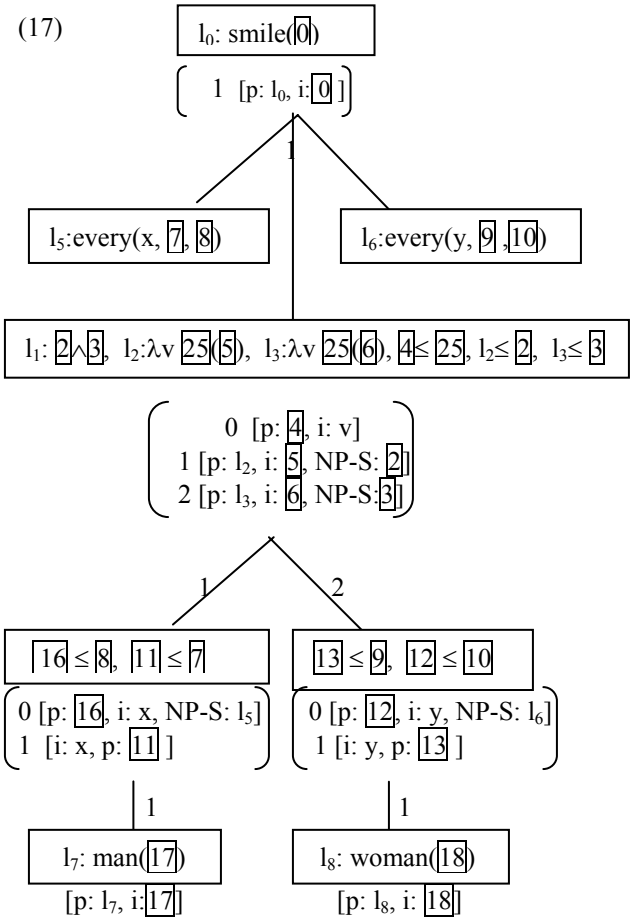
The representation of the conjunction 'and' also contains two propositions l_2 and l_3, which are of the form λv 25(5) and λv25(6, where the variable 25 is a propositional variable, and 5 and 6 are individual variables. It is important, however, that the propositional variable 25 is not unified with any propositional label in the final representation, as we will show below.

[4] This feature can possibly be unified with MaxS, a scope feature proposed in Romero et al 2004 to account for different types of island constraints. The difference is that MaxSc is a feature associated with S trees, whereas NP-S, as described above, specifies the scope of NPs.

representation, as we will show below.

It is also critical for this analysis that the propositional variable which corresponds to the nuclear scope of the quantifier is introduced as part of the NP-tree, not S-tree (as Joshi et al 2003 independently argue, contra Kallmeyer and Joshi 2003). If this variable were part of the S-tree, then the nuclear scope would be identified with a proposition l_0, which is introduced by the S-tree headed by the verb. The desired interpretation, however, is such that the nuclear scopes of the two quantifiers are identified with the propositions l_2 and l_3, introduced by the ConjNP (as the constraints $l_2 \leq 8$ and $l_3 \leq 10$ below show). This interpretation can be achieved, as shown in (17), under the assumption that the feature structures and scope constraints which are responsible for the identification of the nuclear scope of the quantifier are part of the NP tree that attaches to the ConjNP.

(17)



Performing feature unification leads to the following final interpretation of this sentence:

(18)

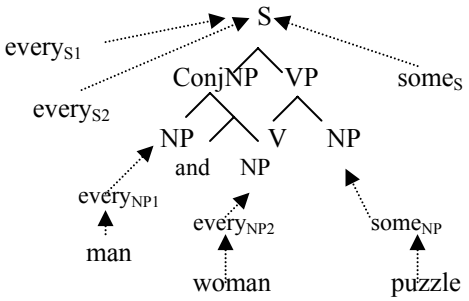
l_1: $l_5 \wedge l_6$
l_0: smile(v) $l_0 \leq 25$
l_2: λv 25(x) $l_2 \leq 8$
l_3: λv 25(y) $l_3 \leq 10$
l_7: man(x) $l_7 \leq 7$
l_8: woman(y) $l_8 \leq 9$
l_5: every(x, 7, 8)
l_6: every(y, 9, 10)

There is only one possible disambiguation of the remaining variables, such that 25 -> l_0 , 8 ->l_2, 10-> l_3 , 7-> l_7, 9 ->l_8 . This disambiguation results in the desired interpretation of the sentence.

As the interpretation in (18) shows, the propositions l_2 and l_3 in the final representation are underspecified in the sense that the propositional variable 25 is not linked to any propositional label. This assumption, as we will see below, allows us to derive an underspecified representation for the scope ambiguities of the sentence in (11).

Semantic representations and feature structures for the sentence in (11) are parallel to the semantic representations in (17), except that there is an additional quantifier.

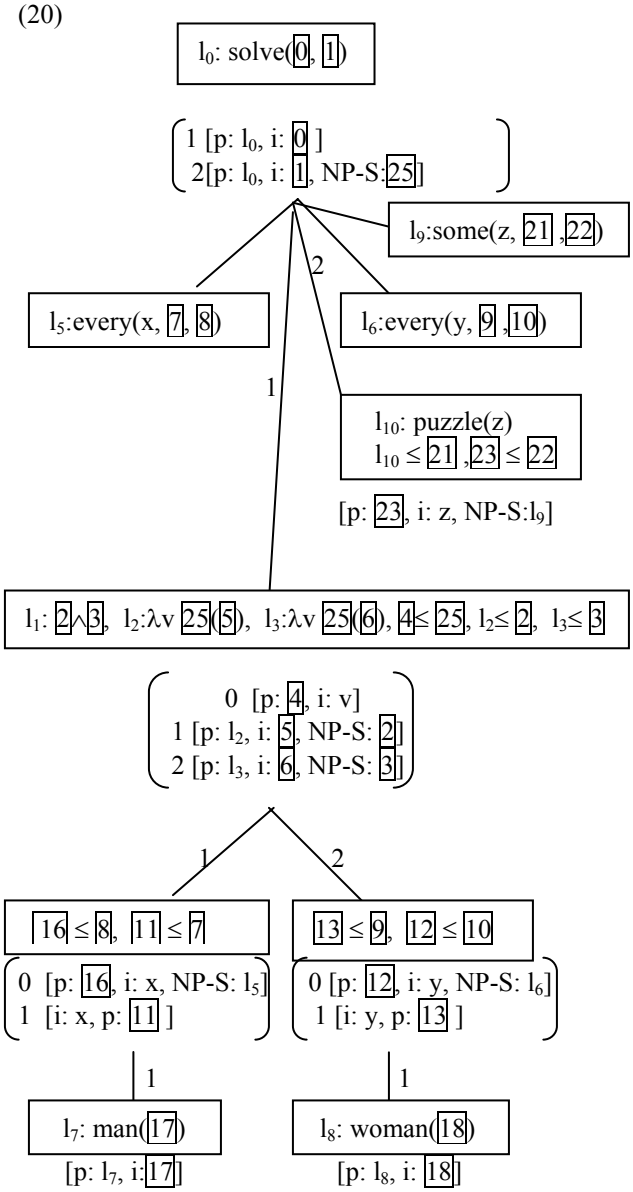
(19)



As the derivation tree is (19) shows, the NP part of the quantifier 'some' is substituted to the NP node of the S tree, whereas the NP-parts of the quantifiers 'every' are substituted to the ConjNP. The scope parts of all three quantifiers are adjoined to S.

The compositional analysis of this sentence which we propose is shown in (20).

(20)



Performing unifications leads to the following final representation:

(21)

l_1: $l_5 \wedge l_6$
l_2: λv 25(x) $l_2 \leq 8$
l_3: λv 25(y) $l_3 \leq 10$
l_7: man(x) $l_7 \leq 7$
l_8: woman(y) $l_8 \leq 9$
l_5: every(x, 7, 8)
l_6: every(y, 9, 10)
l_9: some(z, 21, 22)
l_0: solve(v, z) $l_0 \leq 22$, $l_0 \leq 25$
l_{10}: puzzle(z) $l_{10} \leq 21$

This representation has two possible disambiguations.

The first disambiguation is 22 -> l_0, 25 -> l_9, 8 -> l_2, 10 -> l_3, 7 ->l_7, 9->l_8, 21 ->l_{11}, where the variable 25 is identified with the existential quantifier 'some' (i.e. proposition l_9), and l_0 is identified with its nuclear scope. The propositions l_2 and l_3 in this case are as follows:

(22) l_2: some(z, puzzle(z), solve(x, z))
l_3: some(z, puzzle(z), solve(y, z))

Given that l_2 and l_2 are identified with the nuclear scopes of the quantifiers 'every', the final interpretation is as in (23):

(23) every(x, man(x), some(z, puzzle(z), solve(x, z))) ∧
every(y, woman(y), some(z, puzzle(z), solve(y, z)))

Another possible disambiguation is 25 -> l_0, 22 -> l_1, 8 -> l_2, 10 -> l_3, 7 ->l_7, 9->l_8, 21 ->l_{11},,where 25 is identified with the proposition l_0, so that the propositions l_2 and l_3 are as in (24):

(24) l_2: like(x, z)
l_3: like(y, z)

The nuclear scope of 'some' in this case is identified with l_1, and the final representation represents the second interpretation:

(25) some(z, puzzle(z), every(x, man(x), l_2) ∧
every(y, woman(y), l_3)

As this example illustrates, the desired interpretations are achieved under the assumption that the propositions which correspond to two 'copies' remain underspecified in the final representation.

4 Coordination of non-quantified NPs

Finally, let us extend this analysis to the semantic interpretation of the sentence in (26).

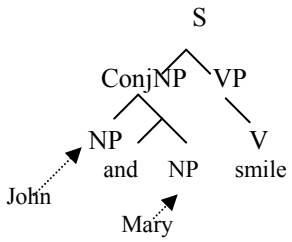
(26) John and Mary smiled.

The desired interpretation of this sentence is 'John smiled and Mary smiled'. To derive this interpretation, the variables 2 and 3 should be identified with the propositions 'smile(x)' and 'smile(y)', as opposed to the coordinated structures with quantified NPs, where these variables were identified with quantifiers.

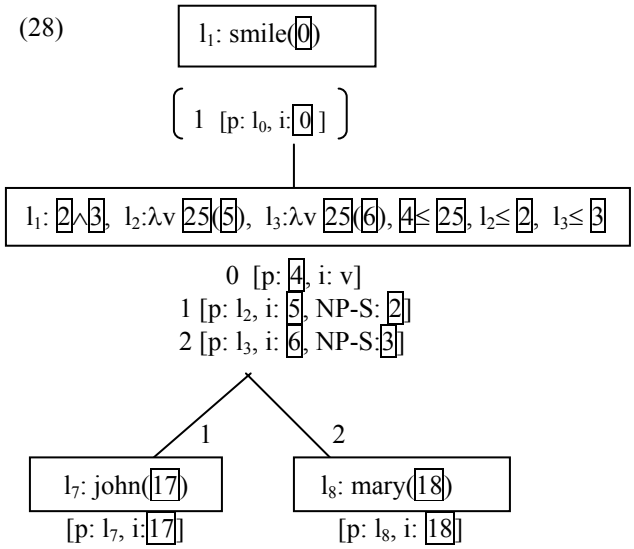
In order to derive the correct interpretation of this sentence, we introduced constraints $l_2 \leq$ 2 and $l_3 \leq$ 3 to the interpretation of the conjunction 'and'. These constraints did not play any role in the analysis of coordinated NPs. If the NPs are not coordinated, however, then these constrains are needed to get the right interpretation.

The derivation tree and compositional interpretation of the sentence in (26) is shown in (27) and (28) below:

(27)



(28)



Performing feature unification leads to the following final interpretation of this sentence.

(29)

l_1: 2 ∧ 3	l_0: smile(v)	$l_0 \leq$ 25
l_2: λv 25(x)	l_7: john(x)	$l_2 \leq$ 2
l_3: λv 25(y)	l_8: mary(y)	$l_3 \leq$ 3

There is only one possible disambiguation of the remaining variables: 25 -> l_0, 2 ->l_2, 3 -> l_3. This disambiguation results in the desired interpretation of the sentence.

5 Conclusion

This paper proposed an analysis of coordinated quantificational and non-quantificational NPs within LTAG

semantic unification framework. It was shown that the analysis of quantifiers which separates scope part and predicate-argument part (e.g. Kallmeyer and Joshi 2003) presents a challenge for a compositional interpretation of conjoined structures. To solve this problem, we proposed to add a new feature to the NP-part of a multi-component quantifier, which would take as its value the propositional label introduced by the scope part.

Another problem discussed in the paper is getting the right scope ambiguities of sentences of the type "Every man and every woman solved a puzzle". Under the analysis of scope ambiguities as resulting from under-specified representation, as proposed in Kallmeyer and Joshi 2003 (alternative ways of analyzing scope ambiguities are discussed in Szabolsci 1997 and Steedman 1999, for example), the question which was raised is to find the right underspecified representation which would account for the two readings of this sentence. Specifically, it was shown that one of the representations of this sentence may require a propositional variable to be identified with two different propositional labels. To solve this problem, we proposed that propositions in the final interpretation that are linked to the nuclear scope of quantifiers are 'underspecified', and are computed in the process of disambiguation.

References

Joshi, A.K. and Schabes, Y. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages*, Springer, Berlin, pp.69-123.

Joshi, A. K., Kallmeyer, L and M. Romero 2003 "Flexible Composition in LTAG: Quantifier Scope and Inverse Linking: in H. Bunt, Ivan der Sluis and R. Morante (eds.) *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5.* Tilburg, pp.179-194.

Kallmeyer, L. and A.K Joshi 2003 Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation 1(1-2),* 358.

Kallmeyer, L. and M. Romero 2004 "LTAG Semantics with Semantic Unification", in *Proceedings of TAG+ 7 Workshop*, Vancouver, BC.

Morrill, G. 1994 *Type-Logical Grammar* Dordrecht, Kluwer.

Partee, B. and M. Rooth 1983 "Generalized Conjunction and Type Ambiguity' in R. Bauerle (ed.) *Meaning, Use, and Interpretation of Language,* Berlin: de Gruyter.

Romero, M., L. Kallmeyer, and O. Babko-Malaya 2004 "LTAG Semantics for Questions", in *Proceedings of TAG+7 Workshop*, Vancouver, BC.

Ross, J.R. 1967. Constraints on variables in syntax. Ph.D. dissertation, MIT

Steedman, M. 1996. *Surface Structure and Interpretation*. Cambridge, Mass. MIT Press.

Steedman, M. 'Alternating Quantifier Scope in CCG', in Proceedings of 37th Annual Meeting of the Association for Computational Linguistics, June 1999, 301-308

Szabolsci, A.(ed.) 1997 *Ways of Scope Taking.* Dordrecht Boston : Kluwer Academic Publishers

Semantics of VP coordination in LTAG

Eva Banik
Linguistics Department
University of Pennsylvania
3401 Walnut street, Suite 400A
Philadelphia, PA 19104
ebanik@babel.ling.upenn.edu

Abstract

This paper proposes to give an analysis of VP coordination in the LTAG semantics framework of (Kallmeyer and Joshi, 2003). First the syntax of VP coordination is described using an operation called *conjoin*. Then we discuss interactions of coordination scope and quantifier scope in simple sentences and their analysis in LTAG. Finally we point out coordination scope ambiguities in embedded sentences that present a problem for the present analysis.

1 Introduction

Perhaps the most natural account of coordination is given in Combinatory Categorial Grammar where the fact that sentences are assigned ambiguous structures not only provides an explanation for all kinds of coordination constructions but also leads to a fully compositional and appropriate semantics.

(Joshi and Schabes, 1991) and (Sarkar and Joshi, 1996) have shown that it is possible to provide a CCG-like account for coordination while preserving the fixed phrase structure of LTAGs by introducing a notion of derivation that allows for the flexibility needed for handling coordination phenomena.

This paper proposes a compositional semantics for VP coordination in LTAG using the notion of derivation as defined by (Sarkar and Joshi, 1996).

The term VP coordination is not fully appropriate to describe the range of phenomena considered here which also includes V- and S-coordination. We will use the term VP coordination to describe coordination phenomena that requires the identification of the shared arguments of two (verbal) predicates.

2 Background

2.1 Syntax of Coordination in LTAG

Because of the locality of arguments in LTAG, it is necessary to introduce a notion of argument sharing in order to handle coordination in this framework.

Making the notation of substitution and adjunction explicit, (Sarkar and Joshi, 1996) represent LTAG trees as an ordered pair of a tree structure and an ordered set of substitution/adjunction nodes from its frontier (see Fig. 1).

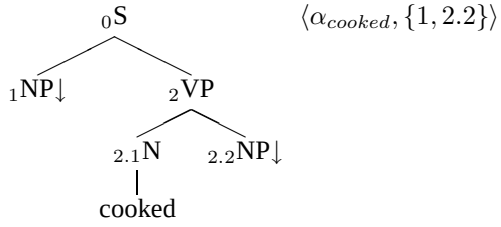


Figure 1: α_{cooked} represented as an ordered pair

Identification of shared arguments is achieved through building contraction sets with the operation *build-contraction*.

Build-contraction takes an elementary tree $\langle\gamma, S\rangle$, places a subset $s \subset S$ from its second projection into a contraction set and assigns the difference $S - s$ to the second projection of the new elementary tree: $\langle\gamma', S - s\rangle$. For example, applying build-contraction to the NP node at address 2.2 in the tree $\langle\alpha_{cooked}, \{1, 2.2\}\rangle$ yields a tree with contraction set $\{2.2\}$: $\langle\alpha_{cooked\{2.2\}}, \{1\}\rangle$ ($\alpha_{cooked\{2.2\}}$ for short). The output of *build-contraction* is shown on Fig.(2).

Coordination is handled by a general coordination schema illustrated in Fig. 3 and a new operation called conjoin (in addition to substitution and adjunction). Conjoin takes three trees and combines them to give a derived tree. One of the trees is always obtained by spe-

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 118-125.

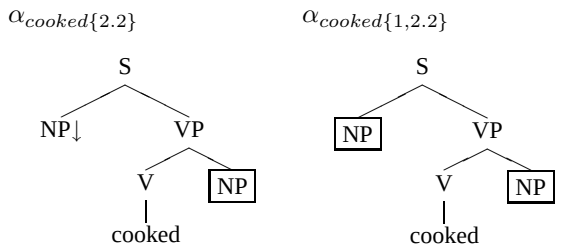


Figure 2: Output of build-contraction

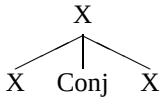


Figure 3: Coordination schema

cializing the coordination schema for a particular category and lexicalizing it with the conjunction. The two trees being coordinated are substituted into the conjunction tree in a special way: the node that is substituted into the conjunction tree is not necessarily the root node but can be some internal node, given by an algorithm called *FindRoot*. *FindRoot* takes into account the contraction sets of the two trees and returns the lowest node dominating all nodes in the second projection of the elementary tree. E.g. FindRoot($\alpha_{cooked\{1,2.2\}}$) will return node address 2.1, corresponding to the *V Conj V* instantiation of the coordination schema, FindRoot($\alpha_{cooked\{1\}}$) will return address 2, corresponding to *VP Conj VP* and FindRoot($\alpha_{cooked\{2.2\}}$) will return the root node, corresponding to *S Conj S* coordination.

The conjoin operation substitutes two elementary trees, T_1 and T_2 into an instance of the coordination schema C using the FindRoot algorithm, creates edges between identical nodes in the contraction sets of T_1 and T_2 and contracts each edge. For example, applying conjoin to Conj(and), $\alpha_{eats\{1\}}$ and $\alpha_{drinks\{1\}}$ gives the derivation tree and derived structure in Fig. 4 and Fig. 5.

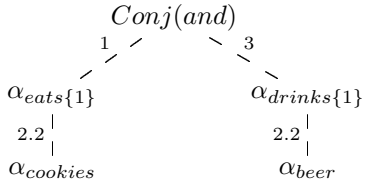


Figure 4: Derivation tree

The contraction set corresponds to a set of arguments that remain to be supplied to a functor. A node in a derivation tree with a non-empty contraction set indicates that the derivation is incomplete.

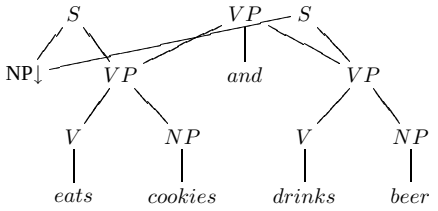


Figure 5: Derived structure

A consequence of introducing contraction and the conjoin operation is that the derivation tree has to be extended to an acyclic derivation graph. If a contracted node in a tree (after the conjoin operation) is a substitution node, then the argument is recorded as a substitution into both elementary trees simultaneously as illustrated in Fig. 6.

(1) Chapman eats cookies and drinks beer

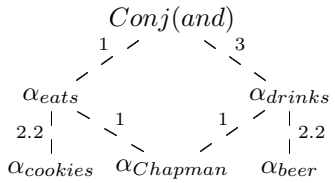


Figure 6: Derivation tree for (1)

An alternative way of viewing the conjoin operation is as a construction of an auxiliary structure from an elementary tree. For example, the conjoin operation would create $\langle\beta_{drinks\{1\}}, \{2.2\}\rangle$ from the elementary tree $\langle\alpha_{drinks}, \{1, 2.2\}\rangle$. In this case, the adjunction operation would create contractions between nodes in the contraction sets of the two trees it applies to.

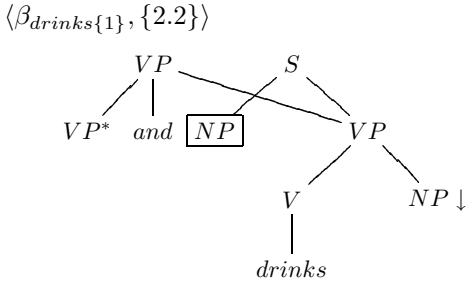


Figure 7: Representing *conjoin* as adjunction

Although this approach requires the same machinery to determine the instantiation of the coordination schema

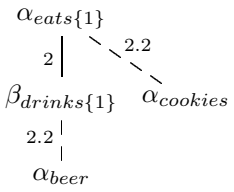


Figure 8: *Conjoin* as adjunction - derivation tree

and to identify shared arguments, it has the advantage that it only uses the traditional LTAG operations of substitution and adjunction. A consequence of this perspective is that the right conjunct is treated as a kind of "modifier" on the left conjunct.

Since we associate semantic representations with individual elementary trees in the lexicon, creating a semantics "on the fly" for the second conjunct combined with the tree for coordination seems less attractive than selecting three elementary trees from the lexicon and combining them with the conjoin operation.

In the rest of the paper we will use the *conjoin* operation to represent the syntax of coordination.

2.2 Semantics in LTAG

We give an analysis in a variant of (Kallmeyer and Joshi, 2003)'s framework. Basic semantic representations are associated with individual elementary trees in the lexicon. They consist of a set of formulas, a set of scope constraints of the form $x \geq y$ (where x,y are propositional labels or propositional variables) and semantic feature structures linked to specific node addresses in the elementary tree (see Kallmeyer and Romero, this volume). Each feature structure linked to a node in the elementary tree consists of a top and a bottom feature structure. Each top and bottom feature structure consists of a feature p and a feature i. The possible values of p are propositional labels and propositional variables, and the possible values for i are individual variables.

Compositional semantics is computed based on the derivation tree. At a substitution or adjunction step, the feature structures are unified just like in a feature-based LTAG (see (Vijay-Shanker and Joshi, 1991))[1]

These unification operations result in value-assignments to some of the variables in the elementary semantic representations. At the end of the derivation, some of the variables will not be assigned a value, therefore the final representation will be underspecified.

The constraints in the final representation specify a partial order on variables and labels (corresponding to the partial ordering on holes and labels in (Kallmeyer and Joshi, 2003)). Disambiguation is performed by assigning values to the remaining variables.

Quantifiers are assigned a multicomponent representation that contains an empty scope tree and a regular NP tree for predicate argument structure[2]. Fig.9 shows the derivation tree for a sentence containing two quantifiers.

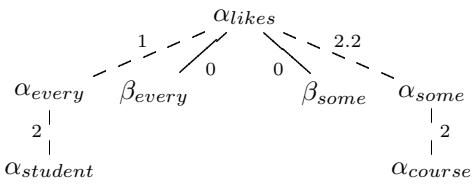


Figure 9: Derivation tree for *"Every student likes some course"*

Following (Kallmeyer and Romero, 2004) (this volume), the semantic representation of quantifiers contains a feature called MaxS to make sure that in a sentence like *"Mary thinks that John likes everybody"* the quantifier can't take scope over *thinks*. The value of the MaxS feature of a quantifier will be identified with the MaxS feature linked to the S node of the tree where the scope part adjoins. Fig.10 illustrates the semantic features associated with the derivation tree in Fig.9. When the two nouns are substituted into the NP parts of the two quantifiers, the individual variables x and y are identified with variables 6 and 7 and when the quantifier is combined with the verb tree the propositional variables 81 and 31 are identified with l_5 and l_3 respectively. Other feature unifications during semantic composition include 41 = l_1, 91 = l_1, MaxS 21 = MaxS 23, MaxS 20 = MaxS 23. The final (underspecified) representation along with the two possible disambiguations is given on Fig.11.

3 Interactions of Quantifier scope and Coordination scope

Analogously to the two perspectives on the syntax of coordination in LTAG (*conjoining* or creating an auxiliary tree from the left conjunct), there have been two approaches to coordination phenomena in the literature: conjunction reduction (deriving coordination from deletion within conjoined sentences) and base generated phrasal conjunction.

[1] At a substitution step, the top feature of the substitution node in the host tree is unified with the top feature of the root node in the substituting tree. At an adjunction step, the top feature of the root of the adjoined tree is unified with the top feature of the node where adjunction takes place and the bottom feature of the foot node is unified with the bottom feature structure of the adjunction site.

[2] In this paper, we adopt a substitution analysis for determiners, i.e. nouns are substituted into the determiner tree (as opposed to the determiner tree being adjoined onto the noun)

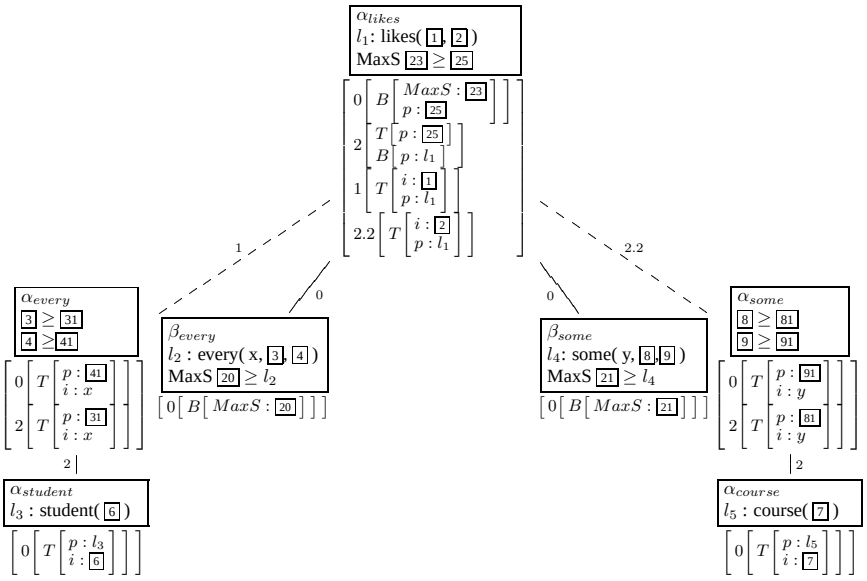
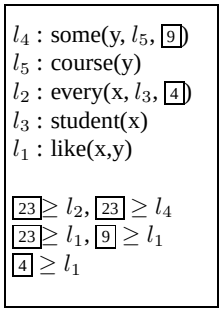


Figure 10: Derivation tree enhanced with semantic features for *"Every student likes some course"*



I. 23 → l_2, 9 → l_1, 4 → l_2

every(x, l_3, some(y, l_5, l_1))

II. 23 → l_4, 9 → l_2, 4 → l_1

some(y, l_5, every(x, l_3, l_1))

Figure 11: Semantics for *"Every student loves some course"*

Based on evidence from e.g. agreement and binding phenomena in various languages, it has been argued that the two conjuncts are not syntactically equivalent. One example is (Munn, 1993) which presents arguments for treating coordinate structures structurally identical to adjuncts. However, semantically the arguments of coordination seem to be of the same type. Various researchers (e.g. (Keenan and Faltz, 1978), (Partee and Rooth, 1983)) have shown that conjunction can be generalized to provide a uniform meaning for *and* and *or*. Although it has also been suggested (e.g. (Larson, 1985), (Winter, 1995), (Winter, 2000)) that conjunction and disjunction have different scopal properties, in this paper we will follow the former line of analysis and assign them equivalent denotations.

First we consider the interaction of quantifier scope and coordination in simple sentences. We say that coordination has wide scope in a construction $Y\ [X_1\ coord\ X_2]$ if the meaning of the construction can be paraphrased as $[Y\ X_1]\ coord\ [Y\ X_2]$.

In cases like (2) the wide scope and the narrow scope readings are logically equivalent, therefore impossible to distinguish.

(2) a Every girl sang and danced.
b Some girl sang or danced.
c John sold or bought a car.
d John caught and ate every fish.

However, coordination scope should be in principle visible in case of disjunction in scope of a universal (every(A, B ∪ C)) and in case of conjunction in the scope of an existential (some(A, B∩ C))[3]. (3) illustrates two such contexts with the quantifier occurring in subject position.

(3) a Some girl sang and danced.
$\exists x[\ girl(x) \wedge sang(x) \wedge danced(x)\ ]$
b Every girl sang or danced.
$\forall x[\ girl(x) \rightarrow sang(x) \vee danced(x)\ ]$

In both cases only the narrow scope reading is available (i.e. the quantifier has scope over the coordination). The same effect can be observed if we replace *some* and *every* with any of the following quantifiers: no girl, not every girl, at least/most five girls, exactly five girls, most girls. Similar scope relations can be observed in (4) where the quantifiers occur in object position.

[3]We are not concerned here with coordination in the restriction of quantifi ers. For an account of NP coordination in this framework see (Babko-Malaya, 2004), this volume.

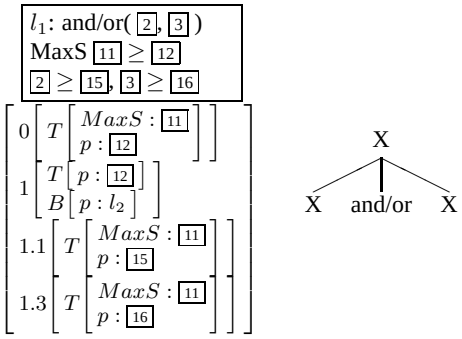


Figure 12: Semantics for *and*/*or*

(4) a John sold or bought every house in this neighborhood.
$\forall x[house(x) \rightarrow sell(j,x) \wedge buy(j,x)]$
b John caught and ate a fish.
$\exists x[fish(x) \wedge caught(j,x) \wedge ate(j,x)]$

However, world knowledge often influences the preferred interpretation. C.f. (5) where the wide scope reading (5b) is prominent.

(5) John sold and bought a car.
a $\exists x[car(x) \wedge sell(j,x) \wedge buy(j,x)]$
b $\exists x[car(x) \wedge sell(j,x)] \wedge \exists x[car(x) \wedge buy(j,x)]$

As a first approximation, we will assume that quantifiers take highest scope in the clause [4] and delegate sentences like (5) to world knowledge or pragmatic factors.

Fig.12 illustrates the elementary semantic representations assigned to *and* and *or*. Note how the MaxS features of both conjuncts are identified with the MaxS of the coordination, resulting in one single MaxS value for the coordinated sentence. This means that the quantifiers that are attached to both conjuncts will automatically have scope over the coordination.

Since coordination doesn't target the root node but takes place at the lowest node that dominates the non-shared arguments of the conjoined elementary trees we need to add the same MaxS feature to all the nodes where coordination can potentially take place (i.e. to V and VP nodes in addition to S)[5].

Fig.13 illustrates the derivation tree extended with semantic features for (4b) and Fig.14 shows the final semantic representation after feature unification and disambiguation. Notice how the desired scope relations are achieved by identifying the MaxS feature of the quantifier with both of the conjuncts and the coordination. The relevant feature identities are 11 = 21 = 31 = 14.

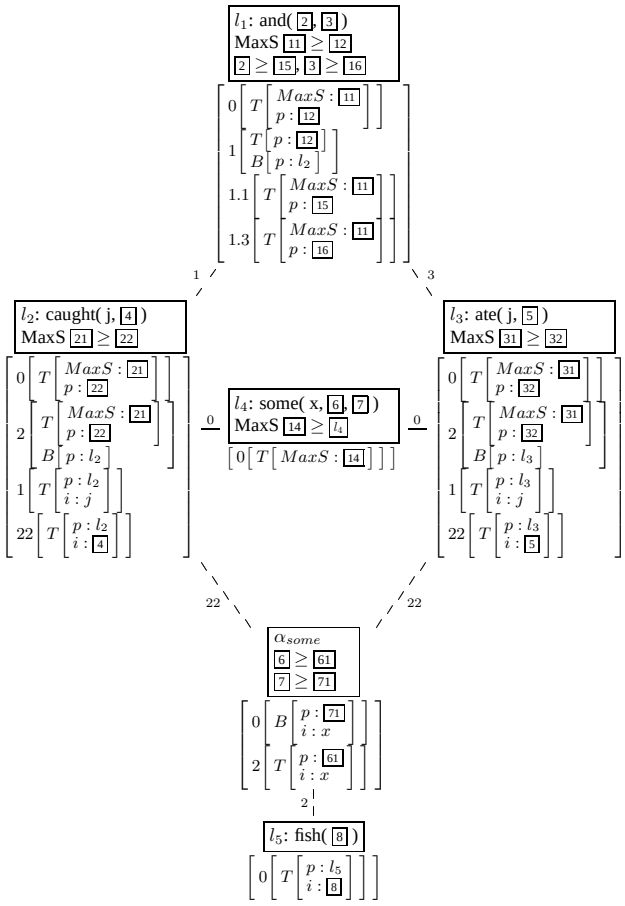


Figure 13: Semantics for *"John caught and ate a fish"*

This analysis of coordination has the consequence that whenever two quantifiers are shared between the two VPs like in (6), both will have scope over the coordination but their relative scope will be underspecified. The resulting semantic representation after feature unification is underspecified for the two readings in (6a) and (6b). Fig.15 shows the semantics and the two possible disambiguations for (6).

(6) Most girls dated and kissed a guy from the neighborhood.
a *most(x,girl(x), some(y,guy(y),and(date(x,y),kiss(x,y))))*
b *some(y,guy(y), most(x,girl(x),and(date(x,y),kiss(x,y))))*

The two readings result from identifying the "highest" MaxS (11) with either the label of $some$ or the label of

[4] We assume for the moment that there are no other scope-taking elements (e.g. wh-phrases) in the clause.

[5] Alternatively, we could defi ne a different kind of semantics for *conjoining* that would have access to the features from the S nodes of the two conjuncts as well as to the features of the node where conjoining takes place.

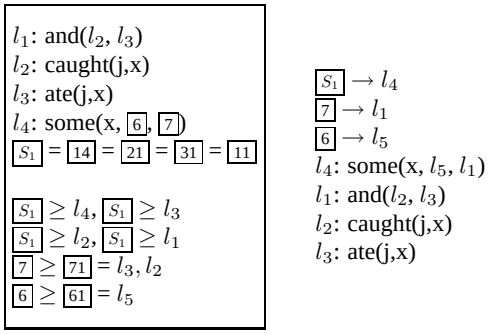


Figure 14: Final semantic representation for (4b)

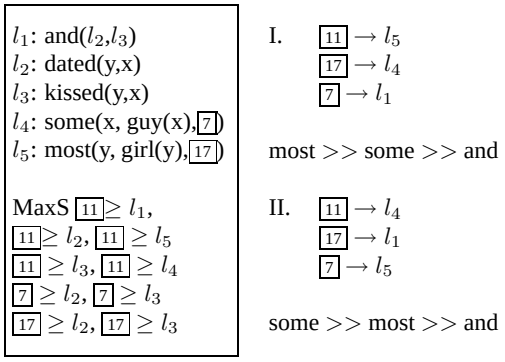


Figure 15: Final representation for (6)

$most$. If we give $some$ highest scope ($\boxed{11} \rightarrow l_4$) that will force $most$ to appear in the scope of $some$ and the coordination to be identified with the scope of $most$ (since both quantifiers have to have scope over the coordination). The reverse scope reading is computed analogously.

(7) illustrates a sentence where both conjuncts have two quantified arguments but only one of the arguments is shared by the two verbs. Our analysis predicts that in this case the shared quantified argument will take scope over the coordination while the two non-shared arguments will have scope below the coordination, i.e. we will get the reading $most >> and >> some_{1,2}$.

(7) Most girls dated a student but had a crush on a teacher.
most(y,girl(y),
∃(x, stud(x), date(y,x))∧∃(z,tea(z),crush(y,z)))

The semantic representation of (7) after feature unification is illustrated in Fig.16. There is only one possible disambiguation in this case. Theoretically, either $some_1$, $some_2$, and or $most$ could have widest scope in the sentence. However, if we identified 11 with l_4 or l_6 we would end up with a contradiction where an argument variable (e.g. 8) would be identified with the

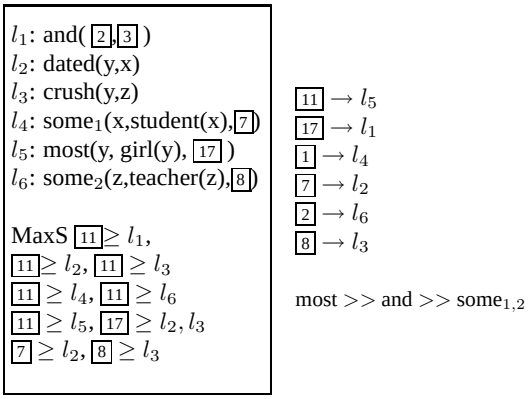


Figure 16: Final representation for (7)

label of the proposition it occurs in (l_6). Identifying 11 with l_1 (i.e. giving the coordination widest scope) would result in a representation where one occurrence of y is outside of the scope of the quantifier that introduced it: [most(y, girl(y), some$_1$(x, student(x, date(y,x))))] AND [some(z, teacher(z), crush(y,z))]. The only possible disambiguation (illustrated in Fig.16) is when $most$ takes widest scope, i.e. 11 is identified with l_5.

4 Other Coordination scope ambiguities

Unfortunately, the above analysis of coordination only works for simple sentences. There are several contexts when coordination can have a wide scope reading. The most famous examples are cases of wide scope readings of *or* in intensional contexts. (Rooth and Partee, 1982), (Larson, 1985) pointed out that when *or* is embedded under one or more intensional operators multiple scopal readings are possible similar to quantified NPs. Most famous examples involve NP coordination (e.g *"Mary is looking for a maid or a cook"*) but there are also cases of wide scope *or* readings for VP disjunction, like the sentence in (8) which is three ways ambiguous.

(8) John believes that Bill said that Mary was drinking or playing video games.
a *J. believes B. said [drink(m) ∨ play(m)]*
b *J. believes ([B. said drink(m)] ∨ [B. said play(m)])*
c *[J. believes B. said drink(m)] ∨ [J. believes B. said play(m)]*

Although they are harder to find, there are also unexpected wide scope readings of *and* (example from (Winter, 1995)):

(9) A woman discovered Radium but a man invented the electric light bulb and developed the theory of relativity.

(9) doesn't attribute the invention of the light bulb and developing the theory of relativity to the same person, rather it says that a man invented the electric light bulb and a man developed the theory of relativity.

There are also examples of wide scope *or* outside of intensional contexts as (10) shows.

(10) (The girls didn't all do equally well in the exam but) every boy failed or got an A.

Unlike the scope of quantifiers, the scope of coordination can appear over a that-clause as well. Consider the scope of *or* in (11) (from (Winter, 1995)).

(11) Mary says that [$_{S_1}$ John is going to marry Sue] OR [$_{S_2}$ Sue is going to divorce Bill].

a Mary says "S_1 or S_2"

b Mary says S_1 or Mary says S_2

A critical situation that distinguishes the two possible readings illustrated in (11a) and (11b) would be the following: Mary says: " Sue and John are going to get married.". Reading (11a) would be false in this situation whereas the sentence in (11) would be true which shows that reading (11b) is attested.

The same phenomenon can be observed with the scope of *and* in (12).

(12) Mary denies that [$_{S_1}$ John is going to marry Sue] AND [$_{S_2}$ Sue is going to divorce Bill].

a Mary denies "S_1 or S_2"

b Mary denies S_1 or Mary denies S_2

A critical situation here would be the following: Mary says:"I don't think John and Sue are going to get married but I'm sure Sue and Bill are going to get divorced". Sentence (12) would be false in this situation, whereas (12a) would be true which means reading (12b) is attested.

The above examples show that the scope of coordination doesn't always obey the syntactic restriction on the scope of quantifiers.

It seems that all the instances of wide scope coordination involve embedding under a matrix verb or some other contextually determined operator (e.g. possible generic reading in (9)). However, not all such embeddings result in scope ambiguities. Complex NPs for example are islands for coordination scope as the unambiguous sentence in (13a) (cf. the ambiguous ((13)b)) shows.

(13) a John maintains the claim that Bill should resign or retire.

b John maintains that Bill should resign or retire.

Since an account of the wide scope readings of coordination would require a more complex semantic theory, I will not attempt to give a full analysis of the examples discussed above. In the rest of this paper I will settle for pointing out some potential problems that an analysis in the LTAG semantics framework would have to face dealing with these facts.

The first problem our analysis would encounter would be picking an S node where a matrix verb could be adjoined. In a derived tree containing VP coordination (see e.g. Fig.5) there are two available S nodes. We could simply equate the two nodes and adjoin a matrix verb on top. This would have the consequence that nothing else could come in between the matrix verb and the coordinated trees, i.e. nothing else could be adjoined onto either of the conjuncts.

Another solution would be to extend the coordination schema and add an S node on top of the coordination for each possible instantiation of the schema. This would have the advantage that the S nodes of the two conjuncts would be distinct and still available for adjunction in case something else (e.g. an adverb) adjoins to one of the conjuncts. The extended instances of the coordination schema are illustrated in Fig.17.

To decide between these two alternatives we would need to consider more data about sentences that involve adjunction at the S node in addition to coordination.

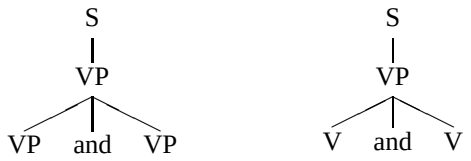


Figure 17: Extended coordination schema

Keeping quantifier scope separate from coordination scope constitutes another challenge for the semantic theory. In a sentence like (8) we need to make sure that coordination can scope over the verb tree it is substituted into, i.e. we need to derive the following scope relations: $believe >> or >> said$ and $or >> believe >> said$. At the same time we also have to make sure that the scope of quantifiers that are embedded in the conjuncts doesn't get passed up the derivation tree. One way to ensure this is to define a feature for coordination scope that is different from the MaxS feature used for representing quantifier scope.

Finally, another problem is that in order to account for the wide scope readings of sentences like (8) we need more than one copy of a formula, instantiated with different arguments.

To model readings (8b) and (8c) we would need the following variable assignments given the simplified semantic representation in Fig.18. To give *or* scope over *said* we need to identify both arguments of the coordination with the label of *said*, yielding the formula

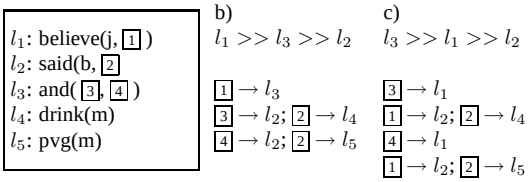


Figure 18:

$believe(or(said(l)4), said(l_5))$. Similarly, in the case of reading c) where *or* has widest scope, we need to identify both of its arguments with the label of *believe* resulting in the reading $or(believe(said(l_4)), believe(said(l_5)))$.

However, this doesn't mean simply assigning the same value to two different variables: in both cases the most embedded arguments of the formula have to be different (l_4 and l_5). This means that for reading b) we need two copies of l_2 ($said$) and for reading c) we need two copies of l_1 ($believe$) and two copies of l_2 ($said$), each time with a different argument, as if the two verbs were 'distributed' over the arguments of or..

5 Conclusions

We have defined a compositional semantics for VP coordination in LTAG using the framework of (Kallmeyer and Joshi, 2003) extended with semantic features. We have discussed interactions between quantifier scope and coordination scope in simple sentences, proposed an elementary semantic representation for coordination and showed that it yields the correct interpretation for basic scope interactions.

The analysis predicts that in simple sentences quantifiers that are shared arguments of two coordinated elementary trees will have scope over coordination whereas quantifiers that are attached to only one of the conjuncts will have narrow scope with respect to the coordination.

We have discussed cases of wide scope disjunction and conjunction in complex sentences that present a problem for this account and pointed out directions for further improving the analysis.

References

Olga Babko-Malaya. 2004. LTAG semantics for NP coordination. In *Proceedings of TAG+7, Vancouver, May 2004*.

Aravind K. Joshi and Yves Schabes. 1991. Fixed and flexible phrase structure: coordination in tree adjoining grammars. In *Proceedings of a workshop on Speech and natural language*, pages 195–199. Morgan Kaufmann Publishers Inc.

Laura Kallmeyer and Aravind Joshi. 2003. Factoring predicate argument and scope semantics: Underspecified semantics with LTAG. *Research on Language and Computation*, 1:1-2:3–58.

Laura Kallmeyer and Maribel Romero. 2004. Ltag semantics with semantic unification. In *Proceedings of TAG+7*.

E. Keenan and L. Faltz. 1978. Logical types for natural language. In *UCLA Occasional Papers in Linguistics 3*. Department of Linguistics UCLA.

R.K. Larson. 1985. On the syntax of disjunction scope. *Natural Language and Linguistic Theory*, 3:217–264.

Alan Munn. 1993. *Topics in the Syntax and Semantics of Coordinate Structures*. Ph.D. thesis, U. Maryland.

Barbara Partee and M. Rooth. 1983. Generalized conjunction and type ambiguity. In Bauerle et al., editor, *Meaning, Use, and Interpretation of Language*, pages 361–383. De Gruyter, Berlin.

Mats Rooth and B. Partee. 1982. Conjunction type ambiguity and wide scope **or**. In *Prceedings of the First West Coast Conference on Formal Linguistics*, pages 353–362. Stanford University.

A. Sarkar and A. Joshi. 1996. Handling coordination in a tree adjoining grammar. Technical report, Dept. of Computer and Info. Sc., Univ. of Pennsylvania, Philadelphia, PA.

K. Vijay-Shanker and Aravind K. Joshi. 1991. Unification based tree adjoining grammars. In J. Wedekind, editor, *Unification-based Grammars*. Cambridge, Massachusetts: MIT Press.

Yoad Winter. 1995. Syncategorematic conjunction and structured meanings. In Mandy Simons and Teresa Galloway, editors, *Proceedings from Semantics and Linguistic Theory V*, pages 387–404, Ithaca, New York. Cornell University.

Yoad Winter. 2000. On some scopal asymmetries of coordination. In Bennis et al, editor, *Proceedings of the KNAW conference on Interface Strategies*.

Verification of Lexicalized Tree Adjoining Grammars

Valerie Barr, Ellen Siefring
Department of Computer Science
Hofstra University
Hempstead, NY 11549-1030 USA
vbarr@hofstra.edu

Abstract

One approach to verification and validation of language processing systems includes the verification of system resources. In general, the grammar is a key resource in such systems. In this paper we discuss verification of lexicalized tree adjoining grammars (LTAGs) (Joshi and Schabes, 1997) as one instance of a system resource, and as one phase of a larger verification effort.

1 Introduction

The work presented here is part of a larger project that has the goal of developing a suitable automated approach to verification and validation of natural language processing (NLP) systems, including structural (white-box) (Beizer, 1990) testing techniques that are suitable for language applications. In previous work (Barr and Klavans, 2001) we established that it is worthwhile to adapt for NLP systems the standard verification, validation, and testing practices that have been developed in the software engineering and intelligent systems communities. These new techniques will supplement the evaluation practices currently carried out and, in many cases, will not require significantly larger test sets.

For our working definitions we combine definitions from the intelligent systems community (e.g. (Gonzalez and Barr, 2000)) and from the software engineering community (e.g. (Voas and Miller, 1995)), as follows:

- Verification – the process of ensuring 1)that the intelligent system conforms to specification, and 2) its knowledge base is consistent and complete within itself; the application of dynamic software testing techniques involving both functional (black-box) and structural approaches.
- Validation – the process of ensuring that the output of the intelligent system is equivalent to that of human experts when given the same input.

As we have noted elsewhere (Barr and Klavans, 2001) there are a number of diagnostic evaluation methods that do a validation check on a system by carrying out a functional test and comparing actual results to expected results (provided by and compared by humans). There are also evaluation methods that allow us to determine whether a system conforms to its specification.

There are a number of methods that are still needed, however. First, we need to determine whether the knowledge represented within an NLP system is consistent and complete. The research presented in this paper begins to address this topic. Specifically, we detail work we have done on the verification of Lexicalized Tree Adjoining Grammars (LTAGs), specifically as implemented in the XTAG formalism (Joshi and Schabes, 1997). As described in the body of the paper, we have constructed a set of structural and relational tests for a LTAG that identify certain lexical and syntactic errors. We applied these tests to subsets of XTAG for English (as examples of a sublanguage in the XTAG formalism), using off-the-shelf database software.

In addition to the above, we need to determine ways by which we can obtain the benefits of structural testing for NLP systems and their components. This will be the subject of future research we plan to carry out. An additional open question, which we do not address here, is whether a more complete verification process will facilitate greater automation of the validation process.

NLP systems are built for a large number of application areas, such as speech recognition, language understanding, language generation, speech synthesis, information retrieval, information extraction, and inference (Jurafsky and Martin, 2000). Systems built for these application areas will differ in terms of the resources they include, the kind of input they expect, and the kind of output they

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 126-131.

generate. In order to narrow the scope of our work at this stage, we focus initially on natural language generation (NLG) systems.

2 Overview of Verification of NLG Systems

Dale and Mellish (Dale and Mellish, 1998) have suggested a direction for improving evaluation of NLG systems. Their proposal is that, rather than attempt to evaluate a complete system, the evaluation effort address the component tasks of the NLG process. They suggest a breakdown of the NLG process (Reiter and Dale, 2000; Dale and Mellish, 1998) into the six tasks of content determination, document structuring, lexical selection, referring expression generation, aggregation, and surface realization. This approach is consistent with our proposal (Barr and Klavans, 2001) that we carry out a component performance evaluation, in order to determine the impact on overall system performance of each subpart or subtask. In other work (Barr, 2003) we began to address the verification and validation questions relevant for each of these generation tasks. (The components of interest will differ across different types of language systems. See (Webber et al., 2002) for an example in the Question-Answering domain).

Another important area to consider is the issue of utilization of linguistic resources by a language processing system. This is an area that we believe cannot be adequately addressed by traditional testing approaches. Typically a language processing system has numerous resources within it, such as the lexicon, the grammar, morphological rules, a pragmatics component, and semantic knowledge (both formal and lexical).

There are a number of aspects of system behavior that are affected by the various resources. For example, it would be useful to clarify exactly how an incomplete lexicon affects system behavior. Or there may be sub-processes within a language generation system that should be verified separately because they utilize only a subset of the available resources. We are also interested in how the various resources participate in the input-output relationship. For example, can we determine which of a system's linguistic resources contributes to the transformation of an input to an output? Can we pinpoint exactly how each element of an output is affected by each linguistic resource? If the grammar in a generation system is capable of parsing a sentence, is there some context in which the system will generate that sentence?

Developing mechanisms for addressing these issues will enable us to more accurately assess the overarching verification issue, which is whether the system does the task, and only the task, for which it was intended. As part of our larger project we intend to define what it means to evaluate all the linguistic resources for completeness and consistency. As a first step in this aspect of verification, we focus on an assessment of the completeness and consistency of the grammar alone.

Previous testing approaches have attempted to identify grammar errors through evaluation of parse system coverage using test-suite or corpus-based methods (Doran et al., 1994; Doran et al., 1997; Bangalore et al., 1998; Prasad and Sarkar, 2000). While these testing approaches are vital to a complete test plan, the source of errors identified through these methods must be manually researched and categorized as a grammar or application defect. If a grammar error is suspected, the underlying grammar must be examined to determine if the error is a coverage issue or grammar fault. Our structural approach to grammar verification insures that grammar defects are identified and corrected early in the testing cycle, before the grammar is embedded in a component application, such as a parser. Our expectation is that this will improve grammar reliability, and subsequent test efforts may then focus on coverage and application defect issues.

3 Grammar Verification

The first step in verifying a grammar is to assess consistency and completeness. We cannot necessarily do this by applying existing methods from other domains. How we do it depends on the kind of grammar used. We have, from the expert systems' realm, methods and tools that are suitable for rule-based systems (for example, the TRUBAC tool (Barr, 1999)). However, the rule formalism, while used in some aspects of NLP, is frequently not used for grammar representation. Yet adapting to the grammar of an NLP system the underlying approach used for rule-based systems may give us the ability to determine consistency and completeness of a grammar.

The grammar formalism we focus on initially is the Lexicalized Tree Adjoining Grammar (LTAG), based on the original TAG formalism(Joshi et al., 1975; Joshi, 1987; Joshi and Schabes, 1997). Analysis of the consistency and completeness of an LTAG will serve as a first step toward the full verification and validation of the generation system in which the LTAG is used.

Our motivation to work with LTAGs, particularly with the XTAG formalism (XTAG Research Group, 2001), is threefold. First, we chose XTAG for English as a vehicle to demonstrate proof of concept of our verification approach. Certainly, given the extensive work that has been done on the XTAG for English, we did not anticipate that we would find any errors in the grammar. However, our expectation is that a verification methodology for XTAG grammars could also be adapted to other key grammar formalisms as well. Second, we assume that there are language systems for which a smaller, domain specific, grammar and sub-language would be desired. The grammar might be a subset of an existing XTAG, such as the XTAG for English, or it might be a newly constructed

grammar that employs the XTAG formalism (Kinyon and Prolo, 2002). The verification steps we propose would be able to detect errors or potential problems in such a grammar. Finally, any language system will be tested with a domain specific test suite. However, a set of static verification tests can serve as a useful and important step before a black-box test is carried out, and can potentially unearth grammar problems that might be masked in functional test results.

4 LTAG Verification

While it is possible that existing mechanisms for evaluating the consistency and completeness of the antecedent-consequent rules in an expert system could be used to do the same for the rewrite rules making up a phrase-structure grammar (PSG), these are not relevant for a grammar made up of trees, not rules. Given a grammar made up of trees, we cannot directly apply the characteristics that are used in evaluating rule-bases for consistency and completeness (conflict, redundancy, circularity, subsumption, unreachability, dead-ends, etc.), but rather must adapt the concepts of completeness and consistency for use with LTAGs.

The characteristics we currently check for in an LTAG can be divided into two categories, structural and relational. Structural tests include ensuring that each elementary tree is properly lexicalized, structurally correct and unique. This includes checking for proper tree hierarchy (e.g. unique root, one parent for each child node, proper tree level and node order) as well as TAG specific checks (e.g. each tree is properly anchored, leaf nodes marked with a phrasal label are substitution sites, no adjunction nodes exist in initial trees, label of adjunction node and root must be the same in an auxiliary tree). Trees with identical structures are flagged. Generally, structural errors will arise from incorrect coding or errors in the translation of the LTAG into a machine representation.

Relational tests look at the relationships between tree structures to identify that:

1. Each auxiliary tree can adjoin in at least one derived tree structure, i.e. every auxiliary tree can be used.

2. Each non-S rooted initial tree can substitute in at least one derived tree structure, i.e. every initial tree can be used.

3. At least one substitution operation can be performed at every substitution node in a derived tree, i.e. a tree exists for each substitution node.

4. All derived trees built using substitution operations are finitely bounded with no recursive end nodes (no recursive sentences or phrases). We cannot eliminate recursion, since adjunction allows unbounded sentences. However, if we consider only substitution, we can insure that a tree substituted at a node does not contain a node with the same phrasal label as an ancestor node.

5. Every sentence that can be built using substitution operations alone has a unique derivation tree structure. While the existence of multiple derivation tree structures does not necessarily represent a grammar error if part-of-speech ambiguity is considered, it could indicate conflicting semantic representations if tree anchors are not properly chosen with respect to linguistic relevance.

These checks on the grammar enable us to identify potential grammar errors such as

1. superfluous trees, which could be indicative of missing trees or errors in other trees. (A tree T may be superfluous, or unusable, because there is no other tree that presents a suitable adjunction or substitution use for T, or because there are errors that prevent a suitable adjunction or substitution site from being identified as such).

2. invalid tree structures, a grammar error which could cause an incorrect generation path to be chosen.

3. missing trees, which may indicate incomplete/inaccurate linguistic realization or communicative intent compromised.

4. duplicate trees, which will violate consistency.

5. redundant trees, which may indicate conflicting linguistic interpretations of anchor. This could happen if the linguistic assumptions on how elementary trees should be formed are not consistently followed in the grammar.

We are presently working on an extension of the work presented here that will identify relational problems in feature based LTAGs. (A static analysis approach that identifies structural problems with feature structures in XTAG (typographical errors, reference of undefined features, equating of incompatible features) is introduced in (Sarkar and Wintner, 1999)).

5 Implementation

We have constructed a system that carries out the above verification checks for an LTAG, employing a relational data representation of LTAG tree structures using the Oracle Database Management System. This relational database model provides the benefits of data independence (with the ability to separate the physical implementation from the logical view), multiple views of the same data (through structured queries across tables), data

consistency (enforcing completeness and consistency of schema), and management of data relationships (via table indexes, primary and foreign keys). In addition, the DBMS approach allows us to efficiently manage and access large quantities of structured data which insures future scalability for large grammars. Data verification is performed using SQL*PLUS, the PL/SQL language and reporting tool of the Oracle Database Client/Server product.

The system operates in four stages: tree conversion, structural testing, relational testing and reporting. Oracle tables are used to store type, classification, and node information about each tree. Tree structures in the grammar are automatically converted into the SQL Data Manipulation Language format to systematically build the associated Oracle tables. Structural tests are performed on each converted table to insure tree and lexical consistency. Relational tests perform comparisons on groups of tree structures to identify missing trees, unused trees and, using substitution operations alone, recursive and non-unique derivations. Control and error information is generated during the verification process.

Initially, tree structures are converted to a non-indexed database table set. This enables structural tree errors such as duplicate nodes to be identified and classified by our testing tool, not the DBMS product. A second conversion is then performed to assign primary and foreign keys to tables, encapsulating the data relationships into the structures. Tree nodes are stored as separate table rows, with identifying tree hierarchy represented as three-tuples of (level, order, parent order). Tree traversals may be accomplished in any order using either the indexed keys or identifying node characteristics (e.g. substitution nodes). Substitution and adjunction operations are performed using constrained table join operations.

6 Results

We have used the XTAG for English to test our grammar verification tool. Since XTAG system releases have been extensively utilized and broadly tested (Doran et al., 1994; Doran et al., 1997; Bangalore et al., 1998; Prasad and Sarkar, 2000; XTAG Research Group, 2001), we did not expect our verification tool to uncover any structural grammar defects in the current release of XTAG. We did, however, expect to identify non-unique derivation structures due to inherent sentence ambiguity in the English language. Additionally, we expected to identify as duplicates certain tree structures that are unique when node features are taken into account.

The results from our grammar verification on XTAG are encouraging. We ran our grammar verification tool on an XTAG set of 1,135 trees with a total of 11,514 nodes. We are able to make the following observations from our results:

- There are no errors in tree hierarchy. Every tree has one unique root node. Each non-root node has one parent node and consistent tree node level and ordering.
- Two tree structures have unidentified part of speech node values. Both trees have internal nodes of 'p'. Since we consider case in our validation of POS, these nodes were flagged as errors.
- There were 128 duplicate tree structures in the grammar. This was an anticipated result. Our expectation is that when we consider node features these trees will be identified as unique structures.
- Every tree was properly lexicalized. That is, there was at least one anchor node identified for each tree structure.
- There were three errors in tree classification. One tree was classified as an auxiliary tree but structurally looks like an initial tree. Two trees were classified as initial trees but structurally look like auxiliary trees. We used XTAG tree naming conventions as alpha or beta to drive our classification scheme. It must be determined if the conversion requirements must be modified or if this is a tree classification discrepancy in XTAG.
- Other than the three trees with classification discrepancies, every elementary tree was structurally correct. Every non-terminal node on the frontier marked with a phrasal label was identified as a substitution node. There were no internal nodes marked for substitution, and in the initial trees no internal nodes were marked as adjunction nodes . For auxiliary trees, there was one unique adjunction node per tree. This adjunction node was on the frontier and matched the POS node value of the tree root.
- Every non-S rooted initial tree was able to substitute in at least one derived tree structure. All initial trees could be used in the grammar.
- Every auxiliary tree was able to adjoin in at least one derived tree structure. All auxiliary trees could be used in the grammar.
- There exists at least one tree eligible for substitution at each substitution node in the grammar. Substitution operations may be performed until all frontier nodes are terminals.
- Application performance could be improved by database performance and tuning techniques. While proof of concept, not processing efficiency, was the

initial motivation for this work, subsequent development efforts should consider performance as an implementation requirement.

Identifying non-unique derivation structures using the full XTAG has proven more difficult. While the use of Oracle as our implementation paradigm allows us to efficiently retrieve, manipulate and store large amounts of data, our attempt to build all possible sentence derivations for a complete grammar proved too exhaustive. We modified our approach to maintain derived tree structures and linearize the nodes for comparison. This worked for simple sentence structures but did not scale up to more complex sentences. We continue to work on a viable solution for this problem. It may be that we are facing a limitation inherent in our choice of the database management system approach. A more recursive-based implementation strategy may be necessary.

One motivation of this work is to provide a tool for verification of smaller, domain specific grammars that may be subsets of larger grammars, such as XTAG. We simulated such a grammar by extracting a subset of XTAG trees and applying our verification tool to this grammar. Since the tree subset was randomly chosen without linguistic significance, we expected our verification tool to identify gaps in the grammar. Our verification tool reported several defects in the grammar including missing trees for substitution nodes and unused elementary trees. Working with a subset of 105 trees from XTAG, our system was able to identify 5 duplicate tree structures, 7 missing trees for substitution nodes and one superfluous tree. The complete verification process on this subset of 105 trees with an average of 12 nodes per tree took less than 20 seconds.

We expanded our test subset to simulate additional grammar errors. Duplicate tree structures were identified when node features were ignored. Extracted tree structures were manually changed to generate structural errors. Tree nodes were added to produce recursive phrases. Trees needed for sentences with multiple parses were selected for the grammar subset.

Our verification tool successfully identified all grammar defects with this handcrafted grammar subset. While some implementation issues remain for large grammars, we have shown that stand-alone grammar verification can be a useful initial test strategy in a complete NLP structural test plan. Grammar errors can be identified and corrected at their source, before the grammar is embedded in a component application. This improves grammar reliability so subsequent test efforts may focus on coverage and application defect issues.

7 Conclusions and Future Work

The set of structural and relational checks we have described can serve as the first stage of verification analysis for an LTAG. At present we have a stand-alone system, easily usable by an NLG researcher, that will convert a grammar into the DBMS format and perform the LTAG verification checks. More experimentation needs to be done to determine how the static identification of grammar errors affects the overall system development process and the quality of the final system. In addition, as these grammar checks do not guarantee any kind of semantic coherence, we are presently extending our approach to feature-based LTAGs, where elements of semantic coherence are enforced within the structure of the grammar components, so that a verified grammar is more likely to generate semantically coherent sentences. Much work remains to address the larger issues of resource verification, verification of generation tasks, and the application of structural testing to language processing systems. Finally, we plan to apply our verification approach to more complex grammars, including one that generates text combined with gestures for an embodied conversational agent.

Acknowledgments

The authors thank Bonnie Webber and Matthew Stone for their continued interest in and suggestions about this work.

References

Srinivas Bangalore, Anoop Sarkar, Christine Doran, and Beth Ann Hockey. 1998. Grammar and parser evaluation in the xtag project. In *Proceedings of the Workshop on Evaluation of Parsing Systems*, Granada,Spain. Language Resources and Evaluation Conference.

Valerie Barr and Judith Klavans. 2001. Verification and validation of language processing systems: Is it evaluation? In *Proceedings of the Workshop on Evaluation Methodologies for Language and Dialogue Systems, ACL2001*, Toulouse, France. Association of Computational Linguists.

Valerie Barr. 1999. Applications of rule-base coverage measures to expert system evaluation. *Journal of Knowledge Based Systems*, 12:27–35.

Valerie Barr. 2003. A proposed model for effective verification of natural language generation systems. In *Proceedings of Florida Artificial Intelligence Research Symposium 2003*, Saint Augustine, FL.

Boris Beizer. 1990. *Software Testing Techniques*. Van Nostrand Reinhold, New York.

Robert Dale and Chris Mellish. 1998. Towards evaluation in natural language generation. In *Proceedings*

of the 1st International Conference on Language Resources and Evaluation, Granada, Spain, May.

C. Doran, D. Egedi, B.A.Hockey, B. Srinivas, and M. Zaidel. 1994. Xtag system - a wide coverage grammar for english. In *Proceedings of the International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan.

Christine Doran, Beth Hockey, Philip Hopely, Joseph Rosenzweig, Anoop Sarkar, B. Srinivas, Fei Xia, Alexis Nasr, and Owen Rambow. 1997. Maintaining the forest and burning out the underbrush in xtag. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Language Engineering (ENVGRAM)*, Madrid, Spain. Association of Computational Linguists.

Avelino Gonzalez and Valerie Barr. 2000. Validation and verification of intelligent systems - what are they and how are they different? *Journal of Experimental and Theoretical Artificial Intelligence*, 12(4), October.

A. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10:136–163.

Aravind Joshi. 1987. An introduction to tree adjoining grammars. In A. ManasterRamer, editor, *Mathematics of Language*, pages 87–113. John Benjamins, Amsterdam.

Daniel Jurafsky and James Martin. 2000. *Speech and Language Processing*. Prentice-Hall, New Jersey.

Alexandra Kinyon and Carlos A. Prolo. 2002. A classification of grammar development strategies. In *Proceedings of the Workshop on Grammar Engineering and Evaluation*, pages 43–49, Taipei, Taiwan.

Rashmi Prasad and Anoop Sarkar. 2000. Comparing test-suite based evaluation and corpus-based evaluation of a wide-coverage grammar for english. In *Using Evaluation within Human Language Technology Programs: Results and Trends*, Athens, Greece. LREC 2000 Satellite Workshop.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK.

Anoop Sarkar and Shuly Wintner. 1999. Typing as a means for validating feature structures. In *Proceedings of Computational Linguistics in The Netherlands 1999 (CLIN99)*, Utrecht. CLIN.

Jeffrey M. Voas and Keith W. Miller. 1995. Software testability: The new verification. *IEEE Software*, 12(3):17–28.

Bonnie Webber, Claire Gardent, and Johan Bos. 2002. Position statement: Inference in question answering. In *LREC'02 Workshop on Question Answering: Strategy and Resources*. Las Palmas.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Metagrammars: a new implementation for FTAG

Sébastien Barrier, Nicolas Barrier
Laboratoire de linguistique formelle
Case Postale 7031
2, place Jussieu 75251 Paris Cedex 05, France
{sebastien,nicolas}.barrier@linguist.jussieu.fr

Abstract

This paper describes work on creating elementary trees for adjective and predicative noun families (Barrier, 2002; Barrier and Barrier, 2003) using Metagrammars, for the FTAG grammar (Abeillé, 1991; Abeillé, 2002). Based on the Candito's work on Metagrammars (Candito, 1996; Candito, 1999a), it adds a fourth dimension, specially designed for word order specification.

1 The metagrammar compiler

Metagrammars represent a TAG as a multiple inheritance network, whose classes specify syntactic properties. An important aspect of classes is that they are all related to one another. Inheritance enables classes that are logically related to one another to share the behaviors and attributes that they have in common.

Our metagrammar imposes an overall organization for syntactic data and formelizes the well-formedness conditions on elementary tree sketches (Vijay-Shanker and Schabes, 1992; Rogers and Vijay-Shanker, 1994).

Each syntactic property of the hand-written inheritance network – the hierarchy – is declared as a complete syntactic set of partial descriptions. Those partial descriptions can be seen as syntactic constraints (dominance, linear precedence, ...) which may leave underspecified the relation between two nodes – the relation can be further explained by adding constraints in sub-classes of the network.

In concrete terms, data are defined as global variables augmented with specific meta-features, constraining for instance the possible part of speech of a node, or function for argument ones.

Structures sharing the same initial subcategorization frame may only differ in the surface realization of the final syntactic function of the arguments nodes, according to their redistribution.

The hand-written hierarchy was initially divided into 3 dimensions, and has been more recently extended to 4 dimensions (Barrier and Barrier, 2003):

- Dimension 1 : initial subcategorization.
- Dimension 2 : redistribution of functions.
- Dimension 3 : Surface realizations of syntactic functions.
- Dimension 4 : word order specification of surface realizations of syntactic functions.

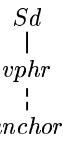
Contrary to (Vijay-Shanker and Schabes, 1992), we do not have explicit lexical rules: diathesis alternations are represented by classes of dimension 2, whereas marked and unmarked cases are represented by classes of dimension 3. Dimension 4 allows to express word order in a directly readable and not confusing way: classes of dimension 1 and 2 were clearly inappropriate (word order has nothing to deal with declaration of grammatical functions), whereas classes of dimension 3 couldn't predict the existence or the lack of another argument.

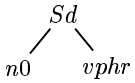
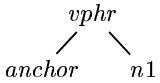
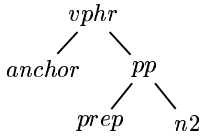
In order to automatically generate elementary trees, the compiler creates additional classes, named "crossing-classes". Each crossing class inherits from one class of dimension 1, then inherits from one class of dimension 2, and lastely inherits from classes of dimension 3, representing the realizations of every function of the final subcategorization. Classes of dimension 4 are not crossed automatically: all the crossings are declared manually by the metagrammar's writer so that he can only express the crossings, which are necessary. Crossings are accordingly only done when all the relevant classes are involved.

Finally each crossing class is translated into one or more elementary trees, satisfying all inherited constraints.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 132-139.

Dimension 1		
The class (DI-TRANS) inherits from (SUBJ), (OBJ) and (IND-OBJ)		
(SUBJ) Class	(OBJ) Class	(IND-OBJ) Class
Variable $arg0$ stands for $NP_0 \downarrow$ and bears Subject function	Variable $arg1$ stands for $NP_1 \downarrow$ and bears Object function	Variable $arg2$ stands for PP_2 and bears Indirect Object function

Dimension 2
The class (NO-REDIS) inherits from (VB-MORPH)
(VB-MORPH) Class
Variable Sd stands for S Variable $vphr$ stands for VP Variable $anchor$ stands for $V\diamond$ 

Dimension 3		
The class (SUBJ-CAN) inherits from (POS-SUBJ) (POS-SUBJ) allows to group all the realizations of the Subject	The class (OBJ-CAN) inherits from (POS-OBJ) (POS-OBJ) allows to group all the realizations of the Object	The class (IND-OBJ-CAN) inherits from (POS-IO) (POS-IO) allows to group all the realizations of the Indirect Object
(SUBJ-CAN) Class	(OBJ-CAN) Class	(IND-OBJ-CAN) Class
Variable $n0$ bears Subject function 	Variable $n1$ bears Object function 	Variable pp bears Indirect Object function Variable $Prep$ stands for $to\diamond$ Variable $n2$ stands for $NP_2 \downarrow$ 

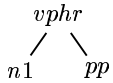
Dimension 4
(OBJ<IO) Class
This class will be used when both (OBJ-CAN) and (IND-OBJ-CAN) will appear


Table 1: Verbal hierarchy for di-transitive verbs

An inheritance hierarchy such as the one shown in Table 1, allows to represent the relevant tree sketch for the english sentence *Max gives a book to Peter*. It will be compiled out of an initial subcategorization with subject, direct object and indirect object (dimension 1), an active canonical redistribution (dimension2), canonical realizations of subject, direct object and indirect object (dimension 3), and a special word order, specifying indirect object follows direct object (dimension 4).

The compiler will automatically cross (DI-TRANS), (NO-REDIS), (SUBJ-CAN), (OBJ-CAN) and (IND-OBJ-CAN) classes. As (OBJ-CAN) and (IND-OBJ-CAN) are crossed, (OBJ<IO) will also be crossed with the other classes. The resulting tree sketch will be the conjunction of all quasi-tree descriptions contained in each class. The nodes with same variables will unify; the variables with same function will also unfify.

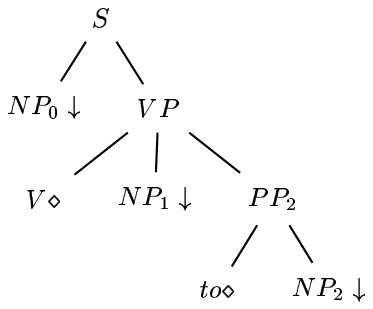


Figure 1: Elementary tree for *Mary gives a book to Peter*

Note that the metagrammar compiler makes use of variables as global variables. There is no way to use local variables. Linear precedence can't be expressed without reference to dominance.

The Metagrammar compiler we use was first developed by (Candito, 1999a) in Lucid Common Lisp and has been in part reimplemented in CLISP by (Barrier, 2002). It generates tree sketches in both XTAG or TAGML2 format with t-feature structures (see below).

2 Choices and implementation

2.1 Linguistics principles and general choices

As mentionned in (Abeillé et al., 2000), FTAG elementary trees respect the following well-formedness principles :

- Strict lexicalization: all elementary trees are anchored by at least one lexical element (the empty string cannot anchor a tree by itself)
- Semantic consistency: no elementary tree is semantically void
- Semantic minimality: elementary trees correspond to no more than one semantic unit
- Predicate argument cooccurence principle : an elementary tree is the minimal syntactic structure that includes a leaf node for each realized semantic argument of the anchor(s)

Semantic minimality and consistency imply that function words appear as co-anchors.

Most of the linguistic analyses follow those of (Abeillé, 1991; Abeillé, 2002) (except that clitic arguments are substituted and not adjoined), complemented by (Candito, 1999a). We dispense with most empty categories, especially in the case of extraction. Semantically void (or non autonomous) elements, such as complementizers, argument marking prepositions or idiom chunks are co-anchors in the elementary tree of their governing predicate.

Passive is characterized by a particular morphology, with a substitution node for the auxiliary verb. Causative constructions are analyzed as complex predicates, with a flat structure, with a substitution node for the causative verb.

For oblique complements, we distinguish between a-objects, de-objects, locatives and other prep-objects, depending on the pronominal realization of the complement.

2.2 New families for FTAG

We have chosen not to reuse Candito's verbal hierarchy because of inconsistencies: it was not fully documented and hard to understand. Some classes of dimension 3 inherit from classes of dimension 1 or 2, which is normally not allowed by the metagrammar concept. Furthermore, this verbal hierarchy contains some empty classes.

We developed 34 new families: 16 adjectival families allow us to create 2690 tree sketches, whereas 18 support verb families allow us to create over 10.000 tree sketches.

2.2.1 Adjectival families

We regard the adjective as the local head of the adjectival predicate, and consider object predicate's constructions as an alternative of causative constructions. An unique family provides tree sketches for both predicative and attibutive adjectives, so that we can encode relative clauses or clitics for different kind of adjective complements. We describe the concept of subject as the category modified by the adjective. No object function can be found: all the complements of the adjectival predicate are always indirect ones.

Our grammar covers the following types of redistribution :

- Predicative adjective : Jean est barbu

- Causative : Sarah Vaughan rend les gens heureux
- Passive causative : Des gens sont rendus heureux
- Impersonal causative passive : Il est rendu impossible de faire cela
- Impersonal : Il est inacceptable de dormir ici
- Attributive adjective : Un homme heureux

The syntactic realizations covered are canonical position, extraction (cleft and relativized), clitic or non-realized.

2.2.2 Predicative noun families

The lexical head is only the predicative noun, whereas the support verb is substituted into the tree associated with the noun. This differs from the light verb families from XTAG (and also from the previous versions of FTAG) where the verb and the noun both anchor the tree. An unique family provides tree sketches for support verb constructions and nominal phrases.

Our grammar covers the following types of redistribution :

- Active: Max commet un crime contre Luc
- Passive: Un crime est commis par Max contre Luc
- Middle: Un crime se commet contre Luc en 5 minutes
- Causative: Léa fait commettre une crime à Max contre Luc
- Passive Impersonal: Il est commis un crime par Max contre Luc
- Impersonal Middle: il se commet un crime toutes les 5 minutes
- Nominal phrase: le crime de Max contre Luc

The syntactic realizations covered are canonical position, extraction (cleft, relativized and questionned), clitic and non-realized.

Datasheet for adjective and predicative noun hierarchies can be found at the end of this article. Each page represents Dimension 1, 2 and 3. Dimension 4 is not shown since it is not particular to these hierarchies. It is specially used for clitic word order.

2.3 Main difficulties

A typical error consists in encoding more than a class expects. One may de facto limit the syntactic properties sharing. Metagrammars do not exempt from studying syntactic phenomena but force ones to understand what classes share with in terms of syntactic properties. Since arguments are realized as independent functions the metagrammar's writer not only has to find a way to arrange them correctly inside the tree but has to encode his classes so that they can be reused for another category.

Another place metagrammars and inheritance networks go wild is in making very deep hierarchy. It can be very tedious to look many levels up to the tree to find out what a particular inherited variable is supposed to be: it is easy to create complex hierarchy that is hard to understand, even for the metagrammer's writer who created it. Inheritance, just like many other elements of OOP is just a tool. If the problem calls for it, it seems interesting to use it, but one doesn't see it as a solution to all problems. With proper usage, metagrammars will save the writer from retyping and will show him that different linguistic objects are related.

3 Current and future work

To take advantage of the hierarchical representation of tree sketches within our metagrammar, we characterize tree sketches as feature structures we call t-feature structures (Abeillé et al., 1999).

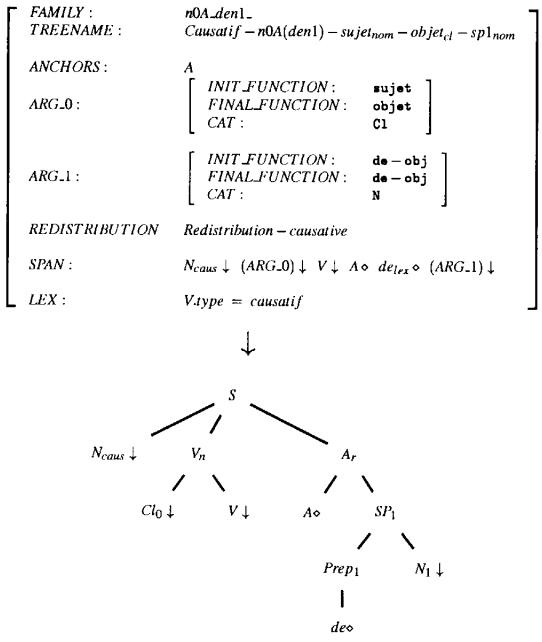
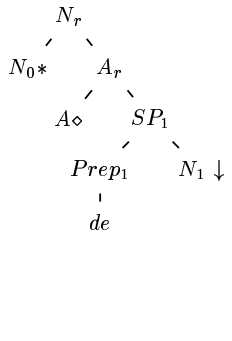
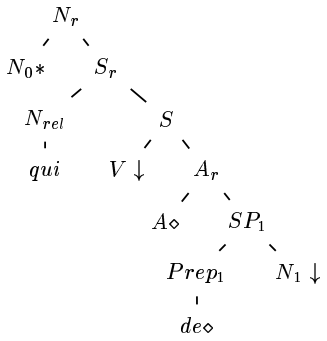


Figure 2: Tree sketch for a causative construction used for an adjectival predicate

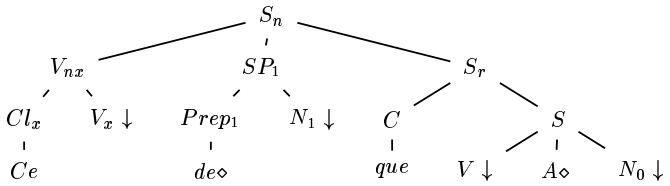
While the automatic generation of the grammar insures consistency, errors may still propagate but on a larger scale, with dramatic effects if it remains undetected. These feature-structures keep track of the successive mapping steps that are performed during the genera-



Un homme fier de sa fille
A man proud of his daughter

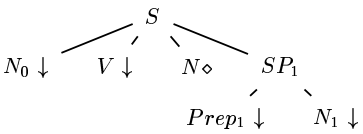


Max qui est fier de sa fille
Max who is proud of his daughter

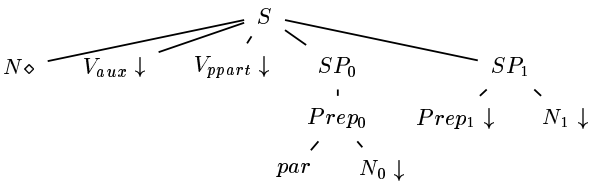


C'est de sa fille qu'est fier Max
It is of his daughter that Max is proud

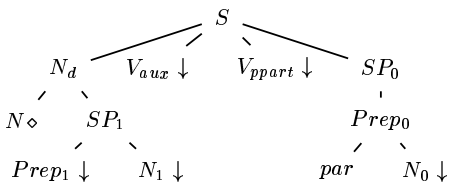
Table 2: Some elementary trees taken from *n0A(den1)* family



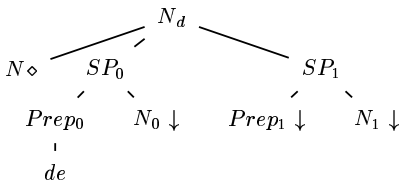
Max commet un crime contre Luc
Max commits a crime against Luc



Un crime est commis par Max contre Luc
A crime is commited by Max against Luc



Un crime contre Luc est commis par Max
A crime against Luc is committed by Max



Le crime de Max contre Luc
Max's crime against Luc

Table 3: Some elementary trees taken from the $n0vN(pn1)$ family

tion process.

Characterizing tree sketches as a combination of features allows us to refer to a set of tree sketches simply by under specifying a feature structure.

It could also be interesting to merge all the hierachies into one. But this will probably be a hard task[1]. Each Metagrammar's writer has indeed his own view of specific problems.

We hope to evaluate our grammar in few weeks by using treebank 'Le Monde' developed at Paris 7 University (Abeillé et al., 2003).

References

Anne Abeillé and Owen Rambow. 2000. *Tree Adjoining grammars*. CSLI Publications.

Anne Abeillé, Marie-Hélène Candito, and Alexandra Kinyon. 1999. Ftag: current status and parsing scheme. In *Vextal'99*.

Anne Abeillé, Marie-Hélène Candito, and Alexandra Kinyon. 2000. The current status of FTAG. In *Proceedings of TAG+5*.

Anne Abeillé, Nicolas Barrier, and Sébastien Barrier. 2001. FTAG, une grammaire LTAG du français. Technical Report 1.0.

Anne Abeillé. 1991. *Une grammaire lexicalisée d'arbres adjoints pour le français*. Ph.D. thesis, Université Paris 7.

Anne Abeillé. 2002. *Une grammaire électronique du français*. CNRS Editions.

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In A. Abeillé, editor, *Treebanks: building and using parsed corpora*, pages 165–188. Kluwer academic publishers.

Sébastien Barrier and Nicolas Barrier. 2003. Une métagrammaire pour les noms prédicatifs du français. In *TALN 2003*.

Nicolas Barrier. 2002. Une métagrammaire pour les adjectifs du français. In *TALN 2002*.

Davy Boonen. 2001. Le prédicat adjectival en ftag. Master's thesis, Université Paris 7.

Marie-Hélène Candito. 1996. A principle-based hierarchical representation of ltag. In *Proceedings* 15^{th} *COLING*.

Marie-Hélène Candito. 1999a. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées, application au français et à l'italien*. Ph.D. thesis, Université Paris 7.

[1]Of course, it does not mean all the new tree sketches cannot be combined into one grammar.

Marie-Hélène Candito. 1999b. Un outil multilingue de construction semi-automatique de grammaire d'arbres adjoints. In *TAL*, volume 40.

Laurence Danlos. 1992. Support verb constructions. In *Journal of French Linguistic Study*.

Laurence Danlos. 1998. GTAG, un formalisme lexicalisé pour la génération inspiré de TAG. In *TAL*, volume 39.2.

Bertrand Gaiffe, Benoît Crabbé, and Azim Roussanaly. 2002. A new metagrammar compiler. In *Proceedings of TAG+6*.

Kim Gerdes. 2002. *Topologie et grammaires formelles de l'allemand*. Ph.D. thesis, Université Paris 7.

Jacqueline Giry-Schneider. 1978. *Les nominalisations en français*. Droz. Genève-Paris.

Jacqueline Giry-Schneider. 1987. *Les prédicats nominaux en français*. Droz. Genève-Paris.

Jan Goes. 1999. *L'adjectif entre nom et verbe*. Duculot.

Maurice Gross. 1981. Les bases empiriques de la notion de prédicat sémantique. In *Langages*, volume 63.

Zellig Harris. 1968. *Mathematical structures of language*. Wiley-Interscience.

Alexandra Kinyon. 2000. Even better than supertags: introducing hypertags. In *Proceedings of TAG+5*.

Michèle Noailly. 1990. *Le substantif épithète*. PUF. Paris.

Michèle Noailly. 1999. *L'adjectif en français*. OPHRYS. Paris.

J. Rogers and K. Vijay-Shanker. 1994. Obtaining trees from their descriptions, an application to tree adjoining grammar. In *Computational intelligence*, volume 10.4.

K. Vijay-Shanker and Y. Schabes. 1992. Structure sharing in lexicalized tree adjoining grammar. In *Proceedings of COLING-92*.

Annexe A - Datasheet for Adjectives

Family	Example	Family	Example
n0A	*Jean est barbu* John is bearded	n0A(as1)	*Jean est attentif à ne blesser personne* John is cautious not to hurt anyone
n0A(pn1)	*Jean est fort en histoire* John is good at history	n0A(des1)	*Jean est certain qu'ils viendront* John is convinced they will come
n0A(an1)	*Jean est sourd à cette proposition* John is deaf to this proposal	n0A(an1)(des2)	*Jean est reconnaissant à Marie de faire ses devoirs* John thanks Mary for doing his homework
n0A(den1)	*Jean est amoureux de Marie* John is in love with Mary	s0A	*Prendre le thé sur la pelouse est inacceptable* Having tea out on the lawn is unacceptable
n0A(an1)(pn2)	*Jean est supérieur à Marie en histoire* John is higher than Mary at history	s0A(pn1)	*Prendre le thé est bon pour la santé* Having tea is good for health
n01(an1)(den2)	*Jean est redevable de 10€ à Marie* John owes Mary 10€	s0A(ps1)	*Faire du sport est bon pour éviter les crises cardiaques* Doing sport is good to prevent heart attacks
n0A(den1)(pn2)	*Jean est quitte de ses dettes envers la société* John has paid his debt to society	s0A(an1)	*Prendre le thé est nécessaire aux hommes* Having tea is necessary to men
n0A(ps1)	*Boire du thé est bon pour le mal de tête* Having tea is good for headaches	s0A(den1)	*Faire du sport est indépendant de vos autres activités* Doing sport is independant from your other activities

Table 4: Adjectival families

	Initial subject				
Construction	N	Cl	S	Redistribution	Example
Predicative adjective	+	+	+	No redistribution	Jean est barbu
Causative	+	+	-	Subject > Object Causer > Subject	Sarah Vaughan rend les gens heureux
Passive causative	+	+	+	Causer > Par_obj Object > Subject	Des gens sont rendus heureux (par Sarah)
Impersonal causative passive	+	+	+	Causer > empty Impersonnal > Subject	Il est rendu impossible de faire cela
Attributive adjective	+	-	-	Subject > Subject_epi	Un homme heureux
Impersonal	-	-	+	Subject > Sentencial indirect cmpl Impersonal > Subject	Il est inacceptable de commettre des erreurs

Table 5: Redistribution frame for adjectives

	Surface realizations					
	Nominal	Clitic	Cleft	Sentencial	Relativized	Non-realized
Subject	Canonical Inverted	X	Nominal Sentencial	X	qui	
Prep-obj	X		Nominal Sentencial	X	X	X
A-obj	X	X	Nominal Sentencial	X	X	X
De-obj	X	X	Nominal Sentencial	X	dont	X
Prep-obj2	X		Nominal		X	X
De-obj2		X	Nominal	X	dont	X
Indirect Sentencial cmpl				X		
Predicative object	Anteposed Postposed	X				
Par-Obj	X					X

Table 6: Surface realization of syntactic functions for adjectives

Annexe B - Datasheet for Predicative Nouns

Family	Example	Family	Example
n0vN	*Max prend un bain* Max takes a bath	n0vPN(as1)	*Max a de la peine a dormir* Max has difficulty in sleeping
n0vN(an1)	*Max fait du chantage à Luc* Max blackmails Luc	s0vN	*Prendre le thé sur la pelouse fait scandale* Having tea out on the lawn scandalized people
n0vN(den1)	*Max fait la censure de cette page* Max censors this page	s0vN(den1)	*Prendre le thé sur la pelouse fait la joie de Luc* Having tea out on the lawn gives great pleasure to Luc
n0vN(loc1)	*Max fait un pélerinage à Lourdes* Max goes on a pilgrimage to Lourdes	s0vPN(den1)	*Faire du sport est à l'avantage de Max* Doing sport gives an advantage to Max
n0vN(pn1)	*Max commet un crime contre Luc* Max commits a crime against Luc	n0vN(den1)(an2)	*Max fait le récit de son histoire à Luc* Max gives an account of his story to Luc
n0vN(des1)	*Max a l'espoir de réussir* Max hopes he will succeed	n0vN(den1)(pn2)	*Max fait la division de 4 par 2* Max divides 4 by 2
n0vN(ps1)	*Max fait un effort pour rester calme* Max makes an effort to stay calm	n0vN(den1)(loc2)	*Mac fait une expédition de livres en Somalie* Max send books in Somalia
n0vPN(pn1)	*Max est en colère contre Luc* Max is angry with Lux	n0vN(pn1)(pn2)	*Max fait une plaisanterie sur Luc avec Léa* Max makes a joke with Léa on Luc
n0vPN(den1)	*Max est dans l'ignorance de cet incident* Max is unaware of this event	n0vN(an1)(des2)	*Max a donné l'ordre à Luc de partir* Max has ordered Luc to go

Table 7: Predicative nouns families

Construction	Redistribution	Example
Passive	object > subject subject > par_object	Un crime est commis par Max contre Luc Un crime contre Luc est commis par Max
Middle	subject > empty object > subject	Un crime se commet contre Luc en 5 minutes Un crime contre Luc se commet en 5 minutes
Causative-A	subject > empty causer > subject	Léa fait commettre un crime à Max contre Luc
Impersonal Middle	subject > empty Impers > subject	Il se commet un crime toutes les 5 minutes
Impersonal Passive	subject > par_object impers > subject	Il est commis un crime par Max contre Luc Il est commis un crime contre Luc par Max
Nominal phrase	object > empty prep_object > cdn	Le crime de Max contre Luc

Table 8: Redistribution frame for predicative nouns

	Surface realizations						
	Nominal	*Clitic*	*Cleft*	*Sentencial*	*Relativized*	*Questionned*	*Non-realized*
Subject	Canonical Inverted	X	Nominal	X	qui	X	
Predicative Noun	X		Nominal		que		
Prep Obj	X		Nominal	X	X	X	X
A-Obj	X	X	Nominal		X	X	X
De-obj	X	X	Nominal		dont	X	X
Prep-Obj2	X		Nominal		X	X	X
A-Obj2	X	X	Nominal		X	X	X
Indirect sentencial cmpl				X			
Par-Obj	X		Nominal			X	X

Table 9: Surface realization of syntactic functions for predicative nouns

Sentences with two subordinate clauses: syntactic and semantic analyses, underspecified semantic representation

Laurence Danlos
TALANA/ LATTICE
Université Paris 7
Laurence.Danlos@linguist.jussieu.fr

Abstract

I show that sentences with two subordinate clauses may receive two syntactic analyses, and that each syntactic analysis may receive two semantic interpretations. Hence, I put forward an underspecified semantic representation such that each syntactic analysis receives only one underspecified interpretation.

1 Introduction

Sentences with two subordinate clauses occur quite often in corpora. Theories and tools in Computational Linguistics are available now which allow us to study such sentences exhaustively, both at the syntactic and semantic level. It is what I intend to do in this paper, while using only well-known techniques.

Several sophisticated theories and discourse processing mechanisms have been designed which put forward a number of *principles*. This study on sentences with two subordinate clauses, which constitute one of the simplest cases of discourses, will question some of these principles (e.g., semantic dependency structures for discourses are tree shaped, discourse structure does not admit crossing structural dependencies). It therefore sheds light on discourse processing in general.

Section 2 focuses on the syntactic analysis of sentences with one or two subordinate clauses, including their linear order variants. The syntactic framework I use is LTAG. I show that sentences with two subordinate clauses may receive two syntactic analyses. Section 3 focuses on the semantic analysis of such sentences. The semantic framework I use is SDRT[1], although I translate the conditions of an SDRS into a dependency graph. I show that sentences with two subordinate clauses may receive four semantic dependency structures. Section 4 studies the mapping between syntax and semantics and shows that each syntactic analysis for sentences with two subordinate clauses receives two semantic interpretations. Hence the need of an underspecified semantic representation (henceforth USR). Section 5 presents this USR. Finally, Section 6 compares this work with D-LTAG (Webber et al., 2003).

[1] SDRT stands for Segmented Discourse Representation Theory (Asher and Lascarides, 2003). It is an extension of DRT, Discourse Representation Theory (Kamp and Reyle, 1993). (S)DRS stands for (Segmented) Discourse Representation Structure.

2 Syntax (in LTAG)

2.1 Sentences with one subordinate clause

Syntactically, subordinate clauses are adjuncts. Therefore in XTAG (XTAG Research Group, 2001) and FTAG (Abeillé et al., 2000), the English and French LTAG grammars, a subordinate conjunction (*Conj*) anchors an auxiliary tree, with two syntactic sentential (clausal) arguments, the foot node for the matrix clause and a substitution node for the subordinate clause.

Both in English and French, a subordinate clause may appear in three different positions relative to the matrix clause: (i) before the matrix clause separated by a punctuation mark (a comma), the linear order is then *Conj$_a$ S2, S1*, (ii) before the VP surrounded by two commas, and (iii) after the matrix clause optionally separated by a punctuation mark, the linear order is then *S1 (,) Conj$_a$ S2*. Therefore, in FTAG, a given subordinating conjunction anchors three auxiliary trees which correspond to these three positions. This is not the case in XTAG, where it anchors four auxiliary trees, two of them for the sentence final position: sentence final adjuncts without comma adjoin at a VP node, while those with a comma adjoin at the root S of the matrix clause. Let us quote (XTAG Research Group, 2001) p. 152 for the former ones. "One compelling argument is based on Binding Condition C effects. As can be seen from examples (1a-c) below, no Binding Condition violation occurs when the adjunct is sentence initial, but the subject of the matrix clause clearly governs the adjunct clause when it is in sentence final position and co-indexation of the pronoun with the subject of the adjunct clause is impossible."

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 140-147.

(1) a. Unless she$_i$ hurries, Mary$_i$ will be late for the meeting.
b. *She$_i$ will be late for the meeting unless Mary$_i$ hurries.
c. Mary$_i$ will be late for the meeting unless she$_i$ hurries.

I agree with the data observed in (1a-c), however my point is that there would be no difference at all if the sentence final adjunct in (1b-c) were separated by a comma. Therefore, I see no reason to lay down two different trees - one adjoining at a VP node, the other one at a S node - for sentence final adjuncts with or without a comma. As in FTAG, I assume only one tree (with an optional comma) for sentence final adjuncts, which adjoins at the root S of the matrix clause. This solution presents the advantage not to rely heavily on the presence or absence of a comma, which is sometimes a matter of taste, as in (1c).

Because of lack of room, I leave aside subordinate clauses which appear before the VP of the matrix clause. To put it in a nutshell, I consider only two auxiliary trees for a subordinating conjunction *Conj*$_a$ according to the linear order: β1(Conj$_a$) when the subordinate clause is postposed to the matrix clause, and β2(Conj$_a$) when it is preposed. Figure 1 shows the auxiliary and derivation trees for postposed and preposed subordinate clauses[2].

2.2 Sentences with two subordinate clauses

A sentence *S1 Conj*$_a$ *S2 Conj*$_b$ *S3* receives two syntactic analyses. In the first one, obtained by recursivity and noted SA1, *Conj*$_a$ *S2* is an adjunct to *S1* and *Conj*$_b$ *S3* an adjunct to *S2*. In the second one, obtained by adjunct iteration and noted SA2, both *Conj*$_a$ *S2* and *Conj*$_b$ *S3* are adjuncts to *S1*. Figure 2 shows the derived and derivation trees for these two analyses. For SA2, the derivation tree is based on multiple adjunctions to the same node, as proposed in (Schabes and Shieber, 1994). These multiple adjunctions are ordered (from left to right). The syntactic ambiguity of sentences *S1 Conj*$_a$ *S2 Conj*$_b$ *S3* is systematic without any comma. On the other hand, sentences *S1 Conj*$_a$ *S2, Conj*$_b$ *S3* with a comma before the second conjunction are preferably analyzed as SA2.

Let us examine the possible variants of the canonical linear order *S1 Conj*$_a$ *S2 Conj*$_b$ *S3*, which corresponds to the case where both *Conj*$_a$ and *Conj*$_b$ anchor a postposed tree. For each analysis, it must be examined what happens (a) when *Conj*$_a$ anchors the preposed tree β2(Conj$_a$) and *Conj*$_b$ the postposed one β1(Conj$_b$), (b) symmetrically, when *Conj*$_a$ anchors β1(Conj$_a$) and *Conj*$_b$ β2(Conj$_b$), (c) and finally when both *Conj*$_a$ and *Conj*$_b$ anchor a preposed tree. Figure 3 shows the linear orders for SA1 and SA2 other than the canonical one.

Consider the syntactic ambiguity issue for these variants. (a2) and (b2) in Figure 3 are both of the form *Conj S, S Conj S.*, with a preposed and a postposed adjunct. Therefore any sentence of this form receives two syntactic analyses and corresponds to two sentences in the canonical order (Section 3.2). The variants (a1) and (c2) are both of the form *Conj S (,) Conj S, S.* The comma before the second conjunction is obligatory in (c2) and nearly forbidden in (a1). Therefore, these forms are nearly unambiguous. The variants (b1) and (c1) correspond to sentences which are syntactically unambiguous.

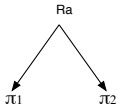
3 Semantics

3.1 Sentences with one subordinate clause

Following works in SDRT, I use an intermediate level of representation to determine the logical form of a discourse (*what is said*). This "semantic" level reflects the discourse structure (*how things are said, how the discourse is rhetorically organized*). This structure plays an important role, e.g., it constrains both anaphora resolution and the attachment of incoming propositions in understanding.

A nice tool for the semantic level is dependency graphs. This is what is adopted in RST[3], but not in SDRT: discourse structures, called SDRSs, are represented as boxes with a Universe and a set of conditions. Nevertheless, it is easy to translate the conditions of an SDRS into a dependency graph (Danlos, 2004). Therefore, while adopting SDRT as a discourse framework, I can use a conventional semantic dependency representation for sentences of the type *S1(,) Conj*$_a$ *S2*. Namely, *Conj*$_a$ denotes a discourse relation R$_a$. R$_a$ is a predicate with two arguments π1 and π2, which correspond to the semantic representations of *S1* and *S2* respectively. These arguments are ordered: π1 precedes π2.

This semantic representation is graphically represented in the DAG besides, also simply written as R$_a(\pi 1, \pi 2)$.



3.2 Semantics for linear order variants

We have seen in Section 2.1 that a subordinate clause can be postposed or preposed. Following works in MTT[4], a trace of the linear order should be recorded in a semantic dependency representation (giving so a piece of information on the communicative structure), however it should

[2] In a derivation tree, a dashed line indicates adjunction, a solid line substitution; each line is labeled by the Gorn address of the argument at which the operation occurs; αi stands for the LTAG tree for *Si*.

[3] RST stands for Rhetorical Structure Theory (Mann and Thompson, 1987). Rhetorical structures correspond roughly to dependency structures.

[4] MTT stands for Meaning to Text Theory, a dependency formalism for sentences (Mel'cuk, 2001).

not affect its dependency structure. From this principle, the position of subordinate clauses should not affect dependency structures: *S1(,) Conj$_a$ S2* and *Conj$_a$ S2, S1* are both represented as $R_a(\pi_1, \pi_2)$ in which π_1 precedes π_2.

What happens for a sentence with two subordinate clauses? Establishing the canonical order with only postposed subordinate clauses may generate ambiguities: for example, a sentence X of the type *Conj$_a$ S1, S2 Conj$_b$ S3*, with a preposed and a postposed adjunct, corresponds in the canonical order either to Y_1 = *S2 Conj$_a$ S1 Conj$_b$ S3* or to Y_2 = *S2 Conj$_b$ S3 Conj$_a$ S1*. X receives two syntactic analyses: either (a2) - Figure 3 - from Y_1 (the first adjunct is preposed), or (b2) from Y_2 (the second adjunct is preposed). These analyses allow us to compute Y_1 and Y_2. From the above principle that the position of subordinate clauses does not affect dependency structures, X does not yield any other DAGs than Y_1 and Y_2. As a consequence, our study on the dependency structures of sentences with two subordinate clauses can be limited to the study of such sentences in the canonical order.

3.3 Sentences with two subordinate clauses

We are going to show that sentences with two subordinate clauses may be interpreted in four different ways. Two interpretations are found in which one conjunction has wide scope over the other one, two other ones without wide scope. The former are represented in tree shaped DAGs, the latter in non tree shaped DAGs.

This semantics study is based on the following compositionality principle. Let $\mathcal{D}_n$ be a DAG with n leaves representing the dependency structure of a discourse D_n. If $\mathcal{D}_p$ is a sub-DAG of $\mathcal{D}_n$ with p leaves, $1 < p < n$, then the discourse D_p corresponding to $\mathcal{D}_p$ should be inferable from D_n[5].

A) Wide scope of *Conj$_a$*: The wide scope of *Conn$_a$* = *because* in (2a) can be seen in the dialogue in (2b-c) in which the answer is *Because S2 Conn$_b$ S3*[6]. The semantic dependency structure of (2a) is DAG (A) in Figure 4. In this DAG, which is tree shaped, the dependency relations must be interpreted in the standard way used in mathematics or computer science: the second argument of R_a is its right daughter, i.e. the tree rooted at R_b which is the semantic representation of *S2 Conj$_b$ S3*. This reflects the fact that *Conj$_a$* has wide scope and is in conformity with our compositionality principle: (A) includes the sub-DAG $R_b(\pi_2, \pi_3)$ and *S2 Conn$_b$ S3* can be inferred, i.e. (2a) is true, then it is true that *Fred played tuba while Mary was taking a nap*.

(2) a. Mary is in a bad mood because Fred played tuba WHILE she was taking a nap.
b. - Why is Mary in a bad mood?
c. - Because Fred played tuba WHILE she was taking a nap.

When *while* is not stressed, the question in (2b) may be given as answer only *Because S2*. The interpretation of (2a) corresponds then to DAG (C) presented below. (2a) when written could be considered as ambiguous with a scope ambiguity of *because*. The scope of *because* is underspecified in the USR proposed in Section 5 for (2a).

B) Wide scope of *Conj$_b$*: The wide scope of *Conn$_b$* = *in order that/to* in (3a) can be seen in the dialogue in (3b-c) in which the question is *Why S1 Conj$_b$ S2?* The semantic dependency structure of (3a) is DAG (B) in Figure 4. This tree shaped DAG must be interpreted in a way similar to (A), which reflects that *Conj$_b$* has wide scope.

(3) a. Fred played tuba WHILE Mary was taking a nap in order to bother her.
b. - Why did Fred play tuba WHILE Mary was taking a nap?
c. - In order to bother her.

As for (2a), (3a) when written could be considered as ambiguous. The scope of *in order that/to* is underspecified in the USR proposed in Section 5 for (3a).

C) S2 factorized: The clause S2 in (4a) is said to be factorized since both *S1 Conj$_a$ S2 = Fred played tuba while Mary was washing her hair* and *S2 Conj$_b$ S3 = Mary was washing her hair before getting dressed for her party* can be inferred from (4a). A similar situation is observed in (4b).

(4) a. Fred played tuba while Mary was washing her hair before getting dressed for her party.
b. Fred was in a foul humor because he hadn't slept well that night because his electric blanket hadn't worked.[7]

In (4a), no conjunction has wide scope over the other one. Its semantic structure is DAG (C) in Figure 4. This DAG is not tree shaped: $\pi 2$ has two parents.

One could argue that tree shaped DAGs (A) and (B) should not be interpreted in the standard way. This is argued in RST in which dependency relations in trees are interpreted with the "nuclearity principle" (Marcu, 1996). With this principle, the arguments of a discourse relation can only be *leaves* of the tree, for example, the second argument of R_a in (A) is $\pi 2$, and the first argument of R_b in (B) is $\pi 1$. This amounts to interpreting (A) as (C), and

[5] On the other hand, the converse principle is not always true (Danlos, 2004): if a sub-discourse D_p can be inferred from D_n, it does not always mean that the DAG $\mathcal{D}_p$ is a sub-DAG of $\mathcal{D}_n$.

[6] To indicate that it is stressed when spoken, the word *while* is written in capital letters in (2).

[7] This discourse is a modified version (including *because*) of an example taken in (Blackburn and Gardent, 1998), who acknowledged that its structure is a "re-entrant graph".

(B) as (D) presented below. But then, cases with wide scope are not represented at all: they are not taken into account, which is unacceptable. As a consequence, tree shaped DAGs must be interpreted in the standard way, in which the arguments of a discourse relation may be either intermediary nodes or leaves.

It is generally assumed that semantic dependency structures for discourses should be tree shaped. As a consequence, to avoid DAGs, some authors use trees in which some predicate-argument relations are given by the nuclearity principle, while others are given by the standard interpretation. Nevertheless, one should not feel free to use trees relying on a mixed interpretation (the standard and nuclearity ones), except if the conditions governing the use of one or the other interpretation can be formally defined. In (Danlos, 2004), I show that no rule can be laid down to choose one of these two interpretations. A mixed interpretation for trees must thus be discarded. Since the standard interpretation is needed for wide scope cases, the nuclearity principle should be discarded. As another consequence, one has to admit that discourse structures for discourses are DAGs.

D) S1 factorized: The clause S1 in (5) is said to be factorized since both *S1 $Conj_a$ S2 = Fred prepared a pizza while it was raining* and *S1 $Conj_b$ S3 = Fred prepared a pizza before taking a walk* can be inferred. The semantic structure of (5) is DAG (D) in Figure 4, which is in conformity with our compositionality principle. This DAG is not tree shaped.

(5) Fred prepared a pizza, while it was raining, before taking a walk.

In discourses analyzed as (D), *S3* is linked to *S1* (which is not adjacent) and not to *S2* (which is adjacent). Therefore, these discourses are counter-examples to the adjacency principle advocated in RST.

DAG (D) exhibits crossing dependencies. It is thus a counter-example to the stipulation made by (Webber et al., 2003), namely "*discourse structure itself does not admit crossing structural dependencies*"[8].

Summary: A sentence with two subordinate clauses may receive one of the four interpretations represented in DAGs (A), (B), (C) and (D). In the next section, we will see that (A) and (C) are the interpretations of the syntactic analysis SA1, while (B) and (D) are those of SA2.

These four interpretations are the only possible ones. In particular, I cannot find any example in which *S3* would be factorized, although I wrote all possible examples I could think of and Laurence Delort, who works on (French) corpus, could not find anyone neither. The factorization of *S3* is represented as DAG (E) in Figure 5. Note that no compositional syntax-semantics rule could lead to (E) from the syntactic analyses SA1 and SA2, which are the only possible ones. More generally, in (Danlos, 2004), I show that any DAG with three ordered leaves other than (A)-(D) is excluded, i.e. does not correspond to coherent discourses with three clauses. For example, DAG (K) in Figure 5 is excluded. This comes from the "letf1-right2 principle", which is a weaker version of the adjacency principle[9].

4 Mapping between syntax and semantics

We are going to examine the interpretation(s) of the syntactic analyses put forward in Section 2. The criterion to be used is that of linear order. So, we are going to examine the linear order(s) for each interpretation (A)-(D).

A) Wide scope of *$Conj_a$*: The linear order variants of (2a), repeated in (6a), are shown in (6b-d).

(6) a. Mary is in a bad mood because Fred played tuba while she was taking a nap.
b. Because Fred played tuba while she was taking a nap, Mary is in a bad mood.
c. Mary is in a bad mood because, while she was taking a nap, Fred played tuba.
d. Because, while she was taking a nap, Fred played tuba, Mary is in a bad mood.

These linear order variants correspond to the variants which are allowed with the first analysis SA1 (see Figure 3). On the other hand, the variants which are allowed with SA2 are forbidden: the discourses in (7) do not make sense (hence the sign #).

(7) a. # Because Fred played tuba, Mary is in a bad mood while she was taking a nap.
b. # While she was taking a nap, Mary is in a bad mood because Fred played tuba.
c. # While she was taking a nap, because Fred played tuba, Mary is in a bad mood.

To conclude, interpretation (A) corresponds to SA1, or conversely SA1 can be interpreted as (A).

B) Wide scope of *$Conj_b$*: The linear order variants of (3a), repeated in (8a), are shown in (8b-d). They correspond to the variants of SA2. I leave it to the reader to

[8] Among discourse connectives, (Webber et al., 2003) distinguish "structural connectives" (e.g. subordinating conjunctions) from discourse adverbials including *then, also, otherwise*. They argue that discourse adverbials do admit crossing of predicate-argument dependencies, while structural connectives do not. I emphasize that (5) comprises only structural connectives (subordinating conjunctions) while its structure exhibits crossing structural dependencies.

[9] Recall that the adjacency principle does not hold because of examples such as (5). In sentences of the type *S1 $Conj_a$ S2 $Conj_b$ S3*, the letf1-right2 principle states that the first (resp. second) argument of a subordinating conjunction is given by a text span which occurs on its left (resp. right). This principle excludes (K) since *S1* is the only text unit on the left of *$Conj_a$*, while π_1 is not the first argument of R_a.

check that the variants of SA1 are forbidden. To conclude, interpretation (B) corresponds to SA2, or conversely SA2 can be interpreted as (B).

(8) a. Fred played tuba while Mary was taking a nap in order to bother her.
b. While Mary was taking a nap, Fred played tuba in order to bother her.
c. In order to bother Mary, Fred played tuba while she was taking a nap.
d. In order to bother Mary, while she was taking a nap, Fred played tuba.

C) S2 factorized and **D) S1 factorized**: When S2 is factorized, the linear order variants correspond to SA1, when S1 is factorized, they correspond to SA2, as the reader can check it.

Summary: A sentence with two subordinate clauses may receive the syntactic analyses SA1 and SA2, SA1 can be interpreted as (A) or (C), SA2 as (B) or (D). In the next section, I put forward underspecified semantic representations (USRs) such that the syntactic analysis SA1 receives only one underspecified semantic representation, USR1, which is specified in (A) or (C), and that SA2 receives only USR2 which is specified in (B) or (D).

5 Underspecified semantic representation

It is now classical to use USRs for quantifier scope ambiguities (among other ambiguities). Following works by (Duchier and Gardent, 2001), I adopt a scope underspecification formalism based on dominance constraints. Let us illustrate the overall idea briefly. The clause *every yogi has a guru* is represented (in a simplified way) as the "tree description" in Figure 6, in which a solid line represents immediate dominance, a dotted line dominance. Quantifier scopes are underspecified in this tree description. The dominance constraints are solved in the trees (a) and (b) in Figure 6 (in both (a) and (b), the root dominates `has(x, y)`). Quantifier scopes are specified: in (a) `forall(x)` has wide scope, in (b) it is `exists(y)`. The USR I propose for subordinate conjunctions follows this overall idea. However, it differs in two ways: (i) "left-dominance" is used instead of dominance, (ii) constraints are solved in DAGs which may be not tree shaped.

Left-dominance: It has been seen in Section 3 that the nuclearity principle is too restrictive (wide scope cases are not taken into account). It will be seen below that dominance relations are not restrictive enough. Therefore, I introduce a new relation, called "left-dominance", which is intermediary and defined as follows.

A node X in a tree left-dominates a node Y iff Y is a daughter of X (immediate dominance) or there exists a daughter Z of X such that Y belongs to the left-frontier of the tree rooted at Z.

As an illustration, R_a left-dominates $\pi 1$, R_b and $\pi 2$ in (A), while R_b left-dominates R_a, $\pi 1$ and $\pi 3$ in (B). Left-dominance is more restrictive than (strict) dominance (e.g. R_a strictly dominates $\pi 1$, R_b, $\pi 2$ and also $\pi 3$ in (A)) and less restrictive than the nuclearity principle (e.g. by this principle, R_a dominates only the leaves $\pi 1$ and $\pi 2$ in (A))[10].

Syntax to semantics: Following works in semantics with LTAG (Candito and Kahane, 1998) (Kallmeyer and Joshi, 2003), I assume that (i) each elementary tree is linked to an (underspecified) semantic representation, (ii) the way the semantic representations combine with each other depends on the derivation tree. I propose the following rule to link the elementary trees of a subordinate conjunction to an USR.

Rule (R1): The USR for $\beta 1(\text{Conj}_a)$ and $\beta 2(\text{Conj}_a)$[11] in which $Conj_a$ denotes a discourse relation R_a is the description of a DAG in which R_a left-dominates $\pi 1$ and $\pi 2$, the semantic representations of the arguments of $Conj_a$. This rule is graphically represented in Figure 7, in which a dashed-dotted line represents left-dominance.

Let us show how rule (R1) allows us to compute the right interpretations for the syntactic analyses SA1 and SA2 depending on their derivation trees.

Interpretations for SA1: From the derivation tree of SA1 given in Figure 2, rule (R1) leads to USR1 given in Figure 8. The constraints on left-dominance and order in USR1 are solved in DAGs (A) and (C). (C) is identical to USR1 except that immediate dominance replaces left-dominance. In (A), R_a left-dominates $\pi 2$. USR1 cannot be solved in (B) since, in (B), R_b dominates $\pi 2$ but does not left-dominate it. On the other hand, in (Duchier and Gardent, 2001) who use dominance, USR1 can be solved in (B), which is not in accordance with the data. This is why I have introduced the notion of left-dominance.

Interpretations for SA2: From the derivation tree of SA2 given in Figure 2, rule (R1) leads to USR2 given in Figure 8. The order of the multiple adjunctions to the same node in SA2 is echoed by the order of the leaves in USR2: $\pi 1$ precedes $\pi 2$ which precedes $\pi 3$. The constraints on left-dominance and order in USR2 are solved in DAGs (B) and (D) - which is correct - but also in (K) given in Figure 5[12]. However, (K) is excluded because it does not follow the left1-right2 principle, see note 9.

To conclude, with (R1), SA1 can be interpreted only as (A) or (C) and SA2 only as (B) or (D), which is correct.

[10] More formally, the nuclearity principle states that, in a (binary) tree rooted at R, the arguments of R are the *leaves* of the tree which are *left-dominated* by R.

[11] Recall (Section 3.2) that linear order does not affect dependency structures. So, $\beta 1(\text{Conj}_a)$ and $\beta 2(\text{Conj}_a)$ are both linked to the same (underspecified) semantic representation.

[12] I thank Laura Kallmeyer for drawing my attention on this point.

6 Comparison with D-LTAG

This study on sentences with two subordinates clauses is extended to other discourses. As requested by the reviewers, let me compare my approach to D-LTAG (Webber et al., 2003), which extends a sentence level grammar, namely XTAG, for discourse processing.

Let us first look at subordinating conjunctions. They anchor *auxiliary* trees in XTAG (or FTAG) and *initial* trees in D-LTAG. Why do they anchor *initial* trees in D-LTAG? The authors give the following answer: "One reason for taking something to be an *initial* tree is that its local dependencies can be stretched long-distance". That is a wrong argument. One major advantage of TAG is that adjunction is possible both in initial and auxiliary trees (and iteratively). So local dependencies in any tree can be stretched long-distance. Moreover, as any other adjuncts, several subordinate clauses can iteratively modify the same matrix clause (Section 2.2). One may wonder how iterativity is taken into account when subordinate conjunctions anchor initial trees.

Secondly, let us examine the distinction made in D-LTAG between structural and anaphoric connectives. The status of some connectives (e.g., *however*) is admittedly not clear and so is determined on empirical grounds, using crossed structural dependencies as a test. In note 8, I have emphasized that (5) comprises only structural connectives - subordinating conjunctions are unquestionably structural connectives in D-LTAG - while its structure exhibits crossing structural dependencies. So the main test to distinguish structural and anaphoric connectives is not valid.

D-LTAG defends the idea that there is no gap between sentence and discourse processing. There exist discrepancies, e.g., discourse adverbials have one argument at the (syntactic) sentence level and two at the (semantic) discourse level[13]. Such discrepancies are handled in the D-LTAG parsing system (Forbes et al., 2002) by the use of two passes: one based on XTAG syntactic trees, the other one on D-LTAG semantic trees. This amounts in positing two levels as in my approach however without a well-defined syntax-semantics interface.

Finally, in D-LTAG, the logical form of a discourse is computed from its derivation tree, a level of representation which is poor compared to SDRSs, e.g., there is no notion of Universe which groups the discourse referents. As said in Section 3.1, a rich semanctic level as the one proposed in SDRT is necessary for text understanding and also for text generation (Danlos et al., 2001). Moreover, the "anaphoric" behavior of discourse adverbials is seriously taken into account in SDRT. Unfortunately, I have no room left to discuss this issue.

[13]There is no such discrepancy for subordinating conjunctions.

References

A. Abeillé, M.-H. Candito, and A. Kinyon. 2000. The current status of FTAG. In *Proceedings of TAG+5*, pages 11–18, Paris.

N. Asher and A. Lascarides. 2003. *Logics of Conversation*. Cambridge University Press, Cambridge.

P. Blackburn and C. Gardent. 1998. A specification language for discourse semantics. In *Proceedings of LACL'98*, pages 61–67, Grenoble.

M.H. Candito and S. Kahane. 1998. Can the TAG Derivation Tree represent a Semantic Graph? In *Proceedings of TAG+4*, pages 25–28, Philadelphia.

L. Danlos, B. Gaiffe, and L. Roussarie. 2001. Document structuring à la SDRT. In *International workshop on text generation - ACL*, pages 94–102, Toulouse.

L. Danlos. 2004. Discourse dependency structures as constrained DAGs. In *Proceedings of SIGDIAL'04*, Boston.

D. Duchier and C. Gardent. 2001. Tree descriptions, contraints and incrementality. In R. Muskens H. Bunt and E. Thijsse, editors, *Computing Meaning*, pages 205–227. Kluwer Academic Publishers, Dordrecht.

K. Forbes, E. Miltsakaki, R. Prasad, A. Sarkar, A. Joshi, and B. Webber. 2002. D-LTAG system: Discourse parsing with a lexicalized tree adjoining grammar. *Journal of Logic, Language and Information*, 12(3):261–279.

L. Kallmeyer and A. Joshi. 2003. Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation*, 1(1–2):3–58.

H. Kamp and U. Reyle. 1993. *From Discourse to Logic*. Kluwer Academic Publishers, Dordrecht.

W. Mann and S. Thompson. 1987. Rhetorical structure theory. In G. Kempen, editor, *Natural Language Generation*, pages 85–95. Martinus Nijhoff Publisher, Dordrecht.

D. Marcu. 1996. Building up rhetorical structure trees. In *The Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1069–1074, Portland.

I. Mel'cuk. 2001. *Communicative Organization in Natural Language: The Semantic-Communicative Structure of Sentences.* John Benjamins Publishing Company, Amsterdam.

Y. Schabes and S. Shieber. 1994. An Alternative Conception of Tree-Adjoining Derivation. *Computational Linguistics*, 20(1):91–124.

B. Webber, A. Joshi, M. Stone, and A. Knott. 2003. Anaphora and discourse structure. *Computational Linguistics*, 44:1–45.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

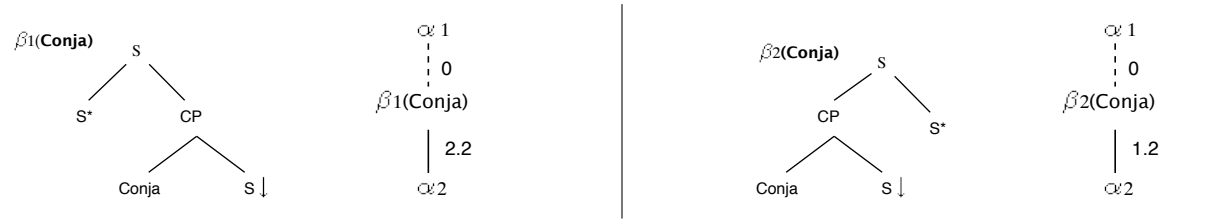


Figure 1: Auxiliary and derivation trees for postposed and preposed subordinate clauses

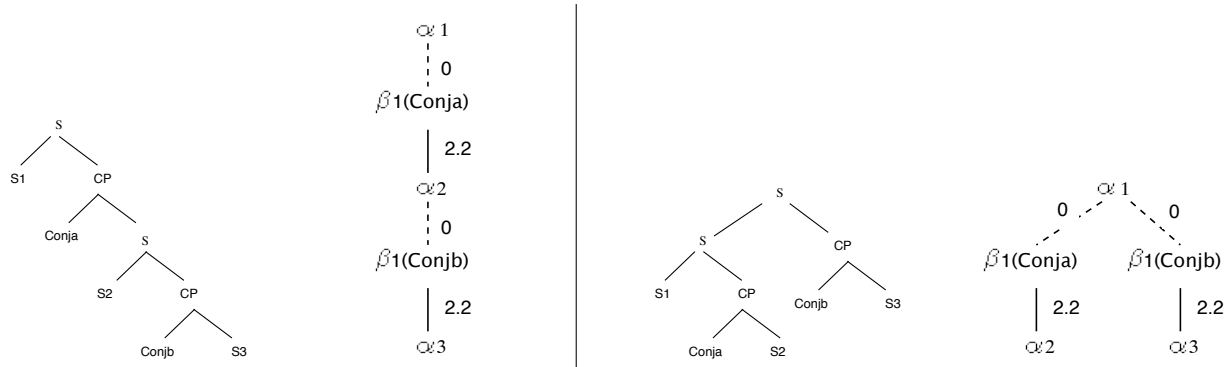


Figure 2: Derived and derivation trees for SA1 and SA2

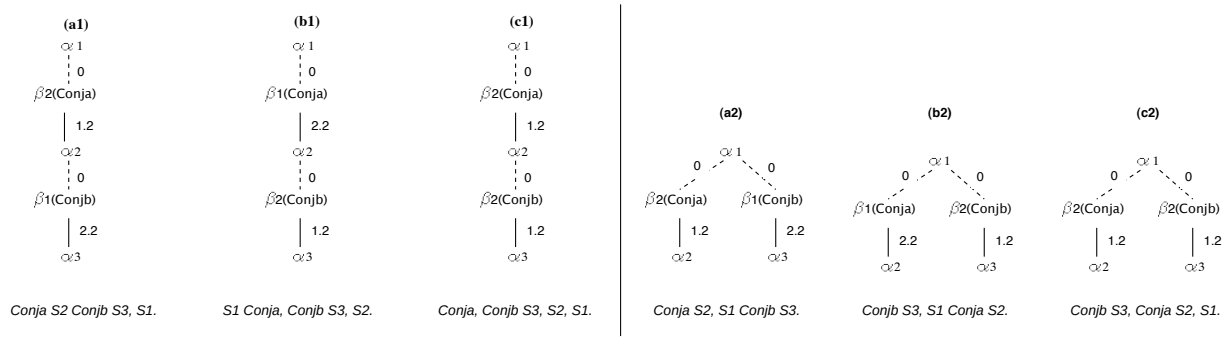


Figure 3: Other linear orders for SA1 and SA2

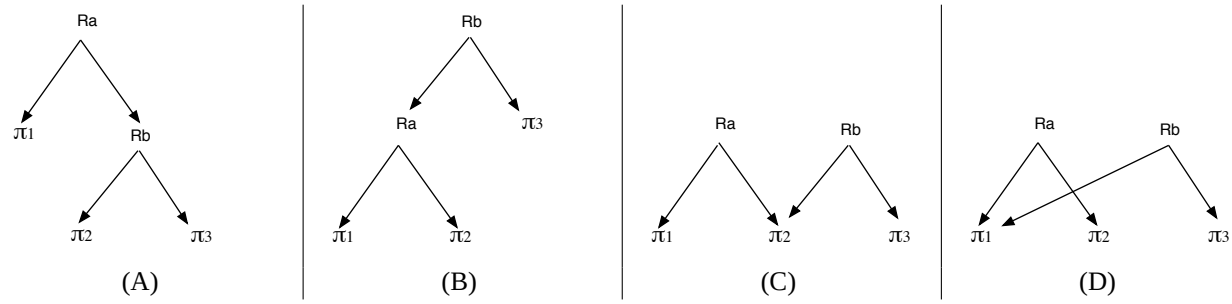


Figure 4: DAGs (A), (B), (C) and (D)

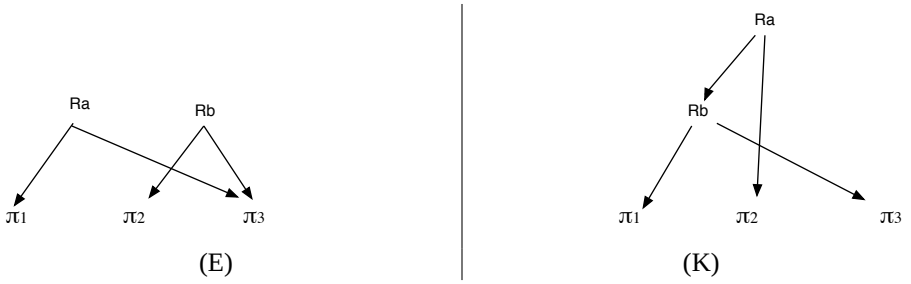


Figure 5: DAGs (E) and (K)

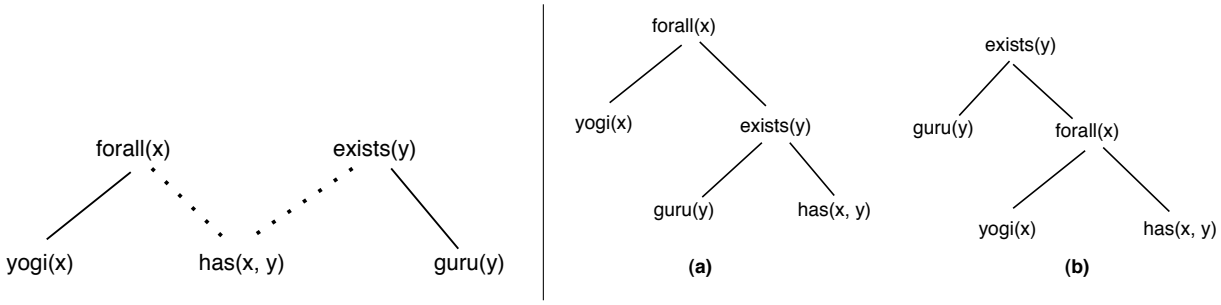


Figure 6: USR for *every yogi has a guru*, and representations with quantifier scope specified

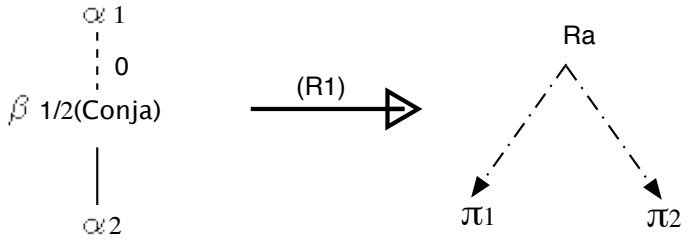


Figure 7: Rule (R1)

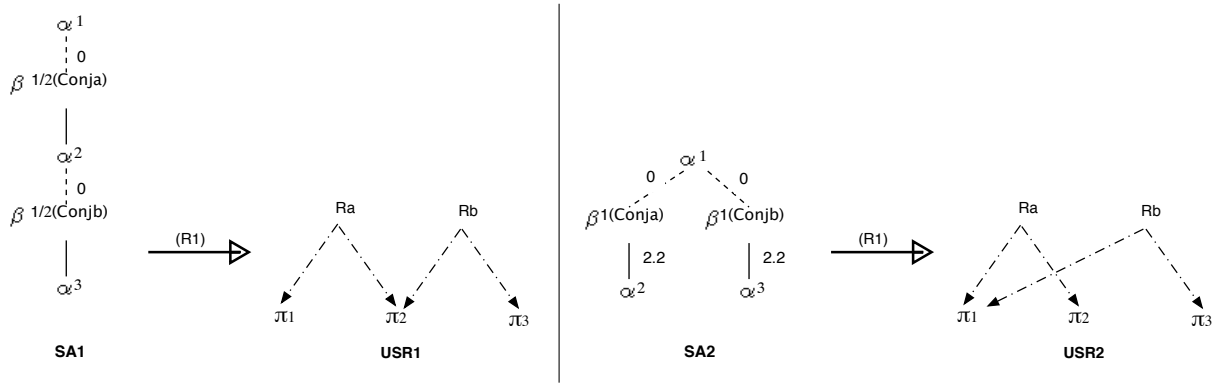


Figure 8: USR1 and USR2 (Underspecified Semantic Representations for SA1 and SA2),

TAG Parsing as Model Enumeration

Ralph Debusmann
Programming Systems Lab
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
rade@ps.uni-sb.de

Denys Duchier
Équipe Calligramme
LORIA – UMR 7503
Campus Scientifique, B. P. 239
54506 Vandœuvre lès Nancy CEDEX
France
duchier@loria.fr

Marco Kuhlmann
Programming Systems Lab
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
kuhlmann@ps.uni-sb.de

Stefan Thater
Computational Linguistics
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
stth@coli.uni-sb.de

Abstract

This paper introduces *well-ordered* derivation trees and makes use of this concept in a novel axiomatization of the TAG parsing problem as a constraint satisfaction problem. Contrary to prior approaches, our axiomatization focuses on the derivation trees rather than the derived trees. Well-ordered derivation trees are our primary models, whereas the derived trees serve solely to determine word order.

1 Introduction

Tree Adjoining Grammar (TAG) relates strings with two kinds of structures: derivation trees and corresponding derived trees. Derivation trees are more informative than their corresponding derived trees in the sense that the derived trees can be reconstructed from them. However, derivation trees are usually interpreted as unordered trees; they then cannot be used to formulate the TAG parsing problem directly, as they do not encode word order information.

This paper suggests to interpret derivation trees as *ordered* trees. It introduces the notion of *well-ordered* derivation trees: A derivation tree is called well-ordered if its nodes stand in the same precedence relation as the anchors in the corresponding derived tree. Because TAG can generate non-context-free languages, well-ordered derivation trees can be non-projective, i.e., they can contain "crossing" edges. The main contribution of this paper is an axiomatization of the exact form of non-projectivity licensed by TAG operations. It thereby provides a novel model-theoretic interpretation of the LTAG parsing problem.

The axiomatization of well-ordered derivation trees is put into practice in a description-based approach to TAG parsing, in which the parsing problem of strongly lexicalized TAGs[1] is interpreted as a *model enumeration problem*: given a description (a logical formula) ϕ of the input string, enumerate all and only those well-ordered derivation trees that are licensed by ϕ. Based on earlier work by Koller and Striegnitz (2002), we show that the solutions to this problem can in turn be characterised as the solutions of a constraint-satisfaction problem (CSP) on finite set integer variables, which can be solved by state-of-the-art constraint technology.

Our approach offers at least two interesting perspectives. First, it enables the encoding of LTAG grammars as certain *dependency grammars*, thereby illuminating the exact relation between the two formalisms. Second, the formulation of the LTAG parsing problem as a CSP opens up a large quantity of existing data to evaluate the constraint-based approach to parsing more thoroughly than what could be done before.

Plan of the paper. In §2, we show how the relation between TAG derivation trees and elementary trees can be formulated as the relation between models and logical descriptions, and introduce the notion of a TAG satisfiability problem. In §3, we extend satisfiability problems to parsing problems; we formalize the notion of well-ordered derivation trees as the structures under investigation in these problems, and show how their solutions can be obtained by solving a CSP. We illustrate this approach by means of an example in §4. §5 discusses the two perspectives of our approach mentioned above. Finally, §6 concludes the paper and presents ideas for future work.

2 TAG Satisfiability Problem

There are two major and constrasting formal perspectives on parsing: proof-theoretic and model-theoretic. The former emphasizes the construction of logical derivations, while the latter more directly states how models satisfy descriptions. The model-theoretic perspective (Cornell and Rogers, 1998) applied to a description-based specification of parsing problems is often more readily amenable to constraint-based processing. This is the view which we adopt for the remainder of this paper.

[1] A TAG is called strongly lexicalized, if each of its elementary trees contains exactly one anchor.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 148-154.

As a first step in our description-based approach to TAG parsing, we formulate the TAG *satisfiability problem*:

Given a multiset of elementary trees, can they be combined using operations of adjunction and substitution to form a complete derived tree?

To formally express TAG satisfiability problems, we introduce the following description language:

$$\phi ::= w : \tau \mid \phi \wedge \phi', \tag{1}$$

where w is taken from a set of *tree variables*, and τ from a set of TAG elementary trees of some grammar. We call $w : \tau$ a *tree literal*. We say that ϕ is *normal* if every tree variable in ϕ appears precisely in one tree literal.

It is well-known that the satisfiability problem is equivalent to the existence of a *derivation tree*, hence the idea to use derivation trees as models of normal ϕ descriptions. In order to make this more precise, we need some definitions:

Let Π be the set of finite paths, i.e. the finite sequences of positive integers. For simplicity in the following, we shall identify a node of a tree with the path $\pi \in \Pi$ that leads to it starting from the root of said tree.

A *derivation tree* is a tree (V, E) formed from vertices V and labeled edges $E \subseteq V \times V \times \Pi$. A model of a normal description $w_1 : \tau_1 \wedge \ldots \wedge w_n : \tau_n$ is a derivation tree where $V = \{w_1, \ldots, w_n\}$[2] and such that the following conditions are satisfied, where we write $w_1 - \pi \rightarrow w_2$ for an edge labeled with path π and representing either an adjunction or a substitution of τ_2 at node π of τ_1:

- If w_k is the root of the tree, then τ_k must be an initial tree.
- For each w_i, its outgoing edges must all be labeled with distinct paths, and for each substitution (resp. adjunction) node π in τ_i there must be exactly (resp. at most) one π-labeled edge.[3]
- For each edge $w_1 - \pi \rightarrow w_2$, if π is a substitution (resp. adjunction) node in τ_1, then τ_2 must be an initial (resp. auxiliary) tree.

In order to model lexical ambiguity, the description language can be extended with a limited form of disjunction, using for example the following extended language:

$$\phi ::= w_k : \{\tau_1^k, \ldots, \tau_{n_k}^k\} \mid \phi \wedge \phi',$$

where the set is to be interpreted disjunctively.

[2] Thus setting the interpretation of tree variables to be the identity substantially simplifies the presentation.

[3] For expositional simplicity, we do not cover adjunction constraints here. If an adjunction node is labeled with an adjunction constraint, then the exact well-formedness condition depends on that particular constraint.

The notion of TAG satisfiability problem as outlined above is implicit in (Koller and Striegnitz, 2002), who formulate the surface realization problem of natural language *generation* as the configuration problem of (unordered) dependency trees. A natural question is whether this treatment can be extended to parsing problems.

Given our formalization of TAG satisfiability problems, parsing problems cannot be expressed directly, as the models under consideration—derivation trees—are unordered trees. In order to express word order, a more natural class of models are derived trees, as these encode word order information in a direct way. However, the problem in using derived trees is that the formalization of the satisfaction relation becomes non-trivial, as the adjunction operation now requires a more complicated interpretation of elementary trees—not as atomic entities, but as groups of nodes that may get separated by material being "glued in between" by adjunction. If not conditioned carefully, this might lead to a formalism that is more expressive than TAG (Muskens, 2001; Rogers, 2003).

We suggest to solve the problem by considering derivation trees as being ordered. In the next section, we will introduce the notion of *well-ordered* derivation trees, which are possibly non-projective, ordered derivation trees whose precedence relation agrees with the precedence relation in the corresponding derived tree. This allows for an extension of the description language from (1) with precedence literals, which can be interpreted on (well-ordered) derivation trees in a straightforward way.

3 Well-ordered Derivation Trees

Our ambition is to tackle the parsing problem where word-order is part of the input specification. To this end, we formulate the TAG *parsing problem* analogously to our earlier definition of the TAG *satisfiability problem*:

*Given an **ordered** multiset of elementary trees, can they be combined using operations of adjunction and substitution to form a complete derived tree where the respective anchors are realized in the same order as stipulated by the input specification?*

To formally express the parsing problem, we extend our description language with precedence literals $w \prec w'$:

$$\phi \quad ::= \quad w : \tau \mid w \prec w' \mid \phi \wedge \phi' \tag{2}$$

where $w \prec w'$ means that w's anchor must precede w''s. For the same reasons as before, the approach that we will develop for this language will trivially extend to one with lexical ambiguity.

For the language of (1), the models where valid derivation trees. Now, however, we must additionally interpret the precedence literals of (2), which means that we need an order on the interpretations of the tree variables.

A derivation tree uniquely determines a derived tree and moreover uniquely determines a mapping I from each node π of each elementary tree w to its interpretation $I(w,\pi)$ as a node of the derived tree. The order that we are interested in is the one induced by the precedence between the interpretations of the anchors. More formally, writing $w_\diamond$ for the anchor node in w, we are interested in the order defined by:

$$w \prec w' \quad \equiv \quad I(w,w_\diamond) \prec I(w',w'_\diamond) \tag{3}$$

Thus, we arrive at the notion of a *well-ordered derivation tree*: a pair of a derivation tree and of the total order it induces on the elementary trees.

Unfortunately, we no longer have a simple way to enumerate these more complex models, unless we also construct the derived trees. Our contribution in this section is to show how the total order that we seek can also be obtained as the solution of a CSP.

3.1 Principles

To talk about order, we will need to talk about the set of anchors that are interpreted by nodes in the subtree (of the derived tree) rooted at $I(w,\pi)$. We write $\mathsf{yield}(w,\pi)$ for this set.

Assuming for the moment that we can freely use the notion of yield, we now show that the well-ordered derivation trees are precisely the valid derivation trees that satisfy the two principles of *convexity* and *order*:

Principle of convexity. The yield of the root of an elementary tree is convex. A set S is said to be convex with respect to a total order $\prec$ if for any $x \notin S$, x either precedes all elements of S or follows them all.

$$\forall w,w' \in V\colon w' \notin \mathsf{yield}(w,\varepsilon) \Rightarrow \\ w' \prec \mathsf{yield}(w,\varepsilon) \vee w' \succ \mathsf{yield}(w,\varepsilon) \tag{4}$$

where we write $x \prec S$ as a shorthand for $\forall y \in S.\ x \prec y$.

Principle of order. If π_1 and π_2 are leaves in elementary tree w and $\pi_1 \prec \pi_2$, then also $\mathsf{yield}(w,\pi_1) \prec \mathsf{yield}(w,\pi_2)$, i.e. all anchors below π_1 precede all anchors below π_2.

$$\forall w \in V\colon \forall \pi_1,\pi_2 \in \Pi\colon \\ \pi_1 \prec \pi_2 \wedge \pi_1 \in \mathsf{leaves}(w) \wedge \pi_2 \in \mathsf{leaves}(w) \Rightarrow \\ \mathsf{yield}(w,\pi_1) \prec \mathsf{yield}(w,\pi_2) \tag{5}$$

It is easy to show that these principles hold at every point of a TAG derivation. We now show that they suffice to completely determine the order among anchors. Consider the adjunction example of Figure 1. For brevity, we omit to say "the yield of": by (5), we know that $\alpha_1 \prec \alpha_2 \prec \alpha_3$ and $\beta_1 \prec \beta_2$. The adjunction places α_2 in the yield of the foot node of β. Therefore, again by (5), we have $\beta_1 \prec \alpha_2 \prec \beta_2$. Now by (4) $\beta_1 \cup \alpha_2 \cup \beta_2$ is convex, therefore α_1 must either precede or follow it. Since $\alpha_1 \prec \alpha_2$, we must have $\alpha_1 \prec \beta_1$. Similarly for α_3.

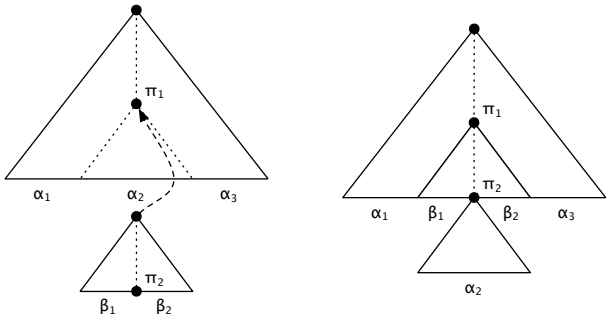


Figure 1: Adjunction

3.2 Axiomatization of Yield

Since we assume a strongly lexicalized version of TAG, each elementary tree has precisely one anchor. Therefore, for simplicity, we shall identify an anchor with the tree variable of the tree literal in which it appears. Thus $\mathsf{yield}(w,\pi)$ is a set of tree variables (standing for their respective anchors). We are going to show that $\mathsf{yield}(w,\pi)$ can also be obtained as the solution of a CSP. In order to do this, we will need to introduce additional functions.

$\mathsf{anchors}(w,\pi)$ is the set of anchors whose interpretations coincide with $I(w,\pi)$. Clearly:

$$\mathsf{anchors}(w,\pi) = \begin{cases} \{w\} & \text{if } \pi \text{ is anchor in } w \\ \emptyset & \text{otherwise} \end{cases} \tag{6}$$

$\mathsf{below}(w,\pi)$ is the set of anchors whose interpretations lie in the subtrees of the derived tree rooted at the interpretations of those nodes in w which are strictly dominated by the node π:

$$\mathsf{below}(w,\pi) = \{\mathsf{yield}(w,\pi') \mid \pi \lhd^+ \pi' \text{ in } w\} \tag{7}$$

$\mathsf{inserted}(w,\pi)$ concerns nodes where a substitution or adjunction has taken place. What is *inserted* is the yield of the tree which is being substituted or adjoined at node π of elementary tree w. We write $w - \pi \rightarrow w'$ for an edge in the derivation tree representing a substitution or an adjunction of w' at node π of w:

$$\mathsf{inserted}(w,\pi) = \begin{cases} \mathsf{yield}(w',\varepsilon) & \text{if } \exists w - \pi \rightarrow w' \\ \emptyset & \text{otherwise} \end{cases} \tag{8}$$

Finally, $\mathsf{pasted}(w,\pi)$ concerns foot nodes. When w is adjoined into w' at π', the subtrees hanging off π' in w' are cut out and pasted back under the foot node of w. Thus:

$$\mathsf{pasted}(w,\pi) = \begin{cases} \mathsf{below}(w',\pi') & \text{if } \pi \text{ is foot of } w, \\ & \text{and } \exists w' - \pi' \rightarrow w \\ \emptyset & \text{otherwise} \end{cases} \tag{9}$$

The yield can be obtained by taking the union of these quantities:

$$\mathsf{yield}(w,\pi) = \mathsf{anchors}(w,\pi) \cup \mathsf{inserted}(w,\pi) \cup \mathsf{pasted}(w,\pi) \cup \mathsf{below}(w,\pi) \quad (10)$$

The intuition for why this is correct can be outlined with an analysis by cases:

1. If π is anchor in w, then $\mathsf{anchors}(w,\pi) = \{w\}$ and all other quantities are empty sets.
2. If π is the foot node of w, then there must be an adjunction $w' - \pi' \rightarrow w$ and the anchors reachable from $I(w,\pi)$ are precisely those reachable from the material pasted at π as a result of the adjunction. anchors, inserted and below are all empty.
3. If a substitution has taken place at node π of w, then π is at least a leaf of w. The anchors reachable from $I(w,\pi)$ are precisely those reachable from the material that was inserted at (w,π). All other quantities are empty.
4. If an adjunction has taken place at node π of w, then at least π is not an anchor. The anchors reachable from $I(w,\pi)$ are now precisely those reachable from the material that was inserted at (w,π). Since $\mathsf{below}(w,\pi)$ is pasted at the foot node of the material that is being inserted, it ends up being included in $\mathsf{inserted}(w,\pi)$. anchors and pasted are empty.
5. If none of the above applies, then the anchors reachable from $I(w,\pi)$ are precisely those reachable from the children of π in w, i.e. from $\mathsf{below}(w,\pi)$. All other quantities are empty.

The definitions of (6,8,9) each contain a case analysis. In the definition of $\mathsf{anchors}(w,\pi)$ (6), the case condition is static: *is π the anchor of w or not?* Thus the satisfaction relation can be stated statically, and (6) can be interpreted as a constraint.

However, (8) and (9) both have conditions which dynamically depend on the existence of a substitution or adjunction dependency in the derivation tree. In order to arrive at a simple CSP, we need to slightly refine the formulation to avoid the case analysis. We take advantage of the fact that there is at most one adjunction (or substitution) at a given node:

$$\mathsf{inserted}(w,\pi) = \cup\{\mathsf{yield}(w',\varepsilon) \mid w - \pi \rightarrow w'\} \quad (11)$$

Let $\mathsf{children}(w,\pi) = \{w' \mid w - \pi \rightarrow w'\}$. (Duchier, 2003) showed how this equation could be reduced to a constraint and included in the CSP. Thus we obtain:

$$\mathsf{inserted}(w,\pi) = \cup\{\mathsf{yield}(w',\varepsilon) \mid w' \in \mathsf{children}(w,\pi)\} \quad (12)$$

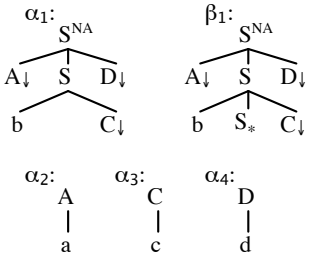
Again, as shown in (Duchier, 2003), this equation has precisely the form required for implementation by the *selection union constraint*. Similarly for pasted we obtain:

$$\mathsf{pasted}(w,\pi) = \begin{cases} \cup\{\mathsf{below}(w',\pi') \mid w \in \mathsf{parents}(w',\pi')\} & \text{if } \pi \text{ is foot in } w \\ \emptyset & \text{otherwise} \end{cases} \quad (13)$$

Given a (normal) TAG parsing problem, we are now able enumerate its models (the well-ordered derivation trees) as solutions of a CSP. First, the part of the CSP which enumerates the derivation trees remains as described by Koller and Striegnitz (2002). Second, for each node π of a tree w, we add the constraints (6,7,10,12,13): this allows us to obtain $\mathsf{yield}(w,\pi)$ as the solution of a CSP. Finally, we add the constraints corresponding to the principles of convexity and order, and the ordering constraints from the specification of the parsing problem. In this manner, we obtain a CSP which allows us to enumerate all and only the *well-ordered derivation trees*.

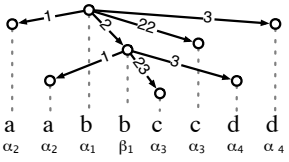
4 Example

We now show how our axiomatization of yield and the axiomatic principles derive the correct precedence constraints for a sample LTAG grammar. The grammar G that we are considering is the following:



G produces the language $L = \{a^n b^n c^n d^n \mid n \geq 1\}$, a language not contained in the set of context-free languages.

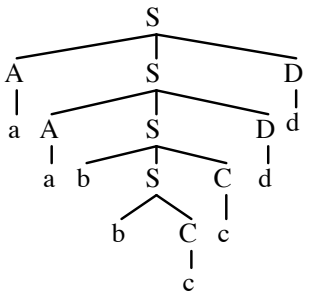
Given G, the derivation tree for the string *aabbccdd* can be drawn as follows:



(Recall that a label π on an edge $w_1 - \pi \rightarrow w_2$ denotes the path address in w_1 at which the substitution or adjunction of w_2 has taken place.) In this tree, all edges except one correspond to substitutions; the edge from the left *b* to the right *b* corresponds to the adjunction of β_1 into α_1.

The given drawing of the derivation tree is well-ordered: The order of the anchors in it (connected to the

nodes by vertical dashed lines) corresponds precisely to the order of the anchors in the corresponding derived tree:



To illustrate the axiomatization of yield, we give the yields of the elementary trees participating in the derivation of the string *aabbccdd* in Figure 2. Each table row pertains to a pair (w,π) of an elementary tree w (identified by its anchor) and a path address π in w, and shows the sets $\mathsf{anchors}(w,\pi)$, $\mathsf{inserted}(w,\pi)$, $\mathsf{pasted}(w,\pi)$ and $\mathsf{below}(w,\pi)$, whose union equals $\mathsf{yield}(w,\pi)$. With respect to the case analysis in the preceding section, each pair (row) corresponds to one of the cases:[4]

1. $(a_1,1)$, $(a_2,1)$, $(c_1,1)$ $(c_2,1)$, $(d_1,1)$ and $(d_2,1)$ correspond to the first case (anchor) since for all these elementary trees, 1 is the address of the anchor. The same holds for $(b_1,21)$ and $(b_2,22)$.

2. $(b_2,22)$ corresponds to the second case (foot node). This is the most interesting case, where the anchors below the adjunction site $(b_1,2)$ (i.e. b_1 and c_2) are "pasted" at the foot node $(b_2,22)$ of b_2.

3. $(b_1,1)$, $(b_1,22)$, $(b_1,3)$, $(b_2,1)$, $(b_2,23)$ and $(b_2,3)$ correspond to the third case (substitution), where $\mathsf{insert}(w,\pi)$ is the only non-empty set, containing the yields of the substituted trees.

4. $(b_1,2)$ corresponds to the fourth case (adjunction). This works like substitution.

5. The pairs (w,ε) correspond to the fifth case (else case) in the case analysis in the preceding section, where only the $\mathsf{below}(w,\pi)$ set is non-empty, containing the yields of the nodes below. The same holds for $(b_2,2)$.

5 Perspectives

The notion of well-ordered derivation trees offers some interesting perspectives. First, it allows us to encode each LTAG grammar into an equivalent dependency grammar. Second, the axiomatization of well-ordered derivation trees can be transformed into a constraint-based parser for LTAG in a straightforward way.

[4]In order to distinguish several occurrences of letters from each other, we have indexed them.

5.1 TAG and Dependency Grammar

If we ignore word order, derivation trees have a natural reading as dependency trees: anchors of elementary trees correspond to lexical entries, substitution and adjunction edges mirror the lexical entry's valency.

Koller and Striegnitz (2002) develop this insight and formulate the surface realization problem of natural language generation as a parsing problem in a dependency grammar with free word order. In their approach, the dependency grammar lexicon is induced by "reading off" the valencies of elementary trees: substitution sites are encoded as obligatory valencies, adjunction sites as valencies that can be filled arbitrarily often.[5] This encoding embeds TAG into dependency grammar in that well-formed dependency trees directly correspond to TAG derivation trees and, indirectly, derived trees. However, the embedding is weak in the sense that its correctness of the encoding relies upon the fact that word order cannot be specified in the grammar; thus, the encoding cannot be applied to parsing problems.

The notion of well-ordered derivation trees allows us to extend the encoding to directly formulate parsing problems. To this end, we need to (i) specify the local (linear) order of the substitution valencies of each lexical entry, (ii) specify the local dominance relation among valencies and (iii) restrict the class of models to well-ordered derivation trees. Both the local order and the local dominance relations can be read off the LTAG elementary trees. The restriction of the class of models to well-ordered derivation trees then guarantees that the locally specified orderings are consistent with the global order in the dependency tree.

From other work on the interpretation of TAG as dependency grammar (Joshi and Rambow, 2003), this encoding is distinguished by three features:

- It does not stipulate any non-traditional rules to combine dependency structures, but only uses the standard "plugging" operation to fill valencies.
- It does not assume nodes in the dependency tree except for the nodes introduced by the (anchors of the) elementary trees.
- It is able to maintain the dependencies associated to a lexical anchor throughout the derivation. This is achieved by "hiding" the structure of the (partially) derived tree in the axiomatization of well-ordered derivation trees.

5.2 Constraint-based parsing of LTAGs

To solve the problem of surface realization as dependency parsing, Koller and Striegnitz (2002) successfully

[5]The encoding is based on a notion of derivation trees where different auxiliary trees can be adjoined at the same node.

(w,π)	anchors	inserted	pasted	below
$(a_1,1)$	$\{a_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(a_2,1)$	$\{a_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(c_1,1)$	$\{c_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(c_2,1)$	$\{c_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(d_1,1)$	$\{d_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(d_2,1)$	$\{d_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(b_1,21)$	$\{b_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(b_2,21)$	$\{b_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
$(b_2,22)$	$\emptyset$	$\emptyset$	$\{b_1,c_2\}$	$\emptyset$
$(b_1,1)$	$\emptyset$	$\{a_1\}$	$\emptyset$	$\emptyset$
$(b_1,22)$	$\emptyset$	$\{c_2\}$	$\emptyset$	$\emptyset$
$(b_1,3)$	$\emptyset$	$\{d_2\}$	$\emptyset$	$\emptyset$
$(b_2,1)$	$\emptyset$	$\{a_2\}$	$\emptyset$	$\emptyset$
$(b_2,23)$	$\emptyset$	$\{c_1\}$	$\emptyset$	$\emptyset$
$(b_2,3)$	$\emptyset$	$\{d_1\}$	$\emptyset$	$\emptyset$
$(b_1,2)$	$\emptyset$	$\{a_2,b_2,c_1,c_2,d_1\}$	$\emptyset$	$\emptyset$
(a_1,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{a_1\}$
(a_2,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{a_2\}$
(c_1,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{c_1\}$
(c_2,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{c_2\}$
(d_1,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{d_1\}$
(d_2,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{d_2\}$
(b_1,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{a_1,a_2,b_1,b_2,c_1,c_2,d_1,d_2\}$
(b_2,ε)	$\emptyset$	$\emptyset$	$\emptyset$	$\{a_2,b_2,c_1,c_2,d_1\}$
$(b_2,2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{b_1,b_2,c_1,c_2\}$

Figure 2: Yields in the analysis of string *aabbccdd*

employ an existing constraint-based parser for Topological Dependency Grammar (Duchier and Debusmann, 2001). In light of the fact that surface realization is an NP-complete problem, the efficiency of this parser is quite remarkable. One of the major questions for a description-based approach to LTAG *parsing* is, whether the benign computational properties of existing, derivation-based parsers for LTAG[6] can be exploited even in the constraint framework.

We have started work into this direction by implementing a prototypical constraint parser for LTAG, and investigating its properties. The implementation can be done in a straightforward way by transforming the axiomatization of well-ordered derivation trees that was given in Section 3 into a constraint satisfaction problem along the lines of Duchier (2003). The resulting parser is available as a module for the XDG system (Debusmann, 2003).

Preliminary evaluation of the parser using the XTAG grammar shows that it is not competitive with state-of-the-art TAG parsers (Sarkar, 2000) in terms of run-time; however, this measure is not the most significant one for an evaluation of the constraint-based approach anyway. More importantly, a closer look on the search spaces explored by the parser indicates that the inferences drawn from the axiomatic principles are not strong enough to rule out branches of the search that lead to only inconsistent assignments of the problem variables. Future work needs to closely investigate this issue; ideally, we would arrive at an implementation that enumerates all well-ordered derivation trees for a given input without failure.

One of the benefits of the constraint formulation of dependency parsing given in Duchier (2003) is that it provides a means of effectively dealing with disjunctive information, e.g. information introduced by lexical ambiguity. The idea is to make the common information in a set of possible lexical entries available to the constraint solver as soon as possible, without waiting for one entire lexical entry from the set to be selected. If e.g. all elementary trees still possible for a given word are of different shape, but have the same number of substitution and adjunction sites labeled with the same categories—i.e., have the same valencies—, the constraint solver can configure the derivation tree before it would need to commit to any specific candidate tree. The question of whether this technique can be applied to widen the bottleneck that lexical ambiguity constitutes for TAG parsing awaits further exploration. With the encoding presented here, and the large

[6]The parsing problem of LTAG can be decided in time $O(n^6)$.

grammatical resources for LTAG that it makes available to the application of constraint parsing, we are at least in the position now to properly evaluate the effectiveness of the constraint-based treatment of constructive disjunction.

6 Conclusion

In this paper, we introduced the notion of well-ordered derivation trees. Using this notion, we presented an axiomatization of the TAG parsing problem with a natural interpretation as a constraint satisfaction problem. The main burden lay in the axiomatization of the yield, which captures the dynamic aspects of a TAG derivation in terms of declarative constraints.

Contrary to previous approaches, we have shifted the emphasis away from the derived trees to the derivation trees. From this perspective, the derivation tree is the crucial ingredient of a TAG analysis, whereas the derived tree serves solely to constrain word order. This focusing on the derivation tree brings our approach in closer vicinity to dependency grammar.

Our approach yields two new avenues for future research. The first is to encode LTAG grammars into equivalent dependency grammars, and to intensify research on the relationship between TAG and DG. Second, the axiomatization of well-ordered derivation trees can be straightforwardly transformed into a constraint-based parser for LTAG. Koller and Striegnitz (2002) have shown that a similar approach can yield interesting results for generation, but we have not yet been able to reproduce them for parsing. To this end, we are moving towards the abstract notion of a *configuration problem* encompassing the constraint-based processing of both TAG, DG, and related frameworks, even semantic ones. We think that this abstraction eases the search for efficiency criteria for solving particular configuration problems, and can thus help us to pin down ways how to do efficient constraint-based TAG parsing in particular.

References

Thomas Cornell and James Rogers. 1998. Model theoretic syntax.

Ralph Debusmann. 2003. A parser system for extensible dependency grammar. In Denys Duchier, editor, *Prospects and Advances in the Syntax/Semantics Interface*, Lorraine-Saarland Workshop Series, pages 103–106. LORIA, Nancy/FRA.

Denys Duchier and Ralph Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the 39th ACL*, Toulouse, France.

Denys Duchier. 2003. Configuration of labeled trees under lexicalized constraints and principles. *Journal of Research on Language and Computation*, 1(3/4).

Aravind Joshi and Owen Rambow. 2003. A formalism for dependency grammar based on tree adjoining grammar. In *Proc. of the First International Conference on Meaning-Text Theory*, pages 207–216, Paris, June.

Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of the 40th ACL*, Philadelphia.

Reinhard Muskens. 2001. Talking about Trees and Truth-conditions. *Journal of Logic, Language and Information*, 10(4):417–455.

James Rogers. 2003. Syntactic structures as multi-dimensional trees. *Journal of Research on Language and Computation*, 1(3/4).

Anoop Sarkar. 2000. Practical experiments in parsing using tree adjoining grammars. In *Proceedings of the Fifth Workshop on Tree Adjoining Grammars, TAG+ 5*, Paris.

LTAG Semantics with Semantic Unification

Laura Kallmeyer
TALaNa-Lattice
UFRL, University Paris 7
2 place Jussieu, Case 7003,
75251 Paris Cedex 05
France
lkallmey@linguist.jussieu.fr

Maribel Romero
Department of Linguistics
610, Williams Hall
University of Pennsylvania
Philadelphia, PA, 19104-6305
U.S.A.
romero@ling.upenn.edu

Abstract

This paper sets up a framework for LTAG (Lexicalized Tree Adjoining Grammar) semantics that brings together ideas from different recent approaches addressing some shortcomings of TAG semantics based on the derivation tree. Within this framework, several sample analyses are proposed, and it is shown that the framework allows to analyze data that have been claimed to be problematic for derivation tree based LTAG semantics approaches.

1 Introduction

An LTAG (Joshi and Schabes, 1997) consists of a finite set of *elementary* trees associated with lexical items. From these trees, larger trees are derived by substitution (replacing a leaf with a new tree, a so-called *initial tree*) and adjunction (replacing an internal node with a new tree, a so-called *auxiliary tree*).

The elementary trees of an LTAG represent extended projections of lexical items and encapsulate all syntactic/semantic arguments of the lexical anchor. They are minimal in the sense that only the arguments of the anchor are encapsulated, all recursion is factored away. These linguistic properties of elementary trees are formulated in the *Condition on Elementary Tree Minimality (CETM)* from (Frank, 1992).

LTAG derivations are represented by derivation trees that record the history of how the elementary trees are put together. A derived tree is the result of carrying out the substitutions and adjunctions. Each edge in the derivation tree stands for an adjunction or a substitution. The edges are equipped with Gorn addresses of the nodes where the substitutions/adjunctions take place.[1] See for example the derivation of (1) in Fig. 1.

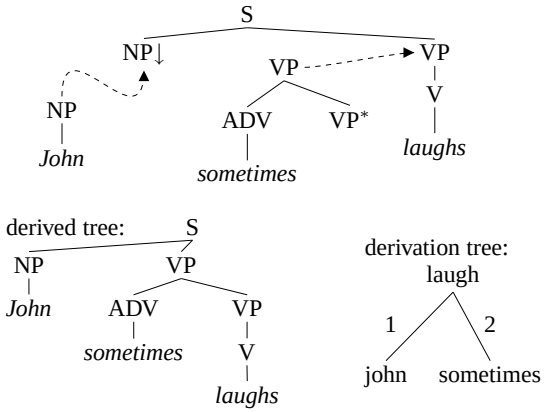


Figure 1: TAG derivation for (1)

(1) John sometimes laughs

Taking into account the minimality of elementary trees and the fact that derivation steps in TAG correspond to predicate-argument applications, it seems appropriate to base LTAG semantics on the derivation tree (Candito and Kahane, 1998; Joshi and Vijay-Shanker, 1999; Kallmeyer and Joshi, 2003). However, it has been observed that in some cases this is problematic since the derivation tree does not provide enough information to correctly construct the desired semantic dependencies.

The goal of this paper is to bring together ideas from several recent approaches in order to develop a general framework for LTAG semantics that allows us to compute semantic representations on the derivation tree, overcoming some otherwise problematic cases. Within this framework we then sketch several sample analyses.

2 Previous approaches to LTAG semantics

The data that are claimed to be the most problematic for derivation tree based LTAG semantics (see (Rambow et

[1] The root has the address 0, the jth child of the root has address j and for all other nodes: the jth child of the node with address p has address $p \cdot j$.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 155-162.

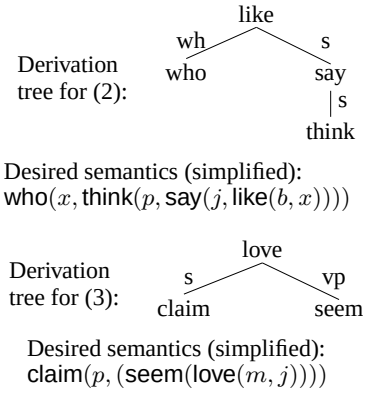


Figure 2: Problematic derivation trees for semantics

al., 1995; Dras et al., 2004; Frank and van Genabith, 2001; Gardent and Kallmeyer, 2003)) are long-distance wh-movements as in (2) and interactions of attitude verbs and raising verbs or adverbs as in (3).

(2) Who does Paul think John said Bill liked?

(3) a. Mary, Paul claims John seems to love
b. Paul claims Mary apparently loves John

The problem of (2) is that in the LTAG analysis, *who* is substituted into the wh-NP node of *like*, *say* is adjoined to the lower S node of *like* and *think* adjoins to *say*. Consequently, in the derivation tree (see Fig. 2), there is neither a link between *who* and *think* nor a link between *like* and *think*.[2] But in the semantics, we want the *think* proposition to be the scopal argument of the wh-operator, i.e., a link between *who* and *think* must be established. This can be done via the semantics of *like* but at least some possibility to link *like* to *think* is necessary. In (3), *claim* and *seem* (or *apparently* resp.) adjoin to different nodes in the *love* tree, i.e., they are not linked in the derivation tree. But the propositional argument of *claim* is the *seems* (*apparently* resp.) proposition. This case however is less hard than (2) since one can choose the semantics of *like* in such a way that the desired scope orders are obtained without a direct link between the embedding attitude verb and the embedded raising verb (adverb resp.). A semantics in the (Kallmeyer and Joshi, 2003) framework is possible here. Example (2) however poses a serious problem for derivation tree based approaches.

Several proposals have been made to avoid the problems that arise when doing semantics based on the derivation tree:

Instead of using the derivation tree for semantics, one could try to compute semantics based on the derived tree. Such an approach is pursued in (Frank and van Genabith, 2001). However, their approach makes use not only of the information available in the derived tree but also of information about how the elementary trees were put together, i.e., of information available in the derivation tree. Therefore, in a sense, their semantics is based on both, derived and derivation tree. Considering that one of the guiding linguistic principles of LTAG is semantic minimality of elementary tree, i.e. that the semantics of elementary trees is non-decomposable, it is more appropriate to link semantic representations to whole elementary trees and to abstract away (at least to a certain degree) from the concrete shape of the elementary trees. This amounts to linking semantic representations to nodes in the derivation tree.

An alternative proposal for computing semantics only on the derivation tree is to enrich the derivation tree with additional links as in (Kallmeyer, 2002a; Kallmeyer, 2002b). In this approach, the derived tree needs not be considered for computing semantics. The problem with this proposal is that sometimes it is not clear which link one has to follow in order to find the value for some semantic variable. Therefore additional rules for ordering the links for semantic computation are needed. The result is a rather complex machinery in order to obtain the dependencies needed for semantics.

More recently, (Gardent and Kallmeyer, 2003) propose to use the feature unification mechanism in the syntax, i.e., in the derived tree, in order to determine the values of semantic arguments. The underlying observation is that whenever a semantic link in the derivation tree is missing, it is either a) a link between trees attaching to different nodes in the same tree (see(3)), i.e., attaching to nodes that can share features inside an elementary tree, or b) a link between trees γ_1 and γ_2 such that γ_2 adjoins to the root of a tree that (adjoins to the root of a tree that ...) attaches to some node μ in γ_1 (see (2)). In this case, indirectly, the top of μ and the top of the root of γ_2 unify and thereby features can be shared. This approach works in the problematic cases and it has the advantage of using a well-defined operation, unification, for semantic computation. But it has the disadvantage of using the derived tree for semantics even though semantic representations are assigned to whole elementary trees (i.e., to nodes in the derivation tree) and not to nodes in the derived tree. Furthermore, the feature structures needed for semantics are slightly different form those used for syntax since they contain semantic variables and labels as possible feature values. Consequently, the number of feature structures is no longer finite (in contrast to feature-based TAG (FTAG) as defined in (Vijay-Shanker and Joshi, 1988)) and therefore the generative capacity of the formalism is extended. In other words, a more powerful formalism is used for syntax just because it is needed for the specific

[2]For the sake of readability, we use names np, vp, r for *root*, f for *foot*, ... for the node positions instead of the usual Gorn adresses.

semantic features.[3]

In order to separate more neatly between syntax with feature structures linked to nodes in the derived tree and semantics where semantic representations are linked to nodes in the derivation tree, we propose in the following to incorporate semantic feature structures in the derivation tree. Formally, this means just extracting the semantic features used in (Gardent and Kallmeyer, 2003) from the derived trees and putting them in a semantic feature structure linked to the semantic representation of the tree in question. Of course one still has to link semantic features to specific node positions in the elementary tree, e.g., in order to make sure that syntactic argument positions get correctly linked to the corresponding semantic arguments.

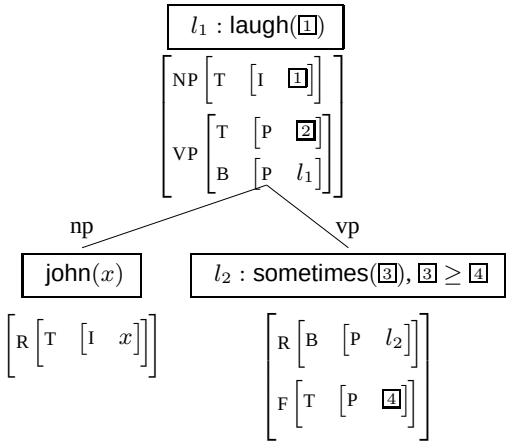


Figure 3: Semantic representations for (1) *John sometimes laughs*

3 LTAG semantics with semantic unification

3.1 Semantic feature structures

Semantic representations are as defined in (Kallmeyer and Joshi, 2003) except that they do not have argument variables: they consist of a set of formulas (typed λ-expressions with labels) and a set of scope constraints. A scope constraint is an expression $x \geq y$ where x and y are propositional labels or propositional variables (these last correspond to the holes in (Kallmeyer and Joshi, 2003)). Each semantic representation is linked to a semantic feature structure. Semantic feature structures are typed feature structures, the type of the whole feature structure is sem. The definition of the feature structures is as follows:

- a feature structure of type sem consists of features 0 (the root position), 1, 2, ..., 11, 12, ... for all node positions that can occur in elementary trees (finite for each TAG), the values of these features are of type tb (for 'top-bottom')
- a feature structure of type tb consists of a T and a B feature (top and bottom) whose values are feature structures of type $bindings$
- a feature structure of type $bindings$ consists of a feature I whose values are individual variables, a feature P whose values are propositional labels, etc.

3.2 Semantic unification

Semantic composition consists only of feature unification. It corresponds to the feature unifications in the syntax that are performed during substitutions and adjunctions and the final top-bottom unifications in the derived tree. In the derivation tree, elementary trees are replaced by their semantic representations plus the corresponding semantic feature structures. Then, for each edge in the derivation tree from γ_1 to γ_2 with position p:

- The top feature of position p in γ_1 and the top feature of the root position in γ_2, i.e., the feature structures $\gamma_1.p.\text{T}$ and $\gamma_2.0.\text{T}$ are identified,
- and if γ_2 is an auxiliary tree, then the bottom feature of the foot node of γ_2 and the bottom feature of position p in γ_1, i.e., (if f is the position of the foot node in γ_2) the feature structures $\gamma_1.p.\text{B}$ and $\gamma_2.f.\text{B}$ are identified.

Furthermore, for all γ in the derivation tree and for all positions p in γ such that there is no edge from γ to some other tree with position p: the T and B features of $\gamma.p$ are identified.

By these unifications, some of the variables in the semantic representations get values. In the end, after having performed these unifications, the union of all semantic representations is built. The result is an underspecified representation.[4]

3.3 A sample derivation

As an example consider the analysis of (1): Fig. 3 shows the semantic representations and the semantic feature structures of the three elementary trees involved in the derivation.

[3] A similar approach is (Stone and Doran, 1997) where, as in (Gardent and Kallmeyer, 2003), each elementary tree has a flat semantic representation, the semantic representations are conjoined when combining them and variable assignments are done by unification in the feature structures on the derived tree. But there is no underspecification, and the approach is less explicit than (Gardent and Kallmeyer, 2003).

[4] For combining feature structure, we adopt an operational way in this paper because this is general practice in LTAG. I.e., unification is an operation on actual structures. Viewing feature structures as descriptions and thinking of unification as finding a consistent model (see, e.g., (Johnson, 1994)), is of course possible as well. But then one needs additional constraints that reflect the identifications performed during substitution, adjunction and top-bottom feature structure unification.

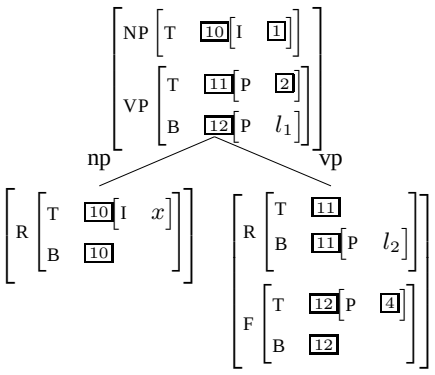


Figure 4: Semantic unification for (1)

The different unifications lead to the feature value identities in Fig. 4. This gives the identities $\boxed{1} = x$, $\boxed{2} = l_2$, and $\boxed{4} = l_1$, which results in the following semantic representation:

(4) $l_1 : \mathsf{laugh}(x), \mathsf{john}(x), l_2 : \mathsf{sometimes}(\boxed{3}), \boxed{3} \geq l_1$

In the end, appropriate disambiguations must be found. These are assignments for the remaining variables, i.e., functions that assign propositional labels to propositional variables, respecting the scope constraints (Kallmeyer and Joshi, 2003). The disambiguated representation is then interpreted conjunctively. (4) has only one disambiguation, namely $\boxed{3} \rightarrow l_1$. This leads to $\mathsf{john}(x) \wedge \mathsf{sometimes}(\mathsf{laugh}(x))$.

4 Alternative ways of obtaining scope constraints

Instead of stating explicit scope constraints of the form $x \geq y$, one could imagine two other possibilities: either i) not using any scope constraints at all and obtaining scope by identifying propositional variables and propositional labels by unification during the derivation, or ii) obtaining scope constraints from the final top-bottom unification instead of stating them explicitely, i.e., not doing real top-bottom unification but adding instead a constraint top $\geq$ bottom whenever a node has not been used for attaching other elementary trees. These alternatives are illustrated in Fig. 5 and 6.

Possibility i) has the obvious problem that is does not allow for underspecified representations, which means that in cases of scope ambiguities the number of representations one would have to generate would explode. Possibility ii) looks more interesting. In Fig. 5 for example, the B feature of position 2 in *laugh* is unified with the (empty) B feature of position 2 in *sometimes* so that in the result, there is a node with T [P $\boxed{3}$] and B [P l_1]. From this node, the desired scope constraint $\boxed{3} \geq l_1$ can

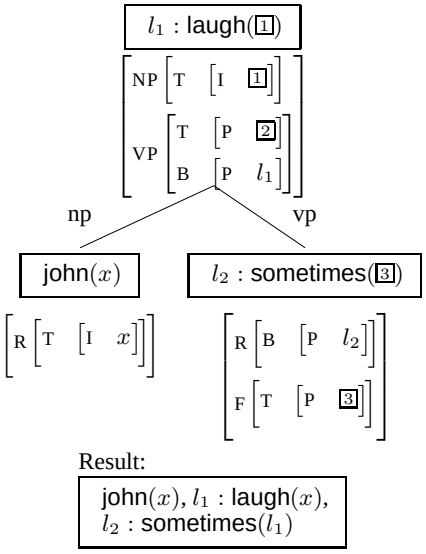


Figure 5: Alternative i): Analysis of (1) without scope constraints

be obtained. One problem with ii) is that in some cases one might need the original final top-bottom unification, so one would have to distinguish between cases where a scope constraint has to be added (these are perhaps the cases of P features) and cases where usual unification is done. But even more problematic is that in some cases, it is not possible to obtain all scope constraints one needs by the final top-bottom mechanism. Examples are cases where two quantifier scope parts attach to the same node as in (5).

(5) someone likes everybody

Following (Kallmeyer and Joshi, 2003), we suppose that the contribution of a quantifier consists of an NP initial tree (the predicate argument part) and a separate auxiliary tree with just one S node (the scope part). The analysis of (5) with possibility ii) is sketched in Fig. 7. The scope constraints one wants to obtain are 1) those that place the proposition coming with the noun in the restriction of the quantifiers, i.e., $\boxed{4} \geq l_3$ and $\boxed{8} \geq l_5$, 2) those that place the *like* proposition in the nuclear scope of the quantifiers, i.e., $\boxed{5} \geq l_1$ and $\boxed{9} \geq l_1$, and 3) those that limit the scope of the quantifier inside the sentence the quantifier attaches to, i.e., $\boxed{1} \geq l_2$ and $\boxed{1} \geq l_4$.[5] For 1) and 2), corresponding top and bottom feature have to be put on some node, in Fig. 7 they are on positions N and L (for the lexical anchor) of the NP tree. However, this is very arbitrary, they are not really related to these nodes. Therefore, it is much more appropriate to state the constraints in a general way

[5]The last constraints are important to make sure that in examples as *Mary thinks John likes everybody* the embedded quantifier cannot take scope over *thinks*.

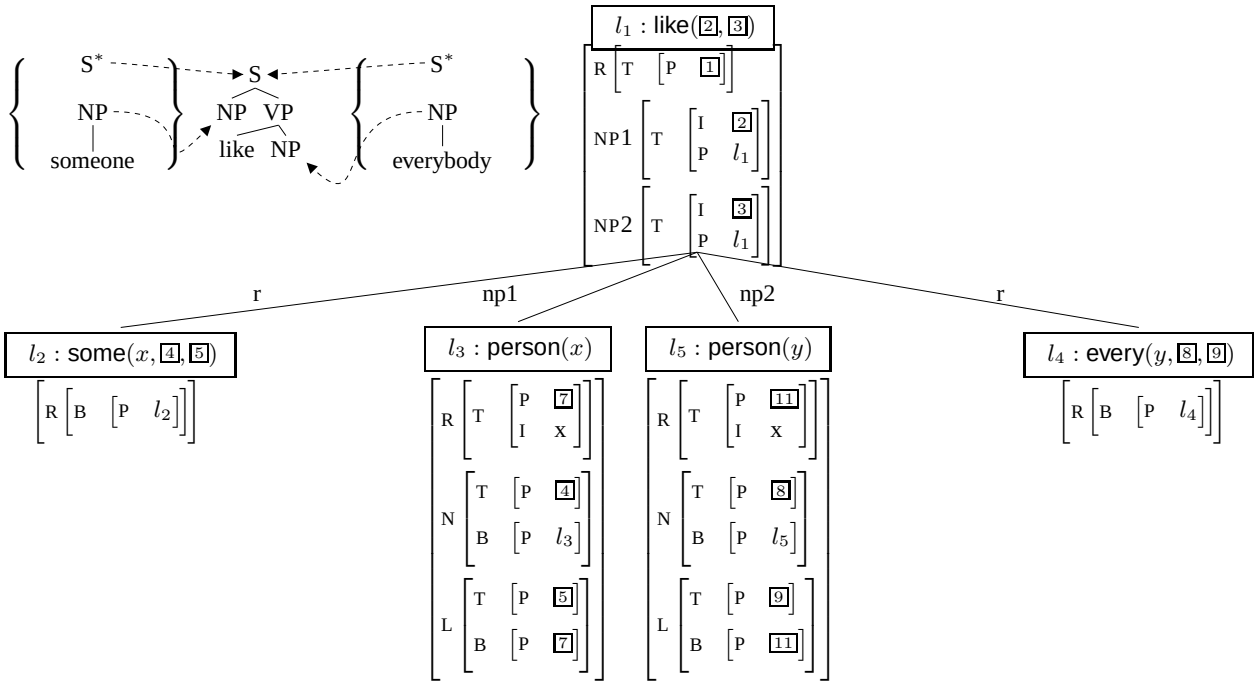


Figure 7: Problematic case for possibility ii): Analysis of (5)

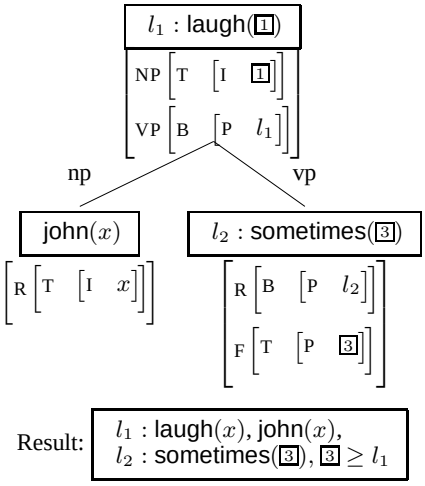


Figure 6: Alternative ii): scope constraints from top-bottom unification

and to link them only to the semantic representation without linking them to any node. The constraints 3) have to come from the scope tree, i.e., they have to be linked to its root since this is the only node in these trees. But this is not possible since in the course of the derivation, the bottom parts of all scope parts attaching to the same node unify because of the unifications done during adjunction. In Fig. 6 for example this means that [P l_2] and [P l_4] are unified, which leads to a failure.

Because of these considerations we decided not to choose possibilities i) or ii) but to state scope constraints explicitely in the semantic representations and use semantic unification with final top-bottom unification as described above.

5 Comparison to Gardent & Kallmeyer

Among the approaches to LTAG semantics mentioned in the beginning of this section, (Gardent and Kallmeyer, 2003) is the closest to our framework.

Obviously, everything one can do in the approach proposed in (Gardent and Kallmeyer, 2003) can be directly transformed into the approach presented here. An advantage of our approach is that semantic feature structures are linked to whole elementary trees and therefore they offer the possibility to define global features for elemenary trees. So far we have not exploited this in this paper but it obviously might be useful, for example for the MAXS and MINP features in section 6.

A problem of (Gardent and Kallmeyer, 2003) is that, as already mentioned, in order to do semantics using the feature structures in the syntax, an arbitrary number of possible feature values needs to be allowed, since the number of labels and individual variables occurring in a sentence cannot be limited in a general way. Consequently the number of possible feature structures is no longer finite and therefore, in contrast to standard FTAG (Vijay-Shanker and Joshi, 1988), the formalism is no longer equivalent to TAG. This means that semantic features are slightly different from those needed for syntax in terms of formal properties. Therefore, it is more appropriate to separate them from syntactic features and to

link them to whole semantic representations (i.e., to link them to whole elementary trees). This is what the approach described above does: instead of increasing the formal power of the syntactic formalism, the extra power needed for semantics is added to the semantic representations, i.e., to nodes in the derivation tree.

A further important difference is that we do not use explicit holes $h_1, h_2, \ldots$ besides propositional variables. Instead, the propositional variables that remain after having performed all unifications are understood as being holes in the sense of previous LTAG semantics approaches. This simplifies the formal framework considerably.

6 Sample analyses

6.1 Quantifiers

(6) everybody laughs

For quantificational NPs as in (6) we propose the analysis shown in Fig. 8. This allows us to obtain the scope constraints mentioned above: the NP proposition is in the restriction of the quantifier because of $\boxed{4} \geq l_3$. Furthermore, the following must be guaranteed: 1. the proposition to which a quantifier attaches must be in its nuclear scope and 2. a quantifier cannot scope higher than the next finite clause. The first constraint must result from the combination of the lower part of the quantifier (the NP tree) and the tree to which it attaches.[6] We introduce a feature MINP to pass the proposition of a tree to an embedded quantifier. The second constraint must result from the adjunction of the scope part of the quantifier. We use a feature MAXS ('maximal scope') that passes an upper limit for scope from a verb tree to an adjoining scope tree. E.g., see Fig. 8 for the analysis of (6). It leads to the following unifications: $\boxed{6} = \boxed{2}$ (adjunction of the scope part), $\boxed{1} = x$ and $\boxed{7} = l_1$ (substitution of the predicate-argument part, and $\boxed{3} = l_1$ (final top-bottom unification). The result is (7) which has just one disambiguation: $\boxed{2} \rightarrow l_2, \boxed{4} \rightarrow l_3, \boxed{5} \rightarrow l_1$.

(7)
$$\begin{array}{l} l_1 : \mathsf{laugh}(x), \\ l_2 : \mathsf{every}(x, \boxed{4}, \boxed{5}), l_3 : \mathsf{person}(x) \\ \boxed{2} \geq l_1, \boxed{2} \geq l_2, \boxed{4} \geq l_3, \boxed{5} \geq l_1 \end{array}$$

Note that this analysis of quantifiers differs crucially form what is proposed in (Gardent and Kallmeyer, 2003) where quantifiers do not have a separate scope part. This separate scope part allows us to account for various constraints for quantifier scope.[7]

[6] This is particularly clear in examples with quantificational NPs that are embedded in other quantificational NPs as considered in (Joshi et al., 2003). Here, the minimal nuclear scope of the embedded NP depends on the embedding NP and not on the verb tree.

[7] (Joshi et al., 2003) derive for example constraints for relative quantifier scope in so-called inverse linking configurations from the way the scope parts combine.

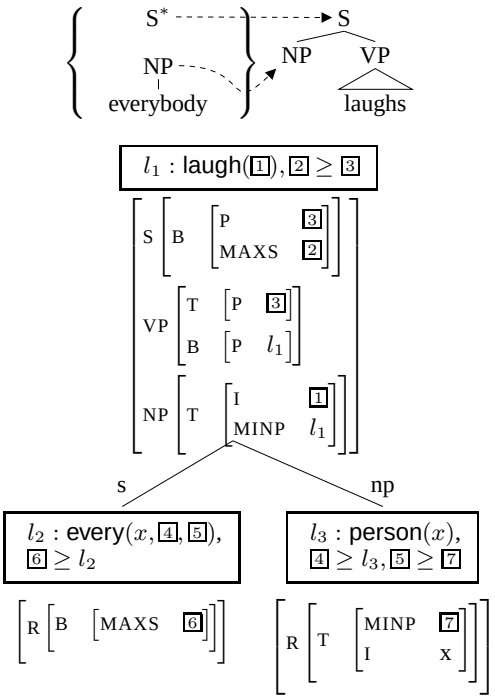


Figure 8: Analysis of (6)

6.2 Attitude verbs

(8) Mary thinks John laughs

The analysis of attitude verbs such as *thinks* in (8) is shown in Fig. 9. The propositional argument of *think* (variable $\boxed{3}$) is the MAXS value of the embedded verb (MAXS of the top of the foot node). This means that quantifiers or adverbs attaching to the lower verb cannot scope over *thinks*.[8] The adjunction leads to $\boxed{3} = \boxed{1}$.

(9) Mary thinks John likes everybody

In (9), wide scope of *everybody* over *thinks* should be disallowed. If its scope part attaches to the S node of *likes*, then the scope is blocked by the MAXS value of *likes*. Consequently, *everybody* cannot have scope over *thinks* because *thinks* takes the MAXS proposition of *likes* as its argument. However, we have to make sure that the scope part of *everybody* cannot attach higher, i.e., to *thinks*.

In general, we allow scope parts to adjoin higher. But, following (Joshi et al., 2003), the compositions must be such that one or more already derived trees or tree sets attach (by substitution or adjunction) to one single elementary tree. If only the NP tree of *everybody* attaches to *like*, there are only two possible continuations and both lead to an incorrect derivation for (9). The first possible

[8] Some counterexamples to finite clause boundness are analyzed nowadays as cases of illusive scope (Fox and Sauerland, 1996).

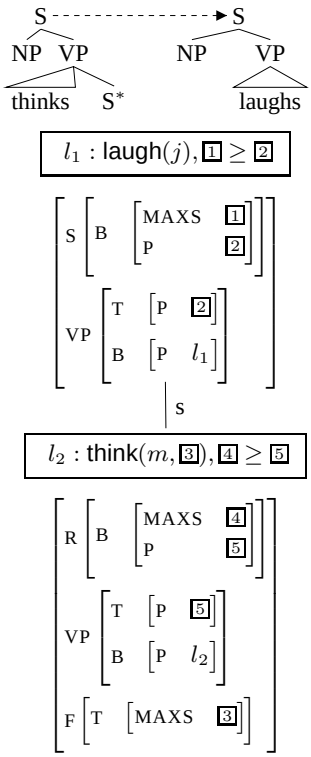


Figure 9: Analysis of (8)

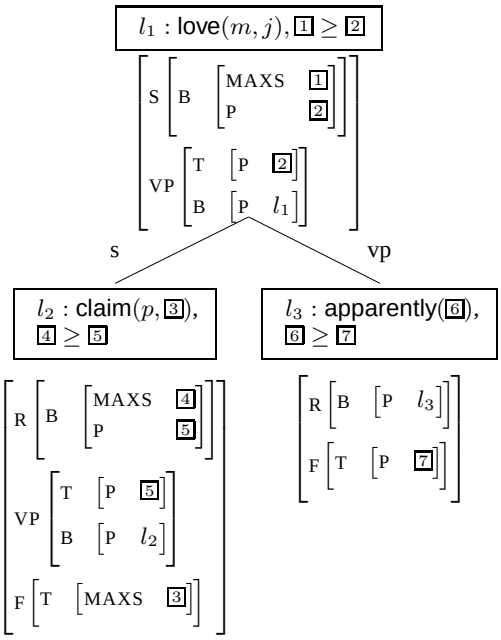


Figure 10: Analysis of (11b)

continuation is to adjoin the scope part of *everybody* to *thinks*. Then the derived *like* tree also must be added to *thinks* since it is part of the same derived tree set, i.e., *thinks* would have a substitution node instead of a foot node. This however is problematic for the analysis of long-distance dependencies in LTAG. The second possible continuation is that *thinks* attaches to *like* simultaneously with the lower *everybody* part. But then the scope part has to find some other node than the S node of *thinks* in order to attach to it. There is no other S node besides those coming from *thinks* or *like*, so this possibility does not work either. Consequently, one has to adjoin the scope part to the *like* S node.

6.3 Problems for derivation based semantics

Now let us come back to the examples (2) and (3) mentioned in the beginning, repeated here as (10) and (11):

(10) Who does Paul think John said Bill liked?

(11) a. Mary, Paul claims John seems to love
b. Paul claims Mary apparently loves John

For an analysis of (10) we refer to (Romero et al., 2004) in this volume. An analysis of (11b) is shown in Fig. 10, (11a) is analyzed in the same way. We analyze raising verbs similar to adverbs (see *sometimes* in Fig. 3). They are in a sense inserted between the top and bottom P values of the node to which they adjoin. They scope over the lower proposition. By unification, the proposition introduced by the topmost adverb/raising verb is the P value of the root of the verb tree which is below the MAXS proposition. Therefore, in (11b), the attitude verb *claim* takes scope over the adverb.

Furthermore, the problem of multiple modifiers as in (12) is also often discussed as an example where the TAG derivation tree does not give the semantic dependencies one needs (see, e.g., (Schabes and Shieber, 1994; Rogers, 2002)). These cases are difficult for a derivation tree based semantics because only the adjective that is closest to the modified noun attaches to the noun, all adjectives that are further to the left attach to the adjective on their right. However, all adjectives equally take the variable provided by the noun as their argument.

(12) roasted red pepper

As shown in Fig. 11, in our approach the arguments of the three predicates, *pepper*, *red* and *roasted* can be easily unified such that they all refer to the same individual.

7 Conclusion

In this paper we introduced an LTAG semantics framework based on the derivation tree. We use feature structure unification on the derivation tree as semantic composition operation, similar to the syntactic features on the derived tree that are used in TAG. Within this framework, we proposed an account of quantificational NPs, adverbs, raising verbs and attitude verbs, and we have shown that

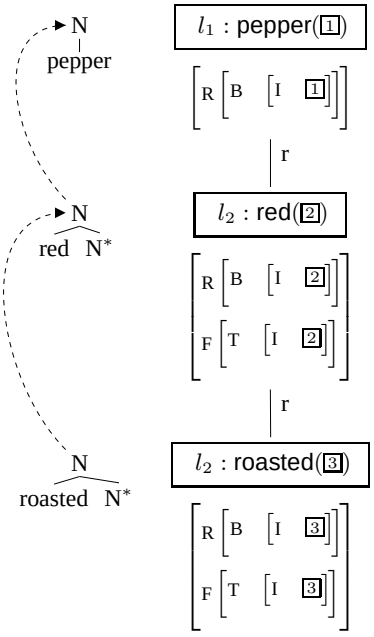


Figure 11: Analysis of (12)

we can analyze the examples considered in the literature as problematic for derivation tree based LTAG semantics approaches.

Acknowledgments

For many fruitful discussions of the framework and the analyses presented here we would like to thank Aravind K. Joshi and all members of the XTAG Group at the University of Pennsylvania. Furthermore, we are grateful to two anonymous reviewers for their useful comments.

References

Marie-Hélène Candito and Sylvain Kahane. 1998. Can the TAG Derivation Tree represent a Semantic Graph? an Answer in the Light of Meaning-Text Theory. In *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks, IRCS Report 98–12*, pages 25–28, University of Pennsylvania, Philadelphia.

Mark Dras, David Chiang, and William Schuler. 2004. On Relations of Constituency and Dependency Grammars. *Journal of Language and Computation*, 2(2):281–305.

Danny Fox and Uli Sauerland. 1996. Illusive Scope of Universal Quantifiers. In *Proceedings of NELS 26*, Amherst.

Anette Frank and Josef van Genabith. 2001. GlueTag. Linear Logic based Semantics for LTAG – and what it teaches us about LFG and LTAG. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG01 Conference*, Hong Kong.

Robert Frank. 1992. *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives*. Ph.D. thesis, University of Pennsylvania.

Claire Gardent and Laura Kallmeyer. 2003. Semantic Construction in FTAG. In *Proceedings of EACL 2003*, Budapest.

Mark Johnson. 1994. Computing with Features and Formulae. *Computational Linguistics*, 20(1).

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, Berlin.

Aravind K. Joshi and K. Vijay-Shanker. 1999. Compositional Semantics with Lexicalized Tree-Adjoining Grammar (LTAG): How Much Underspecification is Necessary? In H. C. Blunt and E. G. C. Thijsse, editors, *Proceedings ot the Third International Workshop on Computational Semantics (IWCS-3)*, pages 131–145, Tilburg.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible Composition in LTAG: Quantifier Scope and Inverse Linking. In Harry Bunt, Ielka van der Sluis, and Roser Morante, editors, *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*, pages 179–194, Tilburg.

Laura Kallmeyer and Aravind K. Joshi. 2003. Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation*, 1(1–2):3–58.

Laura Kallmeyer. 2002a. Enriching the tag derivation tree for semantics. In *Proceedings of KONVENS 2002*, pages 67 – 74, Saarbrücken, October.

Laura Kallmeyer. 2002b. Using an Enriched TAG Derivation Structure as Basis for Semantics. In *Proceedings of TAG+6 Workshop*, pages 127 – 136, Venice, May.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-Tree Grammars. In *Proceedings of ACL*.

James Rogers. 2002. One More Perspective on Semantic Relations in TAG. In *Proceedings of TAG+6 Workshop*, pages 118 – 126, Venice, May.

Maribel Romero, Laura Kallmeyer, and Olga Babko-Malaya. 2004. LTAG Semantics for Questions. In *Proceedings of TAG+7*, Vancouver.

Yves Schabes and Stuart M. Shieber. 1994. An Alternative Conception of Tree-Adjoining Derivation. *Computational Linguistics*, 20(1):91–124, March.

Matthew Stone and Christine Doran. 1997. Sentence Planning as Description Using Tree-Adjoining Grammar. In *Proceedings of ACL*, pages 192–205.

K. Vijay-Shanker and Aravind K. Joshi. 1988. Feature structures based tree adjoining grammar. In *Proceedings of COLING*, pages 714–719, Budapest.

Expanding Tree Adjoining Grammar to create Junction Grammar trees

Ronald Millett
BYU Linguistics
RonMillett@byu.edu

Deryle Lonsdale
BYU Linguistics
lonz@byu.edu

Abstract

Junction Grammar (JG) combines junction operators, multiple linked syntax/semantics trees, and flexible traversal algorithms. The multiple tree and flexible ordering characteristics of MC-TAG and other TAG extensions are somewhat analogous. This paper proposes that these similarities can be integrated to form a new approach, JG-TAG. Relevant aspects of both theories and the proposed new model are discussed in turn, and representative examples are sketched.

1 Introduction

This paper presents an enhanced version of Tree Adjoining Grammar (TAG) that can create trees based on Junction Grammar (JG), a linguistic theory that was proposed in the mid-1960's. The JG tree structures discussed in this paper represent syntactic/semantic structures with node joining operators, multiple linked trees and flexible tree traversal algorithms that recode the concepts of a sentence into a separate articulation tree. Adding junctions and multiple linked trees to Multiple Component TAG (MC-TAG) and other enhancements results in a formalism that can capture more sophisticated sentence relationships using fewer TAG trees than would otherwise be required. With the advantages that TAG's expanded domain of locality provides, an enhanced TAG-like system for JG could also provide an improved platform for computational linguistic applications of JG.

2 Theoretical background

2.1 Overview of Junction Grammar

Junction Grammar (JG) is a linguistic theory proposed in the 1960's (Lytle, 1971; Lytle, 1974; Lytle, 1985; Melby, 1985) that is still being pursued. The theory was developed as a reaction to early Transformational Grammar and challenged many of the basic assumptions that TG was based upon (Lytle, 1979). Early applications of JG have included machine translation (Gibb, 1970; Billings, 1972; Gessel, 1975; Lytle, 1975; Melby, 1978) including the development of a JG transfer language between source and target languages (Melby, 1974); speech synthesis (Melby, 1976; Millett, 1976); and second language instruction (Olsen and Tuttle, 1973). More recent efforts have involved PC-based grammar checking (Lytle and Mathews, 1986), automatic holistic scoring of essays (Breland and Lytle, 1990), and secondary school English grammar teaching (Millett and Lytle, 2004).

Junction Grammar challenged the notion that a basic grammar involves simple concatenation of strings via phrase structure rewrite rules. A fundamental premise of Junction Grammar is that JG operators (and their associated operands) constitute the basic building blocks of grammar. A well-defined process specifies operators and their appropriate application. The basic operator names and their symbols are: conjunction (&), adjunction (+), subjunction (∗), and interjunction (a combination of adjunction and subjunction). Two types of JG trees involving these operators are discussed in this paper: concept trees and articulation trees. Nodes in concept trees have a basic category label (N, V, A or P), predicate/phrase label (PN, PV, PA, or PP) or sentence/clause label (SN, SV, SA, or SP).

Figure 1 shows the JG concept tree for a simple sentence involving JG conjunction and adjunction. The former is used for coordinating conjunctions (e.g. *and*, *or*, *but*) and the arithmetic operators and their lexicalizations such as *plus* (hence &+). JG adjunction joins verbs or prepositions to their objects and phrases to their subjects.

The JG subjunction operator is perhaps the most flexible of the junction operators. Subjunction is used for determiners, quantifiers, complements, relative structures

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 163-170.

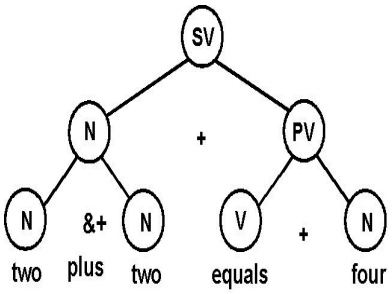


Figure 1: A binary application of the JG conjunction operator for the sentence *Two plus/and two equals four.* The SV node is a (verbal) sentence and the PV node is a (verbal) predicate.

and modifiers. It can also be subdivided into specialized operators that show the direction of information processing in the sentence. For an illustration, consider the following sentence:

(1) *It surprised me that the three children ate the vegetables that I cooked.*

and its associated JG concept tree in Figure 2. The word *three* selects three elements of the class *children*, and the junction has a single dash to the right in the direction of the noun class node. The determiner *the* retrieves a specific discourse-salient instance of three children. This retrieval function is shown by the equal sign to the left of the junction pointing to the determiner.

Figure 2 illustrates a second basic premise of Junction Grammar. JG allows for multiple tree structures unlike other theories that associate all trees structures under a single root. JG keeps complement structures such as *... that the children ate the vegetables ...* in the same tree structure with the subjunction operator, thus providing interoperability between sentence and noun. Using interjunction, JG allows for multiple intersecting trees for modifiers and relative structures via links, thus avoiding the need for empty categories and allowing a distinct contrast between complements and modifiers.

A third basic premise of JG is that trees can be ordered and lexicalized using flexible traversals guided by language-specific lexical ordering and hiatus (word deletion) rules. A JG concept tree represents syntactic/semantic information about the utterance that defines the syntactic and semantic relationships among the basic word concepts. The "goto" instructions and circled numbers in Figure 2 show lexical ordering rules that differ from the default left-to-right depth-first traversal algorithm. The complement clause *... that the three children ate the vegetables ...* is co-referent with the pronoun *it*, but the traversal of this clause is delayed until after the main clause is processed. Another ordering of the same sentence and same tree structure without lexicalizing the pronoun *it*, could be *That the three children ate the vegetables surprised me*. The JG tree would not change for that ordering, except that the pronoun *it* would be annotated as hiatused (another language-specific lexical rule in JG) by including parentheses.

The traversal processing for the trees in Figure 2 would thus be as follows:

1. Start processing at main SV node at top of figure.
2. Process left-to-right order to the subject N node and output *it* at the first N node of the rule $N * SV = N$.
3. Order discontinuously the complement $N * SV$ structure by going to ordering point #1, just above the PV node.
4. Process the PV left-to-right and output *surprised me*.
5. Now return to the unprocessed nodes in the subject by going to ordering point #2, by the embedded SV.
6. Order the SV left-to-right and output words *... that the three children ate the vegetables that ....* The relative pronoun *that* now triggers the processing of the subordinate SV structure. After going to the linked node, go to ordering point #3 at the top of the relative clause SV.
7. Process the relative clause in the default left-to-right order and output *I cooked*. The relative pronoun has already been marked as processed and at this point the entire tree has been processed.

In fact, the concept tree traversal specified above does not directly produce an output word string. Instead, following another foundational principle of JG, some concept tree information is recast into an articulation tree. The flexible traversal described above constructs such a tree, which encodes prosodic, phonological, and graphical information necessary for spoken or written language production. The basic JG operators in an articulation tree specify and relate breath groups and suprasegmental information; its nodes are of category H or W (for prosodic and lexical content respectively). Figure 3 shows concept and articulation trees for the following sentence:

(2) *Which job has Sally declined?*

Relevant traversal processing for the concept tree in Figure 3(a) is as follows, where lexicalization means mapping content appropriately into the articulation tree:

1. Start processing at the main SV node.
2. The ordering rule attached to the top node checks below the SV for a lexical entry with a $< +WH >$

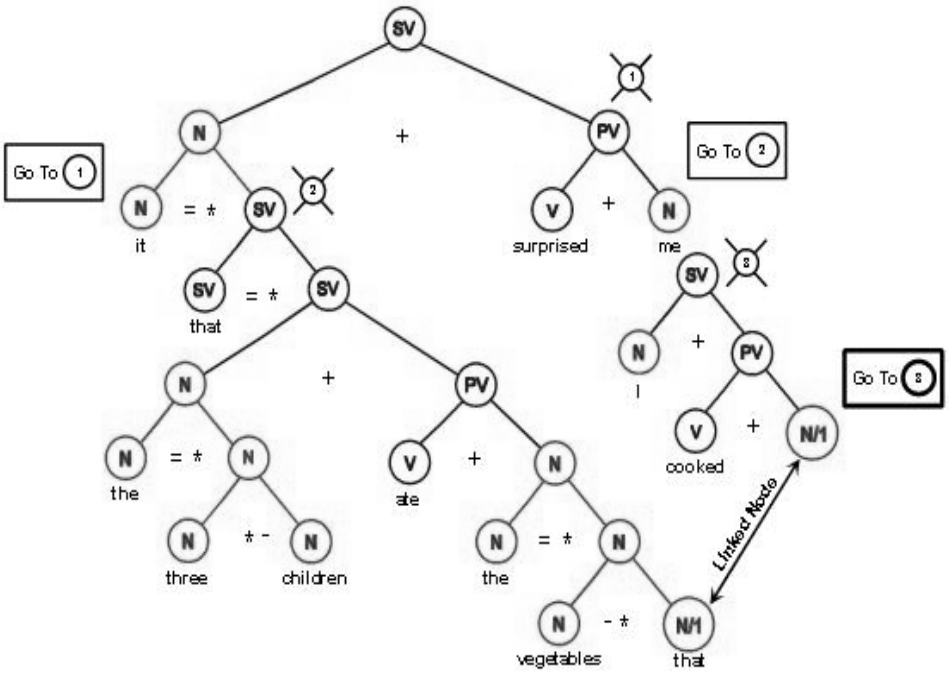


Figure 2: JG subjunction, adjunction and interjunction: a sentence with a relative clause shared across matrix and dependent clauses (via subjunction and adjunction respectively). Lexical ordering is also specified.

feature attached. Therefore, go to the N node dominating the N node with the *Which* lexical entry labeled with the ordering point #1.

3. Lexicalize *Which job* and then return back up the tree go ordering point #2.

4. In Junction Grammar trees the "modalizer" extending to the left of the PV level is the point where auxiliary verbs such as *has* are lexicalized.

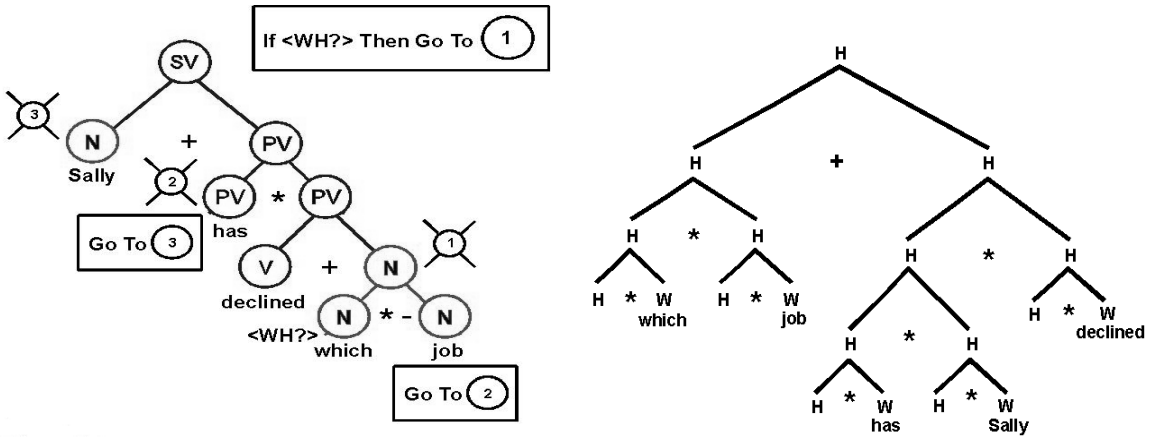
5. Go to ordering point #3 and continue using normal left-to-right order to finish processing the rest of the tree that has not yet been visited. *Sally declined* is lexicalized.

In combination with the basic junction operators discussed earlier, the basic JG rules and constraints are able to describe a wide variety of linguistic structures. Figure 4 shows a table of possible JG linguistic structures for a portion of the interjunction general rule template $X * X/1 = X$ and $Y + X/1 = PY$ or $PZ + X/1 = SZ$, where X varies over N, V, PV, and SV and Y varies over V and P and Z varies over V, A, P, and N.

This sketch of part of the JG theory focuses on only two of the four possible levels; consideration of the other levels is beyond the scope of this paper. It should be noted that JG parsers and/or generators have been implemented for a wide variety of languages including English, French, Spanish, German, Portuguese, Russian, Chinese, and Japanese.

2.2 Overview of TAG

Tree adjoining grammars (TAG) have provided a theoretical framework for linguistic description and natural language processing that has been shown to be superior to simply using rules of a context free grammar (CFG) due in large part to the extended context or "domain of locality" that TAG provides (Abeillé and Rambow, 2000; XTAG Research Group, 2001). With its lexicalized nature and its detailed syntax/semantics interface, LTAG (Lexical TAG) facilitates a straightforward representation of important data such as subcategorization frames for verb classes used in parsing. Appropriate usage of lexical entries is formalized by LTAG trees that not only derive a standard surface level parse tree but also a derivation tree that represents semantic and thematic role relationships for the sentence. Enhancements to TAG have included multicomponent TAG (MC-TAG) that allows for simultaneous linked operations on two or more trees into the derived tree to successfully derive parse trees for examples like "picture-NP extraction". To address the many com-



(a) (b)

Figure 3: Two JG trees for a sentence: the JG concept tree3(a), which is traversed to create the articulation tree 3(b).

binations of orders possible in languages such as German, a free order TAG (FO-TAG) variation was developed that did not force a strict left-to-right processing of all nodes of the derived tree (Becker et al., 1991; Rambow and Lee, 1994) Further enhancements of MC-TAG have included VMC-TAG that allows for non-simultaneous adjunction of multiple component trees to better describe free word order languages (Rambow and Lee, 1994). D-Tree Grammars (DTG) adjusted the basic TAG operations of substitution and adjunction to become subsertion and sister adjunction in order to separate complementation and modification operations and correct inaccuracies in the TAG derivation tree for topicization and to handle word order for wh- extraction sentences for Kashmiri (Rambow et al., 1995). In machine translation applications, synchronous TAG (S-TAG) is used to represent linked source and target language sets of trees that represent required transfer operations while translating between the two languages (Harbusch and Poller, 2000).

3 Comparative analysis

The most obvious similarity between JG and TAG is their use of multiple tree structures. TAG initial and auxiliary trees can define basic lexical options such as subcategorization frames for verbs along with their allowed auxiliary verbs and modifiers. JG represents modifier structures using subordinate tree structures. However, while the output of a TAG derivation is a standard tree diagram for a given sentence plus an associated derivation tree, the JG trees represent the final syntax/semantic representation of the sentence. JG applications have created JG trees by processing a grammar of allowed JG rules plus language-specific lexical ordering and other algorithms. Even though the JG interjunction rule pairs give some expanded domain of locality over single rules, JG has domain of locality limitations similar to context free grammars (CFG) and could greatly benefit from the mildly context sensitive grammar advantages that an LTAG approach could provide. TAG adjunction constraints and feature information are similar to JG lexical features that select basic JG rule groups via algorithms that might apply, for example, for a specific verb family. However, JG applications have relied more on specialized programming accompanying basic JG rules rather than being able to use forests of linked trees to implement lexicalized applications such as what a TAG approach would provide.

Because JG separates the syntax/semantics representation from the ordered words ready to articulate, many of the complexities of TAG trees can be simplified. The JG approach does put language-specific ordering and other lexical rules into algorithms that operate on the JG syntax/semantics trees, a deliberate tradeoff for not describing both syntax and order directly in tree structures. However, as the various TAG and variant systems have been developed, amazingly complex trees are needed to allow for all of the possible variations in word order for German, Korean or Kashmiri. A similar set of trees for JG parallel with a MC-TAG or DTG system would not require explicit encoding of multiple word order variations and hence the number of trees would be reduced.

Figure 5 summarizes some of these similarities and differences between JG and TAG.

3.1 JG-TAG

Because of their similarities, TAG and JG could conceivably be combined into a new JG-TAG approach. A parser

Subjunction rule in main tree	Adjunction rule in subordinate tree	Sample output text
N ∗ N/l = N	V + N/l = PV	The mouse that the cat chased (got away).
N ∗ N/l = N	PV + N/l = SV	The cat that chased the mouse (lives next door).
N ∗ N/l = N	PA + N/l = SA	The tall elephant . . .
N ∗ N/l = N	P + N/l = PP	The ladder upon which (I was standing) . . .
N ∗ N/l = N	PP + N/l = SP	The boy from Atlanta . . .
N ∗ N/l = N	PN + N/l = SN	Harry, the class clown, . . .
V ∗ V/l = V	PA + V/l = SA	He looked up (the number).
V ∗ V/l = V	PP + V/l = SP	(He) dropped (the paint) into the burning cauldron.
PV ∗ PV/l = PV	PA + PV/l = SA	(He) went quickly (into the city).
PV ∗ PV/l = PV	PP + PV/l = SP	(The city) needed water for survival.
SV ∗ SV/l = SV	PA + SV/l = SA	Unfortunately, . . .
SV ∗ SV/l = SV	PP + SV/l = SP	Without a doubt, . . .

Figure 4: Some basic JG relationships via interjunction; nodes are linked via the /1 annotation.

Multiple linked trees:

- JG: syntax/semantics tree representation(s) for a sentence
- TAG: grammar is represented in multiple trees but output is single parse tree plus derivation tree

Expanded domain of locality:

- JG: interjunction rule pairs but still context-free
- TAG: initial and linked auxiliary trees with adjunction constraints, mildly context sensitive with LTAG

Syntax, semantics and language-specific ordering:

- JG: separation of syntax/semantics from lexical ordering; more complex algorithms attached to rules but with simplified trees
- TAG: variations on TAG for complex word order but more complex and numerous trees required

Figure 5: JG and TAG: contrasts involving similar features.

could then be developed following TAG's XTAG model. Because JG representations have separate trees (Lytle, 1979) for syntax/semantics vs. ordered output words, such a system would be simplified from standard XTAG with fewer trees needed for a complete grammar.

The first enhancement needed to create a JG-TAG system would be to attach a junction operator to each nonterminal node. We will represent this junction attached right to the end of the node label. Because the subjunction operator is the "∗" character, JG-TAG would also need to change the foot node sign from a "∗" to a "#". Even though a subjunction operator would not be attached to a foot node, we will also propose another use of the "#" in another enhancement. Generalizing junctions allow for creation of an arbitrary number of conjuncts without spawning new nonterminal nodes. Figure 6 shows how *bears* could be added to the conjunction rule for *lions and tigers*. In this case the auxiliary tree would use the foot node indicator "#" attached to the non-terminal N node. The output tree shows the result of this n-ary adjunction capability. The lexicalization of *and* in these sentences is assured by the conjunction operator for the noun nonterminal node.

Another enhancement for JG-TAG is to allow for subordinate tree structures and their creation and processing. This is where the capabilities of MC-TAG are essential. One auxiliary tree would create the relative pronoun node on the side of the main clause and the other auxiliary tree would create the mirror node on the subordinate relative clause side. To avoid confusion with subscripts that are used in trace nodes in standard theory trees, we propose that the link number between these mirrored linked nodes be a superscript. Figure 7 illustrates this process with the sample working tree for the sentence *The cat caught a mouse*. The two MC-TAG trees would operate on the working tree by first adjoining with the node for cat and

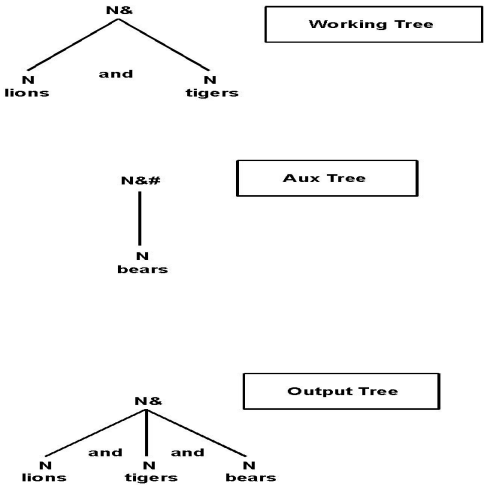


Figure 6: Adjoining trees in JG-TAG to form a three-node conjunction concept tree.

then creating the subordinate relative clause, linked by the superscript "1" with the main tree. The tree that adjoins into the main clause to create the $N * N = N$ rule would be defined as an auxiliary tree and the subordinate clause would be defined as an elementary tree in order to create the new tree. The SV node of the main clause would also be marked as the starting point for processing the resulting JG tree forest. The overall process would be as follows:

1. The left JG-TAG auxiliary tree will left-adjoin with the *cat* node to create a $N * N/1 = N$ subtree with *cat* and *that* associated with the nodes.

2. The right JG-TAG elementary tree *that/1 the dog chased* will then be added to the working tree forest and the relative pronoun noun node with the superscript link stays linked with the N/1 node in the main working tree.

3. The resulting new working tree has the main tree, where processing of the tree would begin, marked as the start tree (*the cat that/1 caught a mouse*).

4. The subordinate tree is accessed during tree traversal by going from the main clause mirrored link to the subordinate clause.

3.2 JG-TAG: prospects and challenges

Early NLP applications of JG used junction rules plus specialized programming to examine rule contexts for triggering language-specific transfer or lexical rules. Currently JG tree processing programs are limited to a proprietary control language used inside recent applications. A JG-TAG system would allow a standard XTAG-like system to be developed that could provide a parsing capability for JG trees, allowing wider access and easier comparisons with existing systems using other theories.

Another benefit of such a system would be its ability to represent a greater portion of the grammar of a language with fewer TAG trees. For example, if the JG concept tree in Figure 3 were represented in JG-TAG, only one tree would be needed to cover both cases where the substituted noun nodes include a "+WH" feature or not. This same tree could also work for nonstandard orderings of an SVO sentence as OSV.

A JG-TAG system would provide an excellent framework to represent subcategorization frames for different verb classes using supertag trees. TAG has always excelled in providing context sensitivity to a basic rule system and a lexicalized JG grammar implementation would allow JG structures that have previously been represented programmatically to be described in a more easily visualized and maintainable data structure format. The verb class JG-TAG trees would also simplify the lexical rules by attaching them to specific verbs and allowing them to be limited to the context of a specific verb.

One of the exercises in creating such a system would involve the format of lexical rules that would be attached to the JG-TAG trees. Each JG-like rule in the tree specifies left-to-right, right-to-left or discontinuous ordering. Recall that the JG approach involves in-situ wh-elements and a specific traversal order without creating target nodes for movement. Thus the algorithm for deciding traversal would reflect, but not implement, movement. The documentation and implementation papers for the JG ordering algorithms and transfer language used in an early machine translation project could be a good starting point for a JG-TAG system (Melby 1974, Gessel 1975).

Another challenge would be matching and using features attached to JG nodes with the TAG feature capabilities. TAG unification features that prevent more than one tense-bearing verb to be attached usually would be implemented by JG lexical agreement rules. However, the feature unification approach from TAG provides a straightforward manner to keep track of main and auxiliary verbs and their inflections as a sentence is created from the tree.

Mandatory, optional and null adjunction constraints allow the relationships between the various TAG tree sets to be carefully defined, linked together and maintained. Expert rule systems generally need these kinds of constraints in order to assure tractable development and maintenance. These same capabilities would be very advantageous to link together JG tree fragments that would define a working grammar for a particular language.

The power of the MC-TAG trees that encapsulate semantic relationships would then output not just a surface ordered derived tree but an order-independent syntax/semantics representation less dependent on the

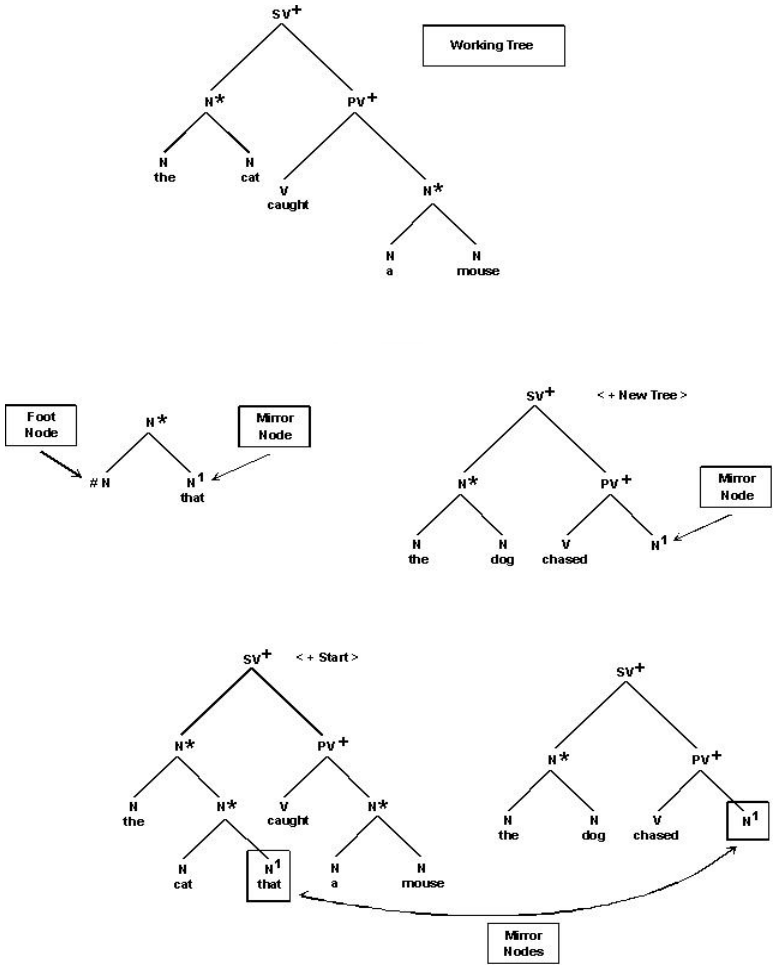


Figure 7: JG-TAG MC trees for a relative clause; the top working tree is operated on by the JG-TAG multicomponent tree that would attach the relative structure at the appropriate node and subordinate tree. The N node is structure-shared between the trees; superscripts specify inter-tree links.

derivation tree for semantic relationships. The JG trees are not at as low a semantic level as the derivation tree but provide structure related to the original utterance (e.g. active vs. passive) and are very rich in specific syntax and semantic relationships (e.g. themes and verb classes with thematic roles (Millett, 1975)) between the concepts of the utterance. Comparative and quantifier structures have a particularly rich semantic structure in JG (Lytle 1985) and a JG-TAG system could facilitate comparison of the capabilities of a JG-based text-understanding application to other standard approaches. A JG-TAG system could also provide a standardized application and coding framework for using Junction Grammar.

4 Conclusions

As TAG formalisms have been applied to natural languages, their advantages over context-free phrase structure rules have become more apparent. Many useful refinements to the basic TAG formalism have supported a wide variety of structures. Meanwhile JG embodies rather different assumptions than do traditional theories: a separation of linguistic data via conceptual and articulation trees, junction operators on non-terminal nodes, multiple-linked tree structures, and flexible traversal of lexical rules. The appreciable overlap of approaches with TAG and JG has prompted this discussion on combining the benefits of both theoretical systems to represent and process Junction Grammar trees. The advantages of the mildly context sensitive lexical JG-TAG system proposed in this paper can expand the domain of locality for JG trees, simplify lexical rules by attaching them to supertag class trees and draw on the extensive NLP experience using TAG based systems to benefit JG. TAG could likely also benefit from junctions, ordering, and multiple tree enhancements from Junction Grammar.

Acknowledgement We wish to thank Eldon Lytle for comments on drafts of this paper. All errors or inaccuracies are the authors'.

References

Anne Abeillé and Owen Rambow. 2000. Tree Adjoining Grammar: An overview. In Anne Abeillé and Owen Rambow, editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*, pages 1–68. CLSI Publications.

T. Becker, A.K. Joshi, and O. Rambow. 1991. Long distance scrambling and tree adjoining grammars. In *EACL-91: Papers presented to the 5th Conference of the European Chapter of the Association for Computational Linguistics*, Berlin.

Floyd H. Billings. 1972. Proposals for ordering well-formed syntactical statements. Master's thesis, Brigham Young University.

Hunter M. Breland and Eldon G. Lytle, 1990. *Computer-assisted writing skill assessment using WordMAP*. American Educational Research Association and the National Council on Measurement in Education, Boston, MA. ERIC Document Reproduction Service No. ED 317 586.

Brian Gessel. 1975. The formulation and computer adaption of synthesis grammars. Master's thesis, Brigham Young University.

Daryl K. Gibb. 1970. An application to mechanical translation of a variable recursive algorithm based on the operations of union and intersection. Master's thesis, Brigham Young University.

Karin Harbusch and Peter Poller. 2000. Non-isomorphic synchronous tags. In Anne Abeillé and Owen Rambow, editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*, pages 147–166. CSLI Publications.

Eldon G. Lytle and N. C. Mathews, 1986. *Field Test of the WordMAP Writing Aids System*. Panaca, NV, May. Official publication of the Lincoln County School District.

Eldon G. Lytle. 1971. *Structural Derivation in Russian*. Ph.D. thesis, University of Illinois.

Eldon G. Lytle. 1974. *A Grammar of subordinate structures in English*. Mouton, The Hague.

Eldon G. Lytle. 1975. Junction Grammar as a base for natural language processes. *American Journal of Computational Linguistics*. Microfiche no. 26.

Eldon G. Lytle. 1979. Junction Grammar: Theory and application. In William C. McCormack and Herbert J. Izzo, editors, *Sixth LACUS Forum*, pages 305–343. Hornbeam Press.

Eldon G. Lytle. 1985. Come on up. In Adam Makkai and Alan K. Melby, editors, *Linguistics and Philosophy: Essays in honor of Rulon S. Wells*, volume 42 of *Amsterdam studies in the theory and history of linguistic science: Current Issues in Linguistic Theory (IV)*. John Benjamins, Amsterdam.

Alan K. Melby. 1974. Formulating and testing syntactic transfers. Master's thesis, Brigham Young University.

Alan K. Melby. 1976. *Computer Generated Intonation in Synthetic Speech*. Ph.D. thesis, Brigham Young University.

Alan K. Melby. 1978. Design and implementation of a machine-assisted translation system. In *Proceedings of the Seventh International Conference on Computational Linguistics*, Bergen, Norway.

Alan K. Melby. 1985. Generalization and prediction of syntactic patterns in Junction Grammar. In Adam Makkai and Alan K. Melby, editors, *Linguistics and Philosophy: Essays in honor of Rulon S. Wells*, volume 42 of *Amsterdam studies in the theory and history of linguistic science: Current Issues in Linguistic Theory (IV)*. John Benjamins, Amsterdam.

Ronald P. Millett and Eldon G. Lytle. 2004. Language included: Book 1: Sentence pictures, picturing grammar, picturing sentences. eBook format.

Ronald P. Millett. 1975. Junction Grammar verb classes in Spanish. Technical report, Brigham Young Univeristy. Junction Grammar Papers.

Ronald P. Millett. 1976. A pitch contour generating algorithm based on a Junction Grammar linguistic model. Technical report, Brigham Young Univeristy. Junction Grammar Papers.

Royden S. Olsen and David M. Tuttle. 1973. The effect of 'language trees' on the acquisition of generative capacity in a second language. Technical report, Brigham Young Univeristy. Junction Grammar Papers.

Owen Rambow and Y. S. Lee. 1994. Word order variation and tree-adjoining grammars. *Computational Intelligence*, 10:386–400.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-tree grammars. In *Proceedings of ACL-95*.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Context-free Approximation of LTAG towards CFG Filtering

Kenta Oouchida[†] **Naoki Yoshinaga**[†] **Jun'ichi Tsujii**[†*]
†University of Tokyo * CREST, JST (Japan Science and Technology Agency)
{oouchida, yoshinag, tsujii}@is.s.u-tokyo.ac.jp

Abstract

We present a method to approximate a LTAG grammar by a CFG. A key process in the approximation method is finite enumeration of partial parse results that can be generated during parsing. We applied our method to the XTAG English grammar and LTAG grammars which are extracted from the Penn Treebank, and investigated characteristics of the obtained CFGs. We perform CFG filtering for LTAG by the obtained CFG. In the experiments, we describe that the obtained CFG is useful for CFG filtering for LTAG parser.

1 Introduction

Recently, lexicalized grammars such as Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) have attracted much attention in practical application context (Deep Thought Project, 2003; Kototoi Project, 2001; Kay et al., 1994; Carroll et al., 1998). However, inefficiency of parsing with those grammars have prevented us from adopting them for practical usage. Especially in the LTAG framework, although many studies proposed parsers that are theoretically efficient (Vijay-Shanker and Joshi, 1985; Schabes and Joshi, 1988; van Noord, 1994; Nederhof, 1998), we do not attain any practical LTAG parser that runs efficiently with large-scale hand-crafted grammars such as the XTAG English grammar (XTAG Research Group, 2001).

Yoshinaga et al. (Yoshinaga et al., 2003) demonstrated that a drastic speed-up of LTAG parsing can be achieved when a LTAG grammar is compiled into a HPSG (Yoshinaga and Miyao, 2002) and a CFG filtering technique for HPSG-Style grammar (Kiefer and Krieger, 2000; Torisawa et al., 2000) is applied to the obtained HPSG. In experiments with the XTAG English grammar, they found that an HPSG parser with CFG filtering (Torisawa et al., 2000) outperformed a theoretically efficient LTAG parser (Sarkar, 2000) in terms of empirical time complexity. Although their approach does not guarantee the theoretical bound of parsing complexity, $O(n^6)$ for a sentence of length n, the empirical results of their CFG filtering are still satisfactory.

In this paper, we propose a novel context-free approximation method for LTAG by reinterpreting the method by Yoshinaga et al. in the context of LTAG parsing. A fundamental idea is to enumerate partial parse results that can be generated during parsing. We assign CFG nonterminal labels to the partial parse results, and then regard their possible combinations as CFG rules.

In order to investigate the characteristics of CFGs produced by our method, we applied our method to two kinds of LTAG grammars. One is the XTAG English grammar, which is a large-scale hand-crafted LTAG, and the other is LTAG grammars extracted from Penn Treebank Wall Street Journal by the grammar extraction method described in (Miyao et al., 2003). Then, we compare parsing speed of a CKY parser using the obtained CFG with parsing speed of an existing LTAG parser.

The remainder of the paper is organized as follows. Section 2 introduces background of our research. Section 3 describes our approximation method. Section 4 reports experimental results with the two kinds of LTAG grammars.

2 Background

2.1 Lexicalized Tree-Adjoining Grammar (LTAG)

An LTAG consists of a set of tree structures, which are assigned to words, called *elementary trees*. A parse tree is derived by combining elementary trees using two grammar rules called *substitution* and *adjunction*. Figure 1 shows elementary trees for *"I"*, *"run"* and *"can"*, and depicts how they are combined by substitution and adjunction.

Substitution replaces a leaf node of an elementary tree by another elementary tree whose root node has the same label as the leaf node. In Figure 1, the leaf node labeled

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 171-177.

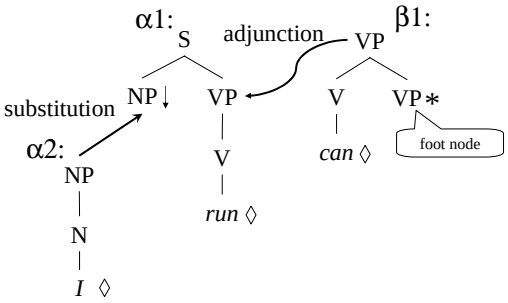


Figure 1: LTAG: elementary trees, substitution and adjunction

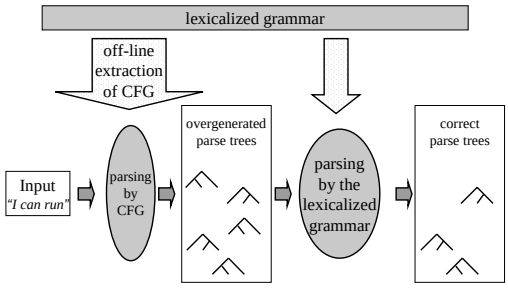


Figure 2: CFG filtering

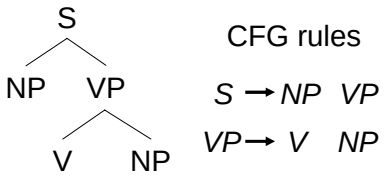


Figure 3: The existing CF approximation for LTAG

"NP" of α_1 is replaced by α_2, which has a root node labeled "NP."

Adjunction replaces an internal node of an elementary tree by another elementary tree whose root node and one leaf node called a *foot node* have the same label as the internal node. In Figure 1, the internal node labeled "VP" of α_1 is replaced by β_2, which has a root node and a foot node labeled "VP."

2.2 CFG filtering

CFG filtering (Harbusch, 1990; Maxwell III and Kaplan, 1993; Torisawa and Tsujii, 1996) is a parsing scheme that filters out impossible parse trees using a CFG extracted from a given grammar prior to parsing. In CFG filtering, we first perform an off-line extraction of a CFG from a given grammar, (*Context-free (CF) approximation*). By using the obtained CFG we can compute efficiently the necessary condition for parse trees the original grammar could generate. Parsing using the obtained CFG as a filter comprises two phases (Figure 2). In the first phase, we parse a sentence by the obtained CFG. In this phase, the necessary condition represented by the CFG acts as a filter of parse trees. In the second phase, using the whole constraints in the original grammar, we examine the generated parse trees, and eliminate overgenerated parse trees.

The performance of parsers with CFG filtering depends on the degree of the CF approximation (Yoshinaga et al., 2003). If CF approximation is good, the number of overgenerated parse trees is small. Thus, the key to achieve efficiency in LTAG parsing is to maintain grammatical restrictions in CFG as efficiently as possible. The more of the grammatical constraints in the given grammar the obtained CFG captures, the more effectively we can restrict the search space.

There are existing CFG filtering techniques for LTAG (Harbusch, 1990; Poller and Becker, 1998). These techniques extract CFG rules by simply dividing elementary trees into branching structures as shown in Figure 3. Since the obtained CFG can capture only local constraints given in the elementary trees, we must examine many global constraints in the second phase.

CFG filtering techniques have also been developed for HPSG (Torisawa and Tsujii, 1996; Torisawa et al., 2000; Kiefer and Krieger, 2000). CFG rules are extracted by applying grammar rules to lexical entries and by enumerating partial parse results (*sign*) that can be generated during parsing (in Figure 4). The obtained CFG can capture global constraints given in the lexical entries, because the generated partial parse results preserve the whole constraints given in the lexical entries.

As Yoshinaga et al. demonstrated using HPSG-style grammar converted from LTAG, finite enumeration of partial parse results produces a better CFG filter than the existing CF approximation for LTAG because of its ability to capture the global constraints. In the paper, we re-interpret CF approximation of HPSG by Yoshinaga's method (Yoshinaga et al., 2003).

3 CF Approximation algorithm for LTAG

In this section, we describe an algorithm of our CF approximation of LTAG. In the following, we first describe an approximation of LTAG which consists only of single-anchored elementary trees. We then describe an approximation of general LTAG which includes multi-anchored elementary trees.

In Section 3.1, we introduce a basic idea in our method. In Section 3.2, we explain our method in detail. In Section 3.3, we explain the way of applying our method to LTAG which comprises multi-anchored elementary trees.

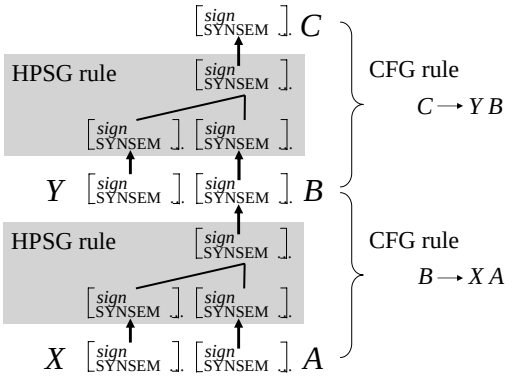


Figure 4: The existing CF approximation for HPSG

3.1 Basic Idea

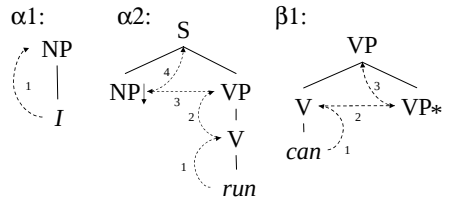
The fundamental idea of our approximation method is to enumerate partial parse results that can be generated during parsing. We obtain a CFG by assigning CFG non-terminal labels to the partial parse results, and regarding bottom-up derivation relationships between the partial parse results as CFG rules.

By recursively applying substitution and adjunction to elementary trees, we enumerate partial parse results derivable by LTAG. We adopt one of the existing mode, *head-corner traversal* (van Noord, 1994) (Figure 5), to recursively apply grammar rules.

In the first step of head-corner traversal, an elementary tree is taken as input and a directed path from an anchor node called *head-corner* to the root node is defined in a certain manner. The path traverses along all the nodes in the elementary tree. Then, grammar rules are incrementally applied to each node along the path.

We assign a non-terminal label of CFG to a subtree. A labeled subtree must include all information for enumeration. We determine this subtree as follows. A tree is divided into two parts, at the node to which we are applying a grammar rule (Figure 6). The "lower" part of the tree is a subtree below the node to which we are applying a grammar rule. The "upper" part consists of the nodes to which we will apply grammar rules in the rest of enumeration. We need only the "upper" part of the tree that includes all information necessary in the rest of the enumeration process. In this paper, we call the node to which we are applying a grammar rule a **processing node**, and we call the upper part of a tree an **active partial tree**. CFG non-terminal labels are assigned to each active partial tree.

By assigning CFG non-terminals to generated active partial trees, we obtain CFG rules as bottom-up derivation relationships between them. In Figure 7, the following CFG rules are obtained: $G \rightarrow A\ F$, $F \rightarrow E$, $E \rightarrow D\ E$, $D \rightarrow B$ and $E \rightarrow C$.



The application possibility of a grammatical rule is checked according to the path shown by the dotted line.

Figure 5: head-corner traversal

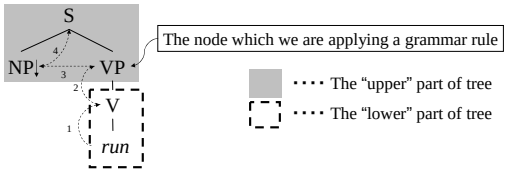


Figure 6: Division of a tree into two parts

3.2 Algorithm

Table 1 shows pseudo-code of our approximation algorithm. The algorithm takes LTAG G as input and outputs a CFG G'.

We start by explaining the top-level function `''extract_cfg_from_ltag.''` The function iteratively picks up two active partial parse trees from the set of active partial trees generated so far, and applies possible grammar rules. Whenever a new active partial parse tree is generated, we assign a new CFG non-terminal label and add it to the set. In case that new partial parse results have not been added during one iteration, we exit with G', which is the resulting CFG.

The function `''apply_rules''` applies the grammar rules to two active partial trees, and change the processing node to the next node. We apply unary rule in line 5, substitution in line 8, and adjunction in line 12.

Let us consider the extraction of CFG from LTAG defined in Figure 1. Figure 7 shows the extraction process. The initial active partial trees A, B and C originate from α_1, α_2, and β_1. In the first iteration in the function `''extract_cfg_from_ltag''`, two partial active trees, D and E, and two CFG rules, $D \rightarrow B$ and $E \rightarrow C$ are extracted. In the second iteration, one partial active tree, F and two CFG rules, $E \rightarrow D\ E$ and $F \rightarrow E$ are extracted. In the third iteration, one partial active tree, G, and one CFG rule, $G \rightarrow A\ F$ are extracted.

When substitution is applied to an active partial tree, the size of the parent's active partial tree is smaller than child's active partial trees. Thus, the number of generated active partial trees is finite, and the number of non-terminal labels in the obtained CFG is finite as well. In other words, if the CFG rules comprise only substitutions,

Table 1: The pseudo code of our algorithm

```
INPUT: G /* LTAG */
OUTPUT: G' /* CFG */

T_n:  /* a set of all active partial trees for nth iteration generated so far */
NT_n:  /* a set of new active partial trees for nth iteration */

Initialize:
T_n := φ
NT_1 := etree(G)
G' := φ
n := 1

1: procedure extract_cfg_from_ltag(G)
2: begin
3:   while ( NT_n ≠ φ )
4:     T_n := T_{n-1} ∪ NT_n
5:     foreach nt_n in ( NT_n )
6:       foreach t_n in ( T_n )
7:         NT := apply_rules( t_n, nt_n )
8:         NT_{n+1} = NT ∪ NT_{n+1}
9:       end foreach
10:    end foreach
11:    n++
12:  end while
13:  return G'
14:end extract_cfg_from_ltag

1: procedure apply_rules(t_1, t_2)
2: begin
3:   NT := φ
4:   if ( sibling( c_node( t_1 ) == nil ) ) /* if we cannot apply grammar rules */
5:     NT = unary( t_1 )
6:     G' = make_rule( t_1, NT ) ∪ G'
7:   else if ( sibling( c_node( t_1 ) ) == ''subst'' ) /* if we can apply substitution */
8:     NT = substitute( t_2, t_1 )
9:     G' = make_rule( t_2, t_1 ) ∪ G'
10:  else if ( sibling( c_node( t_1 ) ) == ''foot'' ) /* if we can apply adjunction */
11:    if ( depth_foot( t_1 ) == 1 || count_adjoing( t_1 ) <= LIMIT )
12:      NT = adjoin( t_1, t_2 )
13:      G' = make_rule( t_2, t_1 ) ∪ G'
14:    if ( depth_foot( t_1 ) >= 2 && count_adjoing( t_1 ) > LIMIT )
15:      NT = *
16:      G' = make_rule( t_2, t_1 ) ∪ G'
17:    end if
18:  end if
19:  return NT
20:end apply_rules
```

etree:	To return the elementary trees with head-corner paths.
c_node:	To return the processing node of the argument.
unary:	To return an active partial tree with the node which we will apply the grammar rules after.
make_rule:	To return the CFG rule of arguments
substitute:	To apply the rule and to return new active partial tree, if we can apply the grammar rule of substitution to the arguments,
adjoin:	To apply the rule and to return new active partial tree, if we can apply the grammar rule of adjunction to the arguments.

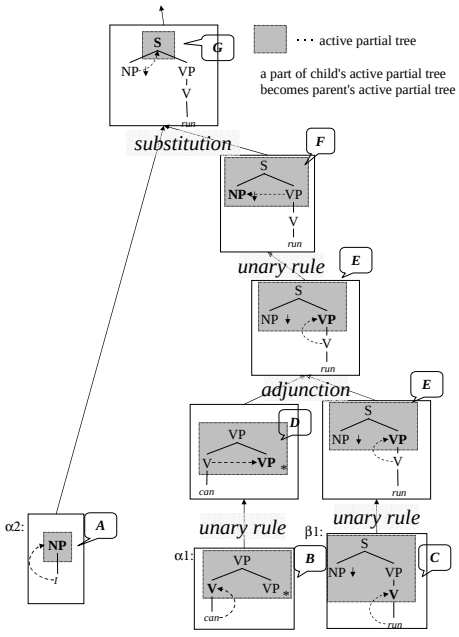


Figure 7: Approximation of LTAG to CFG

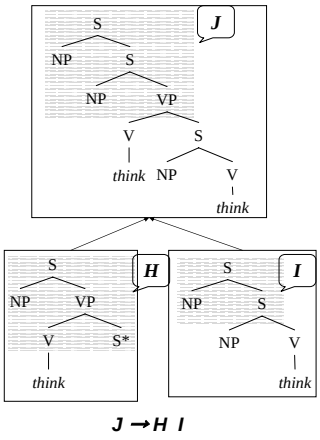


Figure 8: adjunction:the combination of two active partical trees

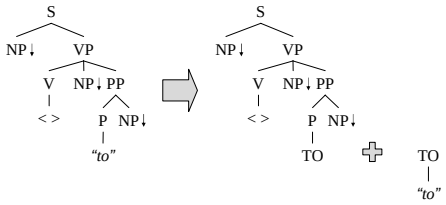


Figure 9: compiling XTAG English grammar

LTAG can be converted to CFG in finite size.

We must be careful about the depth of a foot node of an auxiliary tree when adjunction is applied to an active partial tree. If the depth of a foot node is one, active partial tree becomes the same as one of the two active partial trees to which adjunction are applied. If the depth of a foot node is two or more, parent's active partial tree takes a form of a combination of two active partial trees (Figure 8). This means that the number of active partial trees increases infinitely, if there are some auxiliary trees with a foot node at depth two or more.

In order to prevent the infinite increase of active partial trees, we count the number of the applications of adjunction which generates new active partial trees, and assign a special non-terminal "$*$" to active partial trees when the number of the applications reach a certain threshold. We then add CFG rules, $* \rightarrow X *$ and $* \rightarrow * X$ for all non-terminal labels "X" in order to guarantee that the resulted CFG can generate all parse trees that LTAG can generate. By using these rules, resulted CFG always generate parse trees which are derivable by the given grammar. Thus the obtained CFG can be used as a filter.

3.3 Extention to LTAG including multi-anchored trees

Our method can be applied to LTAG with elementary trees which contain only one anchor. It is the reason that the path from a head node to a root node becomes settled uniquely. The above approximation algorithm a handle general LTAG with multi-anchored trees by simply converting those trees into single anchored trees. When a grammar includes multi-anchored elementary trees, we simply replace an anchor node of them by a node to which can be applied a grammar rule of substitution (e.g. *TO* in Figure 9), and add a new elementary tree (e.g. *TO-"to"* in Figure 9).

4 Experiments

In order to observe the characteristics of CFG obtained by our method, we performed three experiments. In Section 4.1, we apply our method to the XTAG English grammar. In Section 4.2, we apply our method to LTAG grammars of various size extracted from a corpus, and investigate the relation between the size of LTAG grammars and the specification of the obtained CFG. In Section 4.3, we examine the characteristics of the obtained CFG in terms of the parsing speed, and compare the parsing speed of a CKY parser using the obtained CFG with the parsing speed of an existing LTAG parser.

4.1 Experiment on threshold value of adjunction

We applied our algorithm to the XTAG English grammar. In Table 2, we show the obtained CFG approximation of the XTAG English grammar. In this experiment, we varied threshold value of times of adjunction, which generates a new active partial tree, 0.

Table 2: approximating the XTAG English grammar by CFG

# of elementary trees of XTAG	924
# of terminal labels	924
# of non-terminal labels	2,779
# of rules	31,503

Even if the threshold is small, an obtained CFG is useful for parsing, because we hardly perform an adjunction using an auxiliary tree with a foot node with the depth of two or more in LTAG parsing. The maximum number of the possible rules is $2,779 \times (924 + 2,779) + 2,779 \times (924 + 2,779) \times (924 + 2,779) = 370338,116,519,448$. Our method produced 31,503 rules (about 0.00008% of all possible ones). Thus our method is efficient with respect to avoiding meaningless increase of the number of CFG rules. The small percentages means that obtained CFG is excellent as a filter.

4.2 Experiment on Various Size of LTAG

We extracted LTAG grammars from Section 02-06, 02-11, 02-16, and 02-21 of Penn Treebank Wall Street Journal (Table 3), and applied our method to the LTAG grammars.

We investigated the relation between the size of terminal labels, the size of non-terminal labels, and the size of CFG rules (Table 4). The increase of the number of non-terminal labels is slower than the increase of the number of terminal labels. Thus, our method is applicable for a larger LTAG than the LTAG which we used for this experiment.

4.3 Comparing our method to an existing LTAG parser

We investigated parsing performance of the obtained CFG.

We selected sentences which consists of 15 or less words from Section 23 of Penn Treebank Wall Street Journal, and experimented on parsing the sentences by two ways. One way is an existing LTAG parser with the XTAG English grammar. The other way is a CKY parser with the obtained CFG from the XTAG English grammar with threshold 1. The reason why we used the sentences which consists of 15 or less words is that existing LTAG parser is too slow to parse longer sentences.

Figure 10 and Figure 11 show the performances of an existing LTAG parser (Sarkar, 2000) and a CKY parser with the obtained CFG respectively. The CKY parsing is fast enough to employ a CFG filter. For longer sentences, it is expectable that our algorithm is effective.

5 Concluding Remarks and Future direction

In this paper, we showed an approximating method of LTAG by CFG. A specification of a CFG obtained from the XTAG English grammar shows that our method is efficient in the number of CFG rules. In addition, we compared parsing performance between the existing LTAG parser and the CKY parser with CFG which is obtained from automatically extracted LTAG grammars. The comparison showed that the obtained CFG is useful for CFG filtering for a LTAG parser. We will implement CFG filtering for a LTAG parser, and verify the efficiency of CFG filtering with our approximated CFG.

References

John Carroll, Nicolas Nicolov, Olga Shaumyan, Martine Smets, and David Weir. 1998. The LEXSYS project. In *Proc. of the fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 29–33.

K. Harbusch. 1990. An efficient parsing algorithm for Tree Adjoining Grammars. In *Proc. of the 28th ACL*, pages 284–291.

M. Kay, J. Gawron, and P. Norvig. 1994. *Verbmobil: A Translation System for Face-to-Face Dialog*. CSLI Publications.

B. Kiefer and H.-U. Krieger. 2000. A context-free approximation of Head-Driven Phrase Structure Grammar. In *Proc. of the sixth IWPT*, pages 135–146.

J. T. Maxwell III and R. M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.

Deep Thought Project. 2003. http://www.project-deepthought.net/.

Kototoi Project. 2001. http://www-tsujii.is.s.u-tokyo.ac.jp/kototoi/.

XTAG Research Group. 2001. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Y. Miyao, T. Ninomiya, and J. Tsujii. 2003. Lexicalized grammar acquisition. In *Proc. of the 10th EACL companion volume*, pages 127–130.

M.-J. Nederhof. 1998. An alternative LR algorithm for TAGs. In *Proc. of COLING–ACL*, pages 946–952.

C. Pollard and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. CSLI Publications.

P. Poller and T. Becker. 1998. Two-step TAG parsing revisited. In *Proc. of TAG+4*, pages 143–146.

A. Sarkar. 2000. Practical experiments in parsing using Tree Adjoining Grammars. In *Proc. of TAG+5*, pages 193–198.

Table 3: LTAG grammars extracted from Penn Treebank Wall Street Journal

corpus	02-06	02-11	02-16	02-21
# of words	2285	3662	5074	6056
# of non-terminals	190	207	211	216
# of elementary tree	1061	1484	1854	2111

Table 4: Obtained CFG from LTAG grammars

corpus	02-06	02-11	02-16	02-21
# of terminal labels	1061	1484	1854	2111
# of non-terminal labels	1768	2780	3458	3911
# of rules	45173	92908	129542	154407

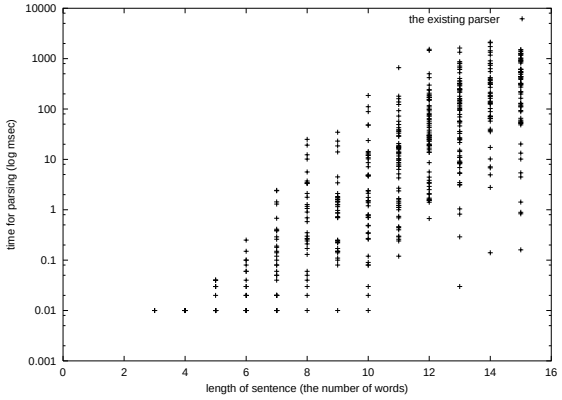


Figure 10: Speed test for an existing LTAG parser

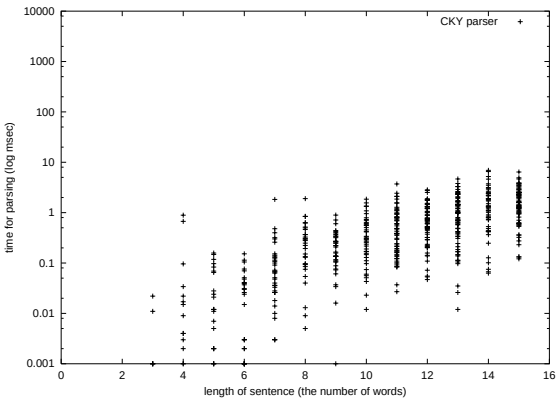


Figure 11: Speed test for a CKY parser with the obtained CFG

Yves Schabes and Aravind Joshi. 1988. An earley-type parsing algorithm for Tree Adjoining Grammars. In *Proc. of the 26th Annual Meeting of the Association for Computational Linguistics (ACL 1988)*, pages 258–269.

Y. Schabes, A. Abeillé, and A. K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In *Proc. of the 12th COLING*, pages 578–583.

K. Torisawa and J. Tsujii. 1996. Computing phrasal-signs in HPSG prior to parsing. In *Proc. of the 16th COLING*, pages 949–955.

K. Torisawa, K. Nishida, Y. Miyao, and J. Tsujii. 2000. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6(1):63–80.

G. van Noord. 1994. Head corner parsing for TAG. *Computational Intelligence*, 10(4):525–534.

K. Vijay-Shanker and A. K. Joshi. 1985. Some computational properties of Tree Adjoining Grammars. In *Proc. of the 23rd ACL*, pages 82–93.

N. Yoshinaga and Y. Miyao. 2002. Grammar conversion from LTAG to HPSG. *The European Student Journal on Language and Speech*. available at http://hltheses.elsnet.org/eventpapers/essliproceed.htm.

N. Yoshinaga, T. Kentaro, and J. Tsujii. 2003. Comparison between CFG filtering techniques for LTAG and HPSG. In *Proc. of the 41st ACL companion volume*, pages 185–188.

On Scrambling, Another Perspective

James Rogers
Dept. of Computer Science
Earlham College
Richmond, IN USA
jrogers@cs.earlham.edu

Abstract

We explore the possibility of accounting for scrambling patterns in German using multi-dimensional grammars. The primary desirable characteristic of this approach is that it employs elementary structures with a single uniform component and combining operations which operate at a single point in the derived structure. As a result, we obtain an analysis that is much closer in spirit to ordinary TAG and to the intuitions of TAG based linguistics. Ultimately, we obtain an account in which the variations in word order are consequences of variations of a small set of structural parameters throughout their ranges.

1 Introduction

The difficulty of accounting for the phenomenon of scrambling, the apparently arbitrary order in which arguments can occur in subordinate clauses in German, has been one of the primary motivations for exploration of extensions to TAG (Rambow, 1994; Kulick, 2000; Rambow et al., 1995; Rambow et al., 2001; Becker et al., 1991). The issue is not generation of the string sets—in most accounts these are actually context-free—but rather doing so within a derivational framework in which lexical heads and their arguments are introduced simultaneously. This idea that elementary structures should include all and only a single thematic domain (Frank, 2002) is generally taken to be the foundation of TAG based linguistic theories. Among other things, it insures that every elementary structure is semantically coherent and that derivations maintain that coherence. Under these assumptions, it has been shown that scrambling is beyond the generative power of ordinary TAG and, in full generality, beyond even set-local multicomponent TAG (Becker et al., 1992).

Generally, extensions to TAG intended to accommodate scrambling involve factorization of elementary structures either into tree sets or into trees with more or less independent regions accompanied by a modification of the combining operation to interleave these regions in the derived tree. In this paper, following the lead of our exploration of similar issues in TAG accounts of English raising phenomena (Rogers, 2002), we explore one illustrative pattern of scrambling using *multi-dimensional grammars* (Rogers, 2003). The primary desirable characteristic of this approach is that it employs elementary structures with a single uniform component and combining operations which operate at a single point in the derived structure. As a result, we obtain an analysis that is much closer in spirit to ordinary TAG and to the intuitions of TAG based linguistics. Ultimately, we obtain an account in which the variations in word order are consequences of variations of a small set of structural parameters throughout their ranges.

We should be clear at the outset that even though our primary motivation is a desire to preserve the fundamental tenets of standard TAG theories of syntax, our goal is not a linguistically complete account of scrambling, or even one that is linguistically motivated beyond the goal of maintaining semantically coherent elementary structures and derivations. Rather, we intend to show how the formal power of the multi-dimensional grammars can, potentially, support such an account. We look only at one particular case of scrambling, but we believe that this case illustrates the relevant formal issues. These results suggest that scrambling phenomena of any concrete degree of complexity can be handled at some level of the multi-dimensional grammar hierarchy. We close the paper with some speculation about what such a result might have to say about the nature of limits on the acceptability of scrambling as its complexity increases.

2 A Formalized Instance of Scrambling

The case we examine, taken directly from Joshi, Becker and Rambow (Joshi et al., 2000) (also (Becker et al., 1991)), involves scrambling within a matrix clause headed by a verb that subcategorizes for two NPs and an

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 178-185.

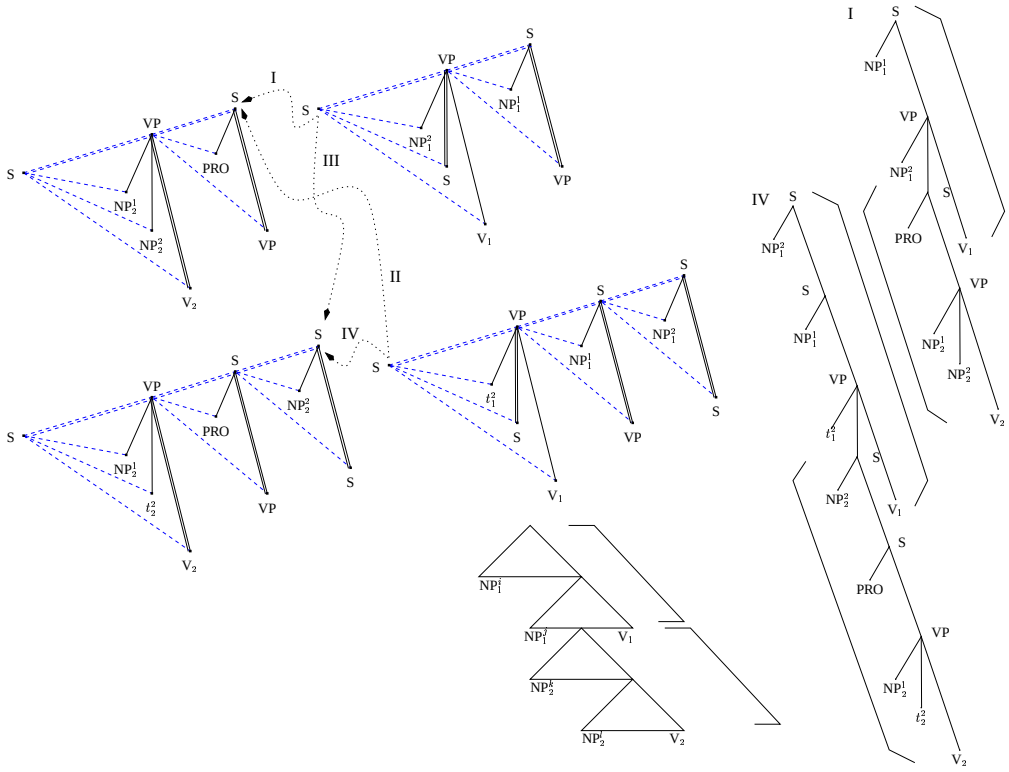


Figure 1: Class A) $\mathrm{NP}_1^i \mathrm{NP}_1^j \mathrm{NP}_2^k \mathrm{NP}_2^l \mathrm{V}_2 \mathrm{V}_1$

S with that embedded S headed by a verb that subcategorizes for three NPs, one of which is a PRO subject. This formalizes as

$$\{\sigma(\mathrm{NP}_1^1, \mathrm{NP}_1^2, \mathrm{NP}_2^1, \mathrm{NP}_2^2)\mathrm{V}_2\mathrm{V}_1 \mid \sigma \text{ a permutation}\}$$

where NP_1^1 and NP_1^2 are the first and second argument of the matrix verb and NP_2^1 and NP_2^2 are the arguments, other than PRO, of the embedded clause.

There are 24 permutations of the four NP arguments. We divide these into six classes (where i, j and k, l each vary over 1,2):

A) $\mathrm{NP}_1^i \mathrm{NP}_1^j \mathrm{NP}_2^k \mathrm{NP}_2^l \mathrm{V}_2 \mathrm{V}_1$
B) $\mathrm{NP}_2^k \mathrm{NP}_1^i \mathrm{NP}_1^j \mathrm{NP}_2^l \mathrm{V}_2 \mathrm{V}_1$
C) $\mathrm{NP}_2^k \mathrm{NP}_2^l \mathrm{NP}_1^i \mathrm{NP}_1^j \mathrm{V}_2 \mathrm{V}_1$
D) $\mathrm{NP}_1^i \mathrm{NP}_2^k \mathrm{NP}_2^l \mathrm{NP}_1^j \mathrm{V}_2 \mathrm{V}_1$
E) $\mathrm{NP}_2^k \mathrm{NP}_1^i \mathrm{NP}_2^l \mathrm{NP}_1^j \mathrm{V}_2 \mathrm{V}_1$
F) $\mathrm{NP}_1^i \mathrm{NP}_2^k \mathrm{NP}_1^j \mathrm{NP}_2^l \mathrm{V}_2 \mathrm{V}_1$.

3 Class A)—CF Structures

Class A) is the canonical structure with, potentially, the arguments extracted within their own clauses. Standard TAG accounts treat the matrix clause as an auxiliary tree adjoining at the root of the embedded clause. Following Rogers (1998) and Rogers (1999) we interpret TAG as a sort of Context-Free Grammar over trees. Just as CFG productions can be interpreted as *local* (height one) trees expanding a root node to a string yield with the derivation process splicing these together to form derivation trees, TAG auxiliary trees can be interpreted as local three-dimensional structures expanding a root node to a tree yield with the TAG derivation process splicing these together to form three-dimensional derivation structures.

These derivation structures correspond exactly to the derivation trees normally associated with TAG, with the exception that the derived structure (in this case a tree) can be obtained from it by restricting to nodes of maximal depth (in the third dimension) in a way analogous to taking the string yield of a CFG derivation tree. The intuition behind these structures is that TAG expresses a hierarchical decomposition of trees analogous to the hierarchical decomposition of strings that those trees represent.

It is important not to misconstrue this notion of "dimension." While it may be convenient to visualize these structures as having actual extent in the third dimension, they are, fundamentally, just graphs with multi-sorted edges and, hence, dimensionless. The three dimensions correspond to linear precedence, ordinary domination and domination of the "adjoining" sort. These are not arbitrary or independent. As they represent recursive hierarchical decomposition, each edge relation is "tree-

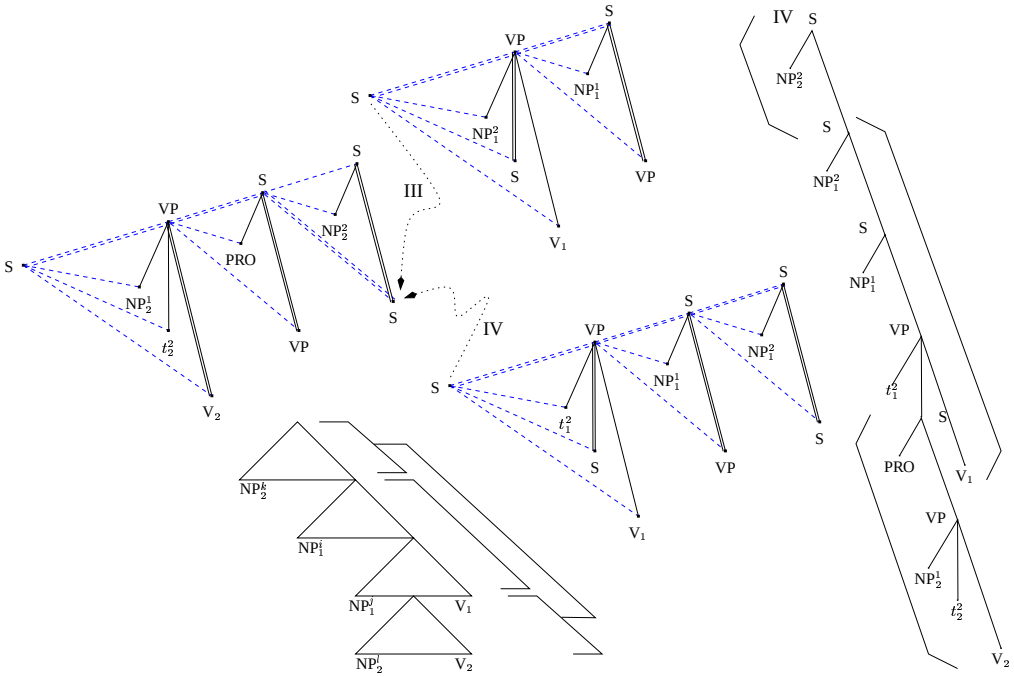


Figure 2: Class B) $\mathrm{NP}_2^k \mathrm{NP}_1^i \mathrm{NP}_1^j \mathrm{NP}_2^l \mathrm{V}_2 \mathrm{V}_1$

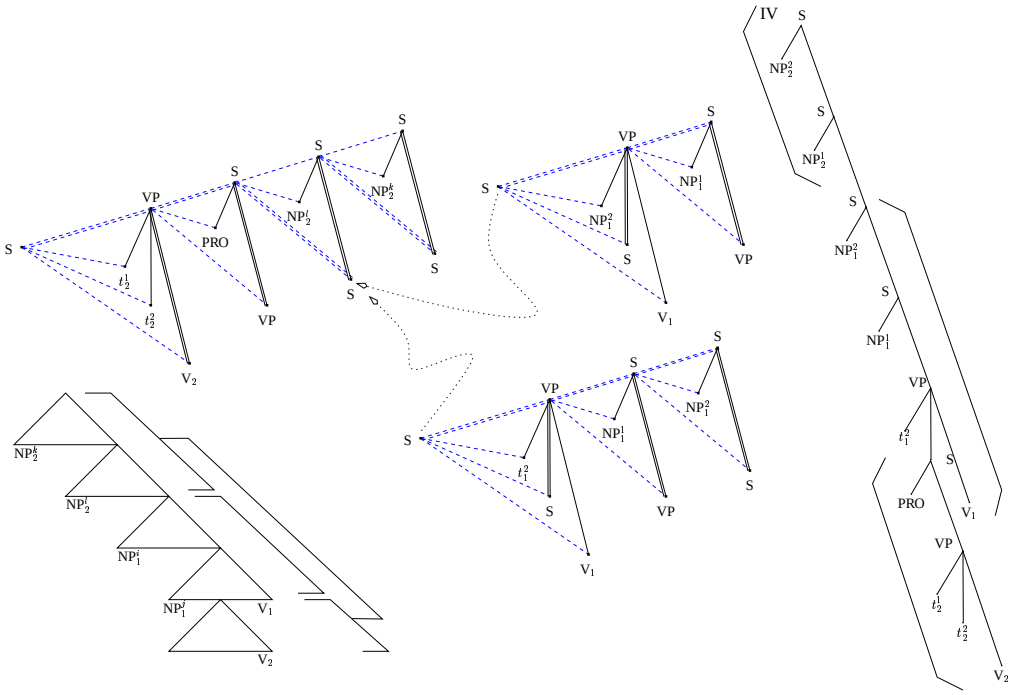


Figure 3: Class C) $\mathrm{NP}_2^k \mathrm{NP}_2^l \mathrm{NP}_1^i \mathrm{NP}_1^j \mathrm{V}_2 \mathrm{V}_1$

like" and descendants of a node along a given dimension inherit the relationships of that node in all smaller dimensions in the same way that linear precedence is inherited by the nodes in a subtree. (See Rogers (2003).)

Here, the adjunction of the matrix tree at the root of the embedded tree attaches the root of the three dimensional structure representing the former at the S node in the (tree) yield of the latter. (See Figure 1.) The upper pair of structures represent the base structures, the lower pair the structures with a locally extracted argument. Each of the four variations of Class A) is obtained from one of the four combinations of these four structures. We have followed Rogers (2002) in treating the subject as if it were adjoined, in some sense, at the root of the VP. This is done, here, for purely formal reasons—it will provide the structural flexibility we will need to derive the more complex scrambling patterns. We carry this structural configuration through in treating extraction; we assume that extracted arguments attach in a similar fashion to the root of the S.

Note that there is potential ambiguity in taking the tree yield of these structures in that the yield of one component might attach at any of the leaves of the yield of the component which immediately dominates it. In TAG, of course, this is resolved by designating one of the leaves as the *foot* with all splicing being done at the foot. Here we designate the foot by marking the *spine* of the components with doubled lines. We carry this through in higher dimensions, as well; each elementary structure, in each dimension, has a spine leading from its root to a foot node in its yield. Two of the four resulting tree yields are shown on right of the figure.

Since adjunction at the root has the same effect as substitution, this is effectively a context-free structure. As shown schematically in the figure, the (two dimensional) yields of the two structures are simply concatenated. Note that, as in the standard TAG accounts, additional recursion is accommodated by adjoining additional subordinating structures at the root of what is here the matrix structure.

4 Classes B) and C)—Ordinary Adjunction

In Classes B) and C), the arguments of V_2 are wrapped around those of V_1, as shown at the bottom of Figures 2 and 3. This is the pattern corresponding to adjunction proper. Class B) can be obtained by extracting either NP_2^1, NP_2^2 then adjoining the matrix structure at the foot of the yield of the extracted NP structure (the point between the extracted element and the original S node). As usual, this has the effect of splitting the tree yields of the subordinate structure into two factors and inserting the tree yield of the matrix structure between them. Note that all that distinguishes this class from Class A) is the third-dimensional foot node of the embedded structure, which is, itself, determined by the form of the extracted NP structure.

Class C) is nearly identical. We extract both of the arguments of V_2 and adjoin, again, at the point between the extracted elements and the original S node.

Note that in both these cases, the scrambling can apply recursively by attaching additional auxiliary structures at the tree yield of the first. If this is attached at the node corresponding to the root of the yield of that structure, in the manner of Class A), the effect is only to move the arguments scrambled out of the more deeply embedded clauses across the new clause. If, on the other hand, it is attached at the foot of the yield of the extracted NP structure, in the manner of Classes B) and C), then the effect will be to scramble additional arguments out of the intermediate clause.

5 Class D)—Higher-Order Adjunction

Class D) is the first of the configurations that cannot be obtained by ordinary adjunction. Here the arguments of V_1 and V_2 don't simply nest one inside the other, but, rather interleave in the way shown schematically on the top left of Figure 4. Since the sequences of labels along the spines of TAG tree sets must form CF languages, such "cross-serial" configurations cannot be generated by TAGs. They can, however, be generated if we add another level of hierarchical decomposition. Grammars at this level yield tree sets with TAL spine languages (corresponding to the third level of Weir's Control Language Hierarchy (Weir, 1992)). A schematic representation of the general embedding pattern provided at this level is given in Figure 5.

To employ this nesting pattern we adopt four-dimensional structures and take the matrix structure to, again, attach between the extracted structure and the original S, but now along the fourth dimension. (Figure 4.) This has the result of splitting the three-dimensional yield of the embedded structure into two factors and inserting the three-dimensional yield of the matrix structure between them. Given the third-dimensional foot of the matrix structure, the "upper" factor of the embedded structure is, effectively, adjoined between the two arguments of the matrix structure which is, in turn adjoined at the root of the "lower" factor. (As shown at the bottom of the figure.) The effect on the tree yield is exactly as if the (two-dimensional) matrix tree had been factored into two components, one adjoining at the root of the embedded tree and one properly along its spine. (As shown at the right.)

It should be noted that with this configuration we can account for all the variations of Classes A) through D) by varying the position of the foot of the matrix structure and

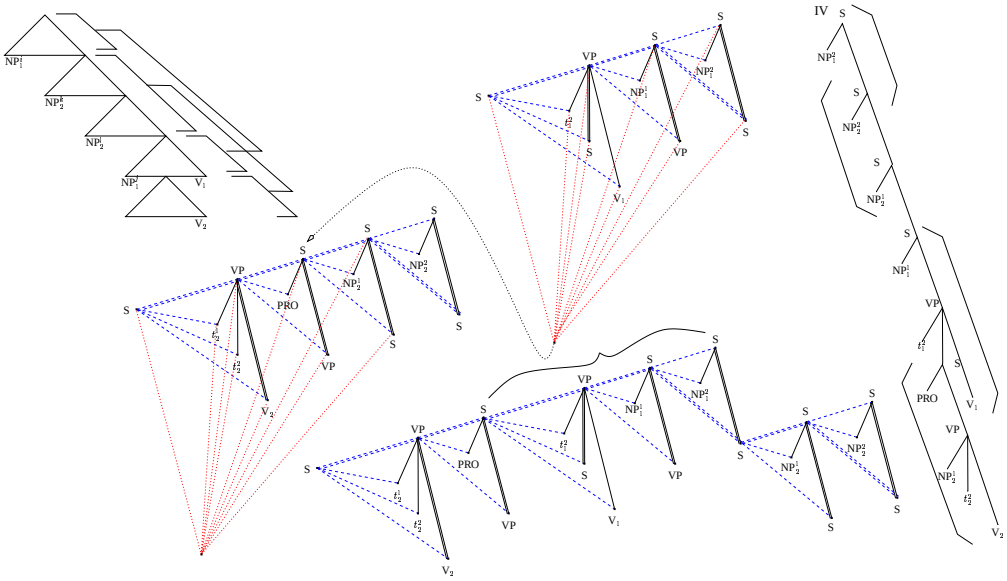


Figure 4: Class D) $\mathrm{NP}_1^i \mathrm{NP}_2^k \mathrm{NP}_2^l \mathrm{NP}_1^j \mathrm{V}_2 \mathrm{V}_1$

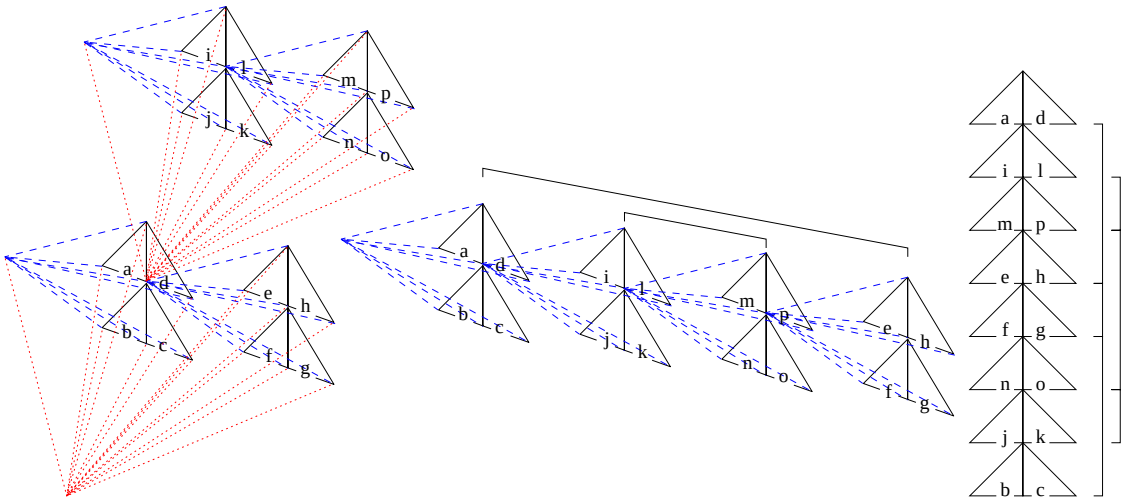
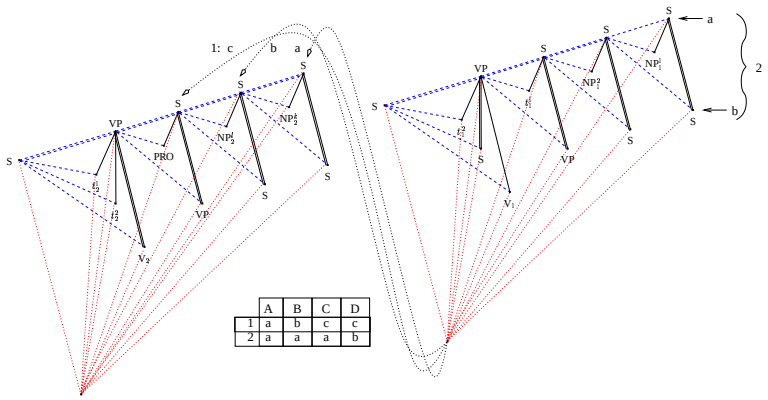


Figure 5: 4^{th}-level nesting



	A	B	C	D
1	a	b	c	c
2	a	a	a	b

Figure 6: A unified account of Classes A) through D)

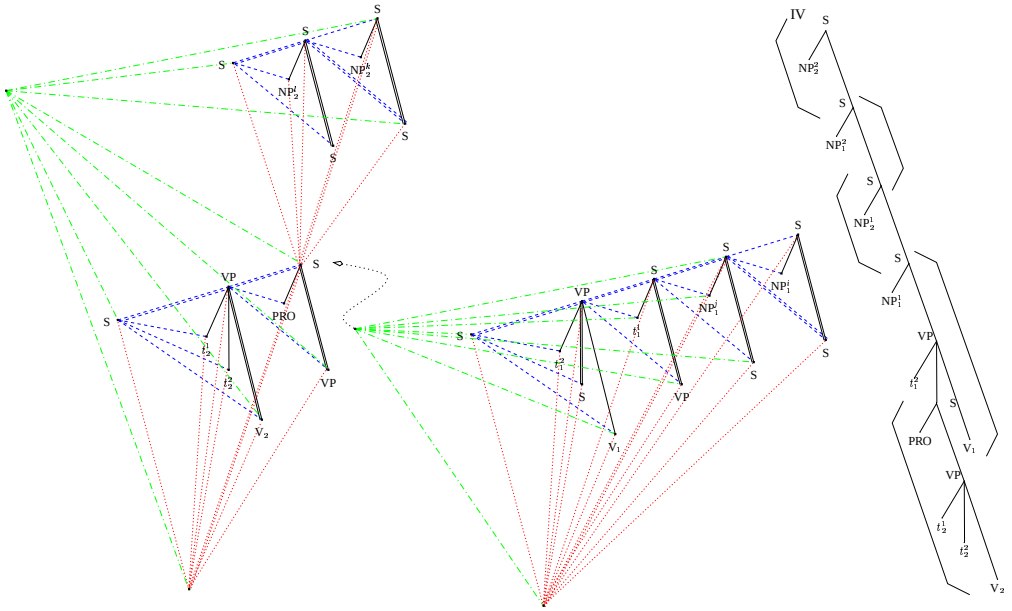


Figure 7: Class E) $\mathrm{NP}_2^k \mathrm{NP}_1^i \mathrm{NP}_2^l \mathrm{NP}_1^j \mathrm{V}_2 \mathrm{V}_1$

the point at which it attaches to the embedded structure. (See Figure 6.)

6 Classes E) and F) and a Unified Account

The nesting pattern of Class E) (Figure 7) requires the embedded tree to be factored into three components, not just two. This can still be obtained with multi-component adjoining in a manner similar to that of Class D), with both components of the matrix structure adjoining properly along the spine of the embedded tree. While this pattern is also obtainable in the four-dimensional grammars, it necessarily uses the upper half of the VP component, which, for the sake of consistency, we would rather not do. Consequently, we again add another hierarchical relationship and lift to the fifth level, taking the two arguments of V_2 to be extracted via the fourth relation rather than the third. Class F) can be treated similarly, with the exception that one of the arguments is extracted along the third relation, the other along the fourth. (Figure 8.)

This variation between Classes E) and F) leads to an account in which all six classes are derivable within a single basic structure, shown in Figure 9. Here there are six parameters of variation:

1. The position of the fourth-dimensional foot of the matrix structure.

2. Whether one or both arguments of V_1 are extracted along the fourth relation.

3. and 4. The position of the three-dimensional feet of the matrix and embedded structures.

5. and 6. And, finally, the relative nesting of the extracted NPs in each structure.

This gives 96 combinations but as the word-order variations are exhaustive, they generate only the 24 distinct configurations of the six classes of structures.

7 Arbitrarily Complex Scrambling

While no level of the multi-dimensional grammar hierarchy can capture scrambling of arbitrary complexity, there is no bound on the number of tree factors that can be interleaved at some level of this hierarchy. In general, grammars at the k^{th} level factor the tree yields of the elementary structures into 2^{k-2} fragments, with the tree yield of the result of adjoining one into another being split into $2^{k-3}+1$ regions from the initial structure interleaved with 2^{k-3} regions from the auxiliary structure. (Figure 10 gives the pattern for the fifth level.) Consequently, scrambling of any concrete degree of complexity can be captured at some level of the hierarchy, although it is not clear that this can necessarily be done in an plausibly "uncontrived" way.

In Joshi et al. (2000), Joshi, Becker and Rambow note that the boundary of general acceptability in scrambling, roughly two levels of embedding, coincides with what can be handled by tree-local MCTAG. This leads them to suggest that this boundary may actually be competence based, rather than performance based as is usually assumed. Here we have additional flexibility. In choosing the level of the competence grammar in the multi-dimensional hierarchy, we set the boundary on the complexity of the scrambling we admit. On the other hand,

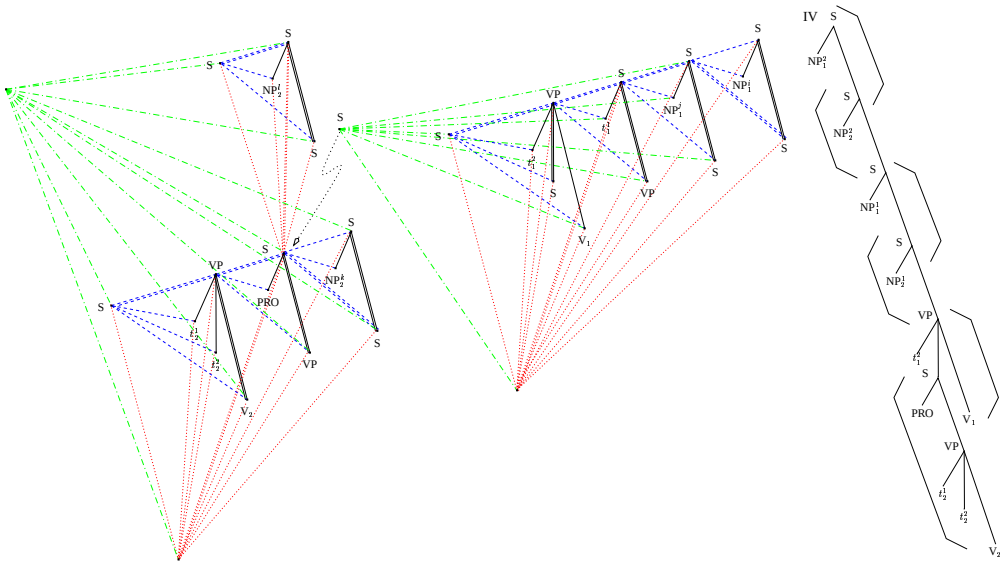


Figure 8: Class F) $\mathrm{NP}_1^i \mathrm{NP}_2^k \mathrm{NP}_1^j \mathrm{NP}_2^l \mathrm{V}_2 \mathrm{V}_1$

given that the level of the grammar corresponds to the number of hierarchical relations we use in encoding the structure of the utterances, one could make a plausible argument that the level of the grammar might be determined by performance considerations, such as working memory limitations. In this way one might arrive at an account of the limits on the complexity of scrambling that was simultaneously performance based—a consequence of bounds on working memory—and competence based—a consequence of the complexity of the grammars which can be processed within those bounds.

References

Tilman Becker, Aravind K. Joshi, and Owen Rambow. 1991. Long-distance scrambling and tree adjoining grammars. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26. ACL.

Tilman Becker, Owen Rambow, and Michael Niv. 1992. The derivational generative power of formal systems or scrambling is beyond LCFRS. Technical report, Institute for Research in Cognitive Science, Univ. of Pennsylvania, Philadelphia, PA.

Robert Frank. 2002. *Phrase Structure Composition and Syntactic Dependencies*. MIT Press.

Aravind K. Joshi, Tilman Becker, and Owen Rambow. 2000. Complexity of scrambling: A new twist to the competence-performance distinction. In Anne Abeillé and Owen Rambow, editors, *Tree Adjoining Grammars*, chapter 6, pages 167–181. CSLI.

Seth Kulick. 2000. *Constraining Non-Local Dependencies in Tree Adjoining Grammar: Computational and Linguistic Perspectives*. Ph.D. thesis, Univ. of Pennsylvania.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-tree grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, pages 151–158, Cambridge, MA.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-tree substitution grammars. *Computational Linguistics*, 27(1):87–121.

Owen Rambow. 1994. *Formal and Computational Aspects of Natural Language Syntax*. Ph.D. thesis, Univ. of Pennsylvania.

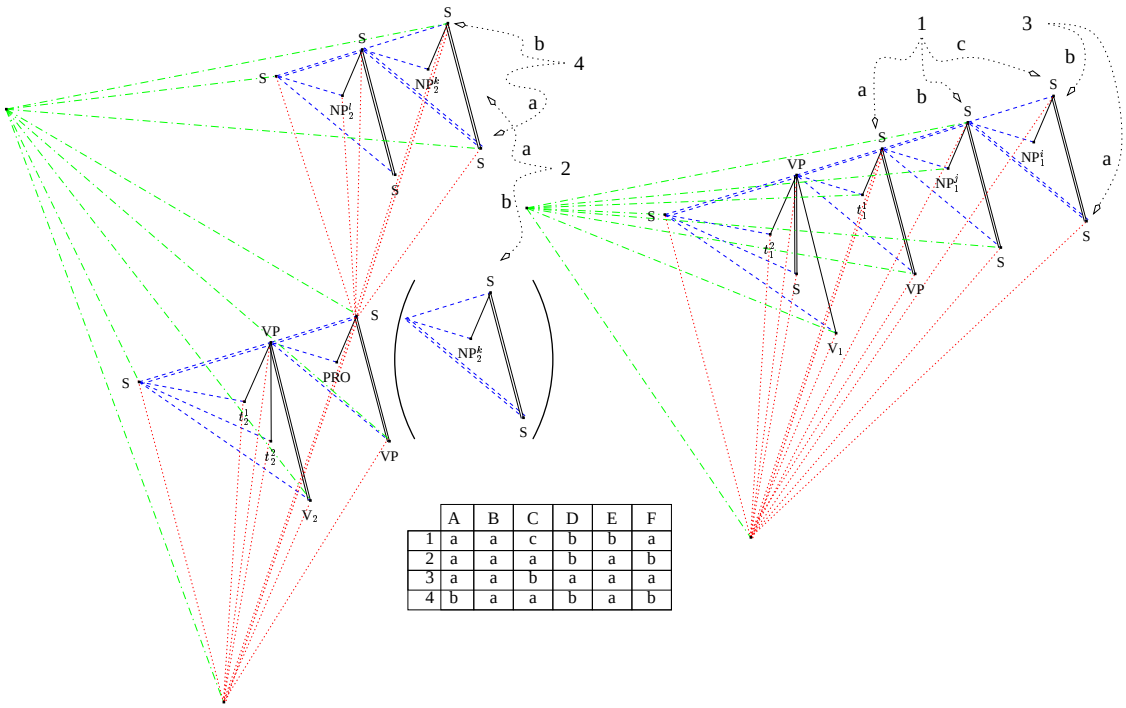
James Rogers. 1998. A descriptive characterization of tree-adjoining languages. In *Proc. of the 17th International Conference on Computational Linguistics (COLING'98) and the 36th Annual Meeting of the Association for Computational Linguistics (ACL'98)*, Montreal. ACL. Project Note.

James Rogers. 1999. Generalized tree-adjoining grammar. In *Sixth Meeting on Mathematics of Language*, pages 189–202.

James Rogers. 2002. One more perspective on semantic relations in tag. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Venice, IT, May.

James Rogers. 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science*, 293(2):291–320.

David J. Weir. 1992. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104:235–261.



	A	B	C	D	E	F
1	a	a	c	b	b	a
2	a	a	a	b	a	b
3	a	a	b	a	a	a
4	b	a	a	b	a	b

Figure 9: Unified account

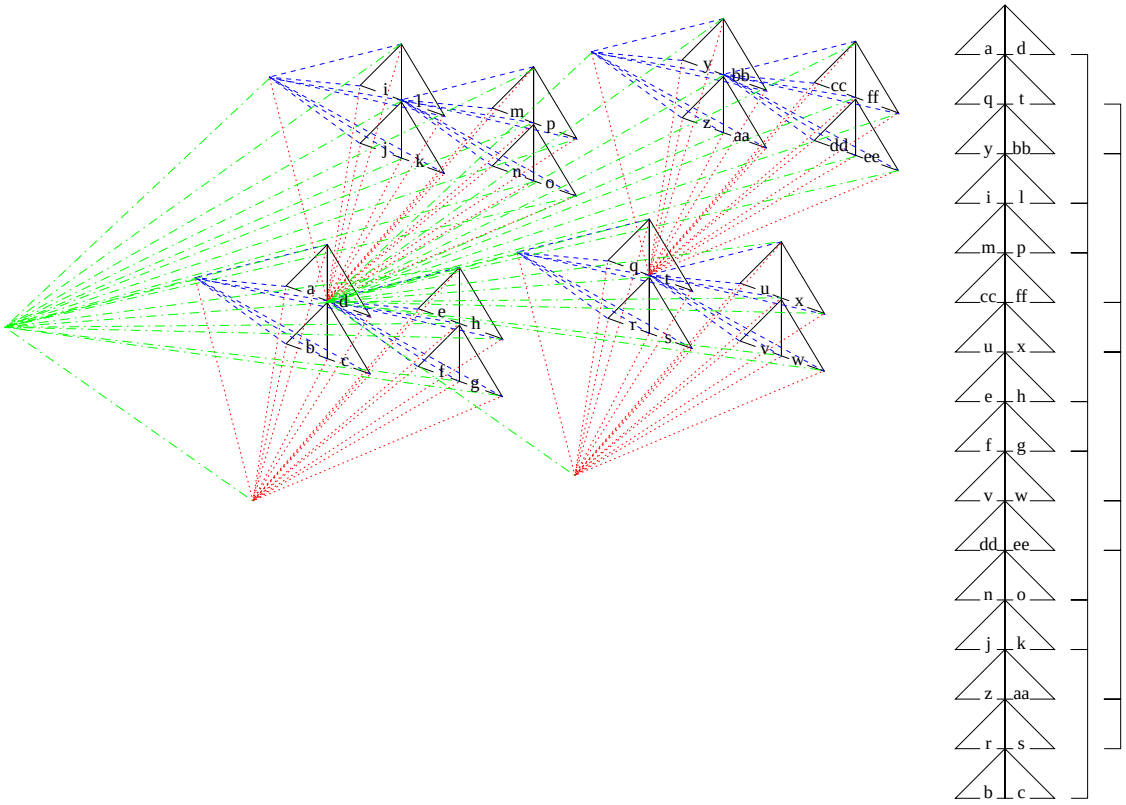


Figure 10: 5^{th}-level nesting

LTAG Semantics for Questions

Maribel Romero
Department of Linguistics
610, Williams Hall
University of Pennsylvania
Philadelphia, PA, 19104-6305
U.S.A.
romero@ling.upenn.edu

Laura Kallmeyer
TALaNa-Lattice
UFRL, University Paris 7
2 place Jussieu, Case 7003,
75251 Paris Cedex 05
France
lkallmey@linguist.jussieu.fr

Olga Babko-Malaya
CIS Department
3330 Walnut St.
University of Pennsylvania
Philadelphia, PA 19104-6389
U.S.A.
malayao@linc.cis.upenn.edu

Abstract

This papers presents a compositional semantic analysis of interrogatives clauses in LTAG (Lexicalized Tree Adjoining Grammar) that captures the scopal properties of wh- and non-wh-quantificational elements. It is shown that the present approach derives the correct semantics for examples claimed to be problematic for LTAG semantic approaches based on the derivation tree. The paper further provides an LTAG semantics for embedded interrogatives.

1 Introduction.

Following (Karttunen, 1977), an interrogative clause Q expresses a function from possible situations (or worlds) to the set of true answers to that question Q in that situation. For example, the interrogative clause (1) has the meaning (2), where *who* contributes the $\exists$-quantification $\exists x[\mathsf{person}(x, s_0)]$. In a situation s_0 where Pat, Al, Kate and nobody else called, $[\![\mathtt{Q}(s_0)]\!]$ equals the set (3).

(1) who called?

(2) $\lambda s_0 \lambda p.p(s_0) \wedge$
$\exists x[\mathsf{person}(x, s_0) \wedge p = \lambda s.\mathsf{call}(x, s)]$

(3) $\{\ \lambda s.\mathsf{call}(\mathsf{pat}, s),\ \lambda s.\mathsf{call}(\mathsf{al}, s),\ \lambda s.\mathsf{call}(\mathsf{kate}, s)\ \}$

The aim of this paper is to develop a compositional semantic analysis of interrogative clauses in LTAG, with two goals: **(i)** the main goal is to capture the scopal properties of quantificational elements within the question, and **(ii)** the secondary goal is to achieve the correct semantics for interrogatives embedded under e.g. *know*.

The scope data concerning goal **(i)** are the following. When an interrogative clause contains a *wh*-element and a non-*wh* quantificational element, as in (4), the semantic contribution of *who* must be outside the proposition headed by λs, whereas the semantic contribution of *everybody* must be inside that proposition, as shown in (5).[1]

(4) (John knows) who likes everybody

(5) (John knows) $\lambda s_0 \lambda p.p(s_0)$
$\wedge \exists x[\mathsf{person}(x, s_0) \wedge p = \lambda s.\forall y[\mathsf{person}(y, s)$
$\rightarrow \mathsf{like}(x, y, s)]]$

Note that, when we have more than one wh-phrase and more than one non-wh-quantifier, the non-wh-quantifiers can yield difference scope configurations among themselves (and so can the wh-phrases among themselves, trivially). But all the wh-phrases must take scope above the λs proposition and all the non-wh-quantifiers must take scope below it. This is illustrated in (6), which has the readings (7)-(8), but not e.g. the readings (9)-(10).

(6) (John knows) who seemed to introduce who to everybody

(7) (John knows) $\lambda s_0 \lambda p.p(s_0)$
$\wedge \exists x \exists y[\mathsf{person}(x, s_0) \wedge \mathsf{person}(y, s_0)$
$\wedge\ p = \lambda s.\mathsf{seem}(\lambda s'.\forall z[\mathsf{person}(z, s')$
$\rightarrow \mathsf{introduce}(x, y, z, s')], s)]$

(8) (John knows) $\lambda s_0 \lambda p.p(s_0)$
$\wedge \exists x \exists y[\mathsf{person}(x, s_0) \wedge \mathsf{person}(y, s_0)$
$\wedge\ `p = \lambda s.\forall z[\mathsf{person}(z, s)$
$\rightarrow \mathsf{seem}(\lambda s'.\mathsf{introduce}(x, y, z, s')], s)]$

(9) (John knows) $\lambda s_0 \lambda p.p(s_0)$
$\wedge \exists x[\mathsf{person}(x, s_0) \wedge p = \lambda s.\exists y[\mathsf{person}(y, s)$
$\wedge\ \mathsf{seem}(\lambda s'.\forall z[\mathsf{person}(z, s')$
$\rightarrow \mathsf{introduce}(x, y, z, s')], s)]]$

(10) (John knows) $\lambda s_0 \lambda p.p(s_0)$
$\wedge \exists x \exists y[\mathsf{person}(x, s_0) \wedge \mathsf{person}(y, s_0)$
$\wedge \forall z[\mathsf{person}(z, s_0)$
$\rightarrow p = \lambda s.\mathsf{seem}(\lambda s'.\mathsf{introduce}(x, y, z, s')], s)]$

[1] We leave aside the so-called pair-list readings arising when *everybody* c-commands the trace of the *wh*-phrase and a special absorption operation takes place (Chierchia, 1993).

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 186-193.

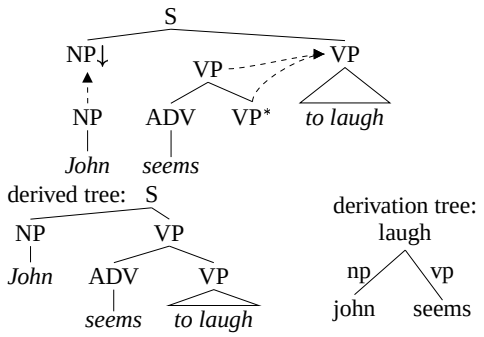


Figure 1: Derivation for (13)

With respect to goal **(ii)**, we need to construct a question meaning that will be able to combine with a question taking verb like *know*. In the end, a sentence like (11) must receive the truth-conditions in (12). The expression $Dox_j(s_0)$ in (12) stands for the set of doxastic alternatives of John in s_0, that is, for the set of possible situations s' that conform to John's beliefs in s_0. The formula (12) states that we are in a situation s_0 such that, for all of John's belief alternatives s' in s_0 and for all propositions p, $p \in [\![\text{who called}]\!](s')$ iff $p \in [\![\text{who called}]\!](s_0)$.

(11) John knows who called.

(12) $\lambda s_0.\forall s'_s \in Dox_j(s_0)\forall p_{<s,t>}$
$[\exists x[\mathsf{person}(x,s') \wedge p(s') \wedge p = \lambda s.\mathsf{call}(x,s)]$
$\leftrightarrow \exists x[\mathsf{person}(x,s_0) \wedge p(s_0) \wedge p = \lambda s.\mathsf{call}(x,s)]]$

2 Semantic unification

For LTAG semantics, we use the semantic unification framework described in (Kallmeyer and Romero, 2004) that is very close to (Gardent and Kallmeyer, 2003): We do compositional semantics on the derivation tree, i.e., each elementary tree has a semantic representation and the derivation tree indicates how to do semantic computation. Semantic representations are equipped with semantic feature structures. Semantic representations are sets of formulas (typed λ-expressions with labels) and scope constraints. A scope constraint is an expression $x \geq y$ where x and y are propositional labels or propositional variables. Semantic feature structures have features P for all node positions p that can occur in elementary trees.[2] The values of these features are feature structure that consist of a T and a B feature (top and bottom) whose values are feature structures with features I for individual variables, P for propositional labels and S for situations.

Semantic composition consists of unification: In the derivation tree, elementary trees are replaced by their semantic representations and their semantic feature structures. Then, for each edge from γ_1 to γ_2 with position p:

[2]For the sake of readability, we use names np, vp, ... for the node positions instead of the usual Gorn adresses.

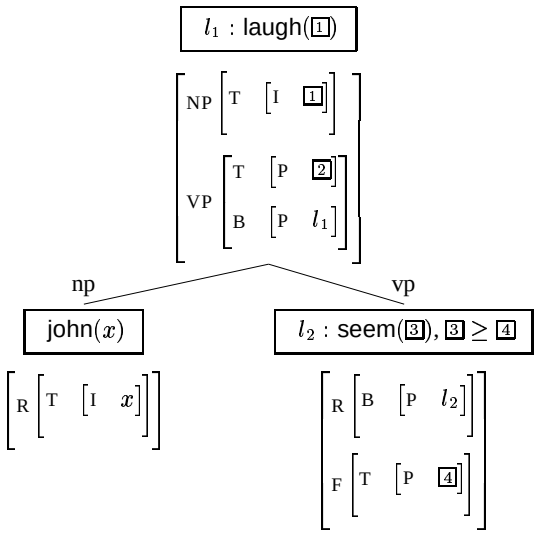


Figure 2: Semantics for (13)

1. the T feature of position p in γ_1 and the T feature of the root of γ_2 are identified, and 2. if γ_2 is an auxiliary tree, then the B feature of the foot node of γ_2 and the B feature of position p in γ_1 are identified. Furthermore, for all γ occurring in the derivation tree and all positions p in γ such that there is no edge from γ to some other tree with position p: the T and B features of $\gamma.p$ are identified. By these unifications, some of the variables in the semantic representations get values. Then, the union of all semantic representations is built which yields an underspecified representation. Finally, appropriate disambiguations must be found, i.e., assignments for the remaining propositional variables that respect the scope constraints in the sense of (Kallmeyer and Joshi, 2003). The disambiguated representations are interpreted conjunctively. As an example, Fig. 1 and 2 show the derivation and the semantics for (13).

(13) John seems to laugh

The feature identities because of unification are $\boxed{1} = x$, $\boxed{2} = l_2$, $\boxed{4} = l_1$which leads to (14). There is only one disambiguation, $\boxed{3} \rightarrow l_1$ which yields the semantics $\mathsf{john}(x) \wedge \mathsf{seem}(\mathsf{laugh}(x))$.[3]

(14) l_1 : laugh(x), john(x), l_2 : seem(3), 3 ≥ l_1

3 Scopal properties of wh-phrases

3.1 Quantificational NPs

Following previous approaches ((Kallmeyer and Joshi, 2003; Joshi et al., 2003) and also (Kallmeyer and Romero, 2004)), we assume that quantifiers as *everybody*

[3]For simplification, in (14) situation variables are omitted.

in (15) have a multicomponent set containing an auxiliary tree that contributes the scope part and an initial tree that contributes the predicate argument part. Fig. 3 illustrates this approach.

(15) everybody laughs

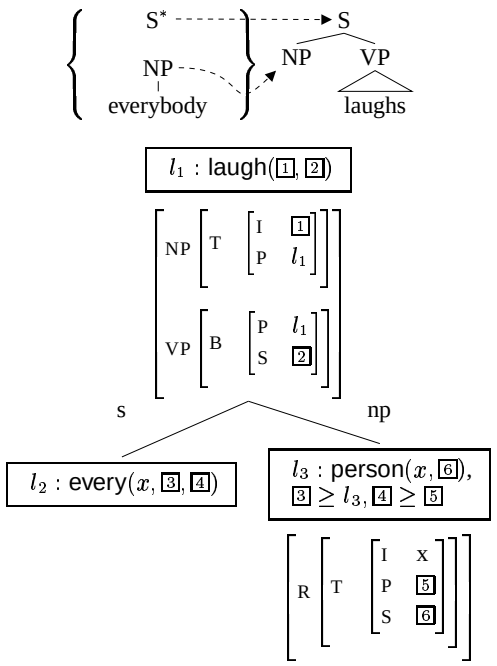


Figure 3: Analysis of (15)

The analysis in Fig. 3 leads to the feature identities $\boxed{1} = x, \boxed{5} = l_1$. As a result one obtains (16). There is one disambiguation, $\boxed{3} \rightarrow l_3, \boxed{4} \rightarrow l_1$, that yields the semantics $\mathsf{every}(x, \mathsf{person}(x, \boxed{6}), \mathsf{laugh}(x, \boxed{2}))$.

(16) l_1 : $\mathsf{laugh}(x, \boxed{2})$, l_2 : $\mathsf{every}(x, \boxed{3}, \boxed{4})$,
l_3 : $\mathsf{person}(x, \boxed{6})$, $\boxed{3} \geq l_3, \boxed{4} \geq l_1$

Following (Percus, 2000), situation variables in verbs must be locally bound, and situation variables in NPs can be non-locally bound by any situation binder in the sentence (e.g. by *know* in (4)). In the current example (15), the situation variable $\boxed{2}$ in the verb *laugh* and the situation variable $\boxed{6}$ in *everybody* will default to s_0 (the situation of the whole proposition), since there is no situation binder in the formula. This yields the final semantics $\mathsf{every}(x, \mathsf{person}(x, s_0), \mathsf{laugh}(x, s_0))$.

3.2 Wh-phrases as quantifiers

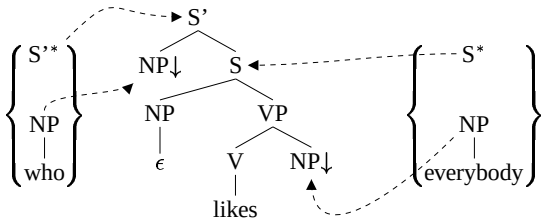
Consider again example (4) *who likes everybody?* and its Karttunen-style semantics in (5), repeated as (17) below. To achieve this result in LTAG, we propose the derivation and the semantics in Fig. 4. The crucial ingredients are as follows.

(17) (John knows) $\lambda s_0 \lambda p.p(s_0)$
$\wedge \exists x[\mathsf{person}(x, s_0) \wedge p = \lambda s.\forall y[\mathsf{person}(y, s/s_0)$
$\rightarrow \mathsf{like}(x, y, s)]]$

The semantic representation for the interrogative elementary tree of *like* must include all the semantic information in (5) except for $\exists x[\mathsf{person}(x, s_0)]$ –coming from *who*– and $\forall y[\mathsf{person}(y, s/s_0)]$ –coming from *everybody*. Since the wh- and non-wh-quantificational elements must have scope over different portions of the formula, the semantic representation of the interrogative tree for *like* is split into several separate subformulae, each with its own label and with constraints guaranteeing the correct scopal configuration among them. First, it contains the formula l_1 : $\mathsf{like}(\boxed{1}, \boxed{2}, \boxed{3})$, shared by all the family trees for *like*. Second, it contributes the formula $l_2 : p = \lambda s.\boxed{7}$, which will take scope over l_1, given that $\boxed{7} = \boxed{4}$ and that $\boxed{6} = l_1$ (by identification of T and B features in positions S and VP respectively) and given the scope constraint $\boxed{4} \geq \boxed{6}$. Finally, the interrogative tree for *like* contributes the expression $q_3 : \lambda p.\boxed{5}$, with scope over l_2 due to the scope constraint $\boxed{5} \geq l_2$. (Note that q_3 is not a propositional formula and hence cannot be interpreted as conjoined with the rest. See section 4 and footnote 6 on this issue.)

What we need to achieve with respect to scope is that all quantificational NPs take scope under $\boxed{7}$ and over l_1, and that all wh-phrases take scope under $\boxed{5}$ and over l_2 We propose a multi-component analysis of *wh*-phrases parallel to that of quantificational NPs, with the only difference that the scope part of a wh-quantifier adjoins to S' whereas the scope part of a non-wh-quantifier adjoins to S, as shown in Fig. 3. This parallel treatment is appropriate since the scope of wh-quantifiers is not strictly related to their surface positions, e.g., in situ wh phrases can take wide scope. We then define a "scope window" for wh- and non-wh-quantificational NPs by using two semantic features linked to the two parts of the multi-component: MAXS is linked to the S* or S'* part and gives the upper limit of the scope window, and P is linked to the NP-part and determines the lower limit of the scope window. In the case of *everybody* in Fig. 4, the value of MAXS is $\boxed{13}$, then $\boxed{13} = \boxed{4}$ (by adjunction to S in *like* tree), and finally $\boxed{4} = \boxed{7}$ (by T/B unification in S of *likes*). The value of *everybody*'s lower limit P is $\boxed{16}$, and $\boxed{16} = l_1$ (by substitution into position NP in *like* tree). This gives us the desired result $\boxed{7} \geq l_6 \geq l_1$, where l_6 introduces the $\forall$-quantification corresponding to *everybody*.[4] The case of *who* is parallel. Its MAXS feature, in the S'* part, has the value $\boxed{8}$, and $\boxed{8} = \boxed{5}$ (by adjunction to S'). Its lower limit feature P, in the NP part, has the value $\boxed{11}$, and $\boxed{11} = l_2$ (by substituion into position WH of *like* tree). This yields the desired scope $\boxed{5} \geq l_4 \geq l_2$, where l_4 corresponds to

[4] See also (Kallmeyer and Romero, 2004) for further motivation of the MAXS feature for quantifiers.



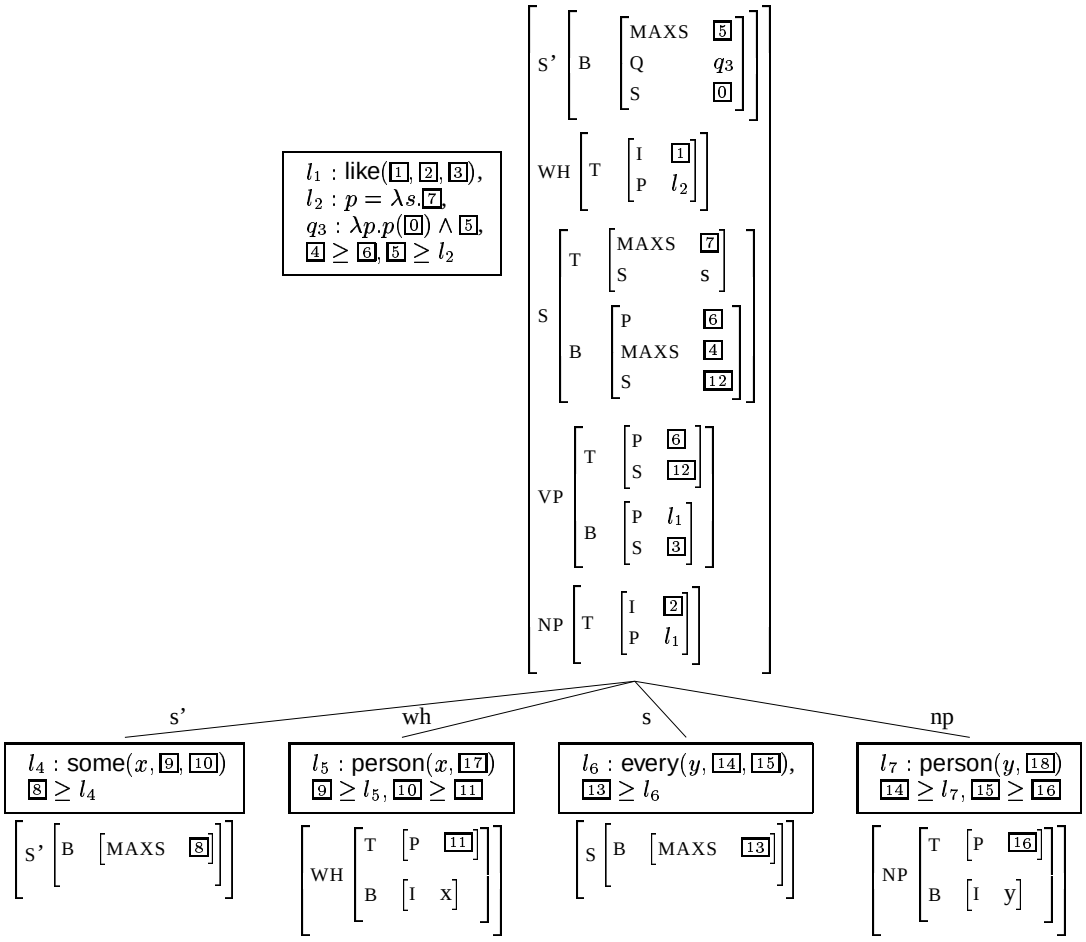


Figure 4: Derivation and derivation tree with semantics for (4) *who likes everybody*

the ∃-quantification of *who*. Hence, by defining an upper limit feature MAXS and a lower limit feature P for wh- and non-wh-quantifiers, we can obtain the right scopal configurations.

The semantic representation one obtains for (4) is (18):

(18)
$$\begin{array}{l} l_1 : \mathsf{like}(x, y, s), \\ l_2 : p = \lambda s.\boxed{7}, \quad q_3 : \lambda p.p(\boxed{0}) \wedge \boxed{5}, \\ l_4 : \mathsf{some}(x, \boxed{9}, \boxed{10}),\ l_5 : \mathsf{person}(x, \boxed{17}), \\ l_6 : \mathsf{every}(y, \boxed{14}, \boxed{15}),\ l_7 : \mathsf{person}(y, \boxed{18}), \\ \boxed{7} \geq l_1, \boxed{5} \geq l_2 \\ \boxed{5} \geq l_4, \boxed{9} \geq l_5, \boxed{10} \geq l_2 \\ \boxed{7} \geq l_6, \boxed{14} \geq l_7, \boxed{15} \geq l_1 \end{array}$$

As intended, (18) allows only one disambiguation, namely $\boxed{5} \rightarrow l_4$, $\boxed{9} \rightarrow l_5$, $\boxed{10} \rightarrow l_2$, $\boxed{7} \rightarrow l_6$, $\boxed{14} \rightarrow l_7$, $\boxed{15} \rightarrow l_1$. The situation indices $\boxed{0}$ and $\boxed{17}$ default to s_0 and the value of $\boxed{18}$ remains underspecified (it could be s_0 or s). This leads to $q_3 : \lambda p.p(s_0) \wedge \mathsf{some}(x, \mathsf{person}(x, s_0), p = \lambda s.\mathsf{every}(y, \mathsf{person}(y, s/s_0), \mathsf{like}(x, y, s)))$.

3.3 Multiple wh-questions

A more complex example is (6) *who seemed to introduce who to everybody*, where two wh-quantifiers (one of them in situ) interact with a raising verb and a non-wh-quantifier. In order to treat in situ wh-quantifiers correctly, it must be possible to obtain the minimal scope of wh-quantifiers from any NP substitution node. Therefore, in NP substitution nodes we have to provide both, the minimal scope of wh-quantifiers and the minimal scope of non-wh-quantifiers. In the case of *like* in Fig. 4 for example, the minimal scope of *who* is l_2 while the minimal scope of *everybody* is l_1. We will use the feature WP for the first and the feature P for the second. For example, at the object substitution node in the tree for *introduce* in Fig. 5 we put a P value (as before) and additionally a WP value in case a wh-quantifier is added.

The derivation of (6) *who seemed to introduce who to everybody* and its semantic analysis are shown in Fig. 5. The raising verb in (6) adjoins to the VP node. This means that its label l_8 will become the value of the top P feature $\boxed{6}$ of the VP node, which is below the MAXS feature $\boxed{4}$ for non-wh-quantifiers (see the constraint $\boxed{4} \geq \boxed{6}$ in the semantics of *introduce* in Fig. 5). The scope trees of the wh-quantifiers adjoin both to the S' node, i.e., their scopes are limited by the MAXS value $\boxed{5}$ of the root. And, because of the WP features, both wh-quantifiers take scope over the proposition l_2 containing $\boxed{7}$, equated in turn with the non-wh MAXS value $\boxed{4}$ ($\boxed{7}$= $\boxed{4}$ by T/B unification in S of *introduce*). Consequently, we obtain the following scope orders: the two wh-quantifiers have both scope over *seem* and *everybody*, but the scope order of the raising verb and the non-wh-quantifier is unspecified.

3.4 Long-distance wh-dependencies

In long-distance wh-dependencies as (19) one also wants to obtain an interpretation where the wh-quantifier takes scope over all verbs in the sentence while providing the argument of the most embedded verb. Such examples have always been claimed to be problematic for derivation tree based LTAG semantics approaches (see (Kallmeyer and Romero, 2004) and the literature cited there).

(19) Who does Paul think John said Bill liked?

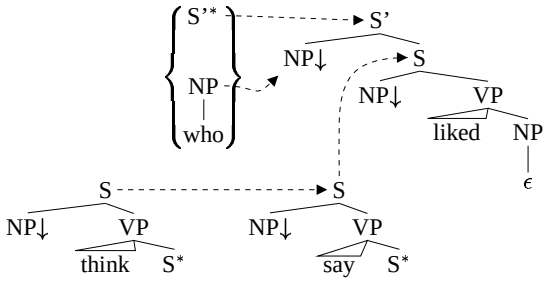
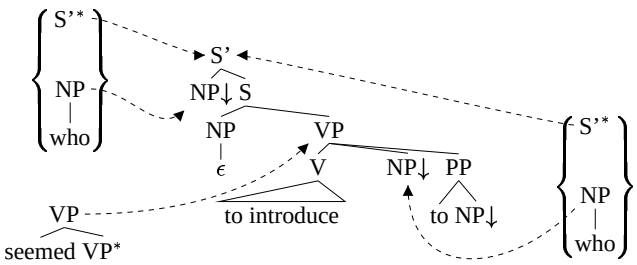


Figure 6: Derivation of (19)

The syntactic analysis of (19) (see (Kroch, 1987)) is shown in Fig. 6, and the combination of *like*, *say* and *think* in the semantics is shown in Fig. 7. Each of the attitude verbs takes the bottom MAXS proposition of the S node as its argument and it gives a larger proposition with a new (higher) bottom MAXS value. In the end, the highest of these MAXS values is unified with the top MAXS of the S node (i.e., with $\boxed{7}$). Therefore, all attitude verbs are embedded under the top MAXS value of the S node of *like* which is in the scope of any wh-quantifier added to *like*. In this way the correct scope analyses for wh-quantifiers in long-distance dependencies are obtained. The initial NP tree of such a quantifier is of course as before substituted for the corresponding argument position in *like* which leads to the correct predicate argument dependencies.

3.5 Comparison with other approaches to the scope of wh-phrases

The Karttunen-style semantic tradition ((Lahiri, 1991), (Chierchia, 1993), among many others), within the Montagovian Formal Semantics framework, draws the distinction between wh-scope and non-wh-scope by basing the semantics on the derived tree and using different semantic types for the relevant nodes. The S node has the propositional type <s,t>, and the semantics of non-wh-quantificational elements operates on functions of that type. The S' node (or, more specifically, the C' node) has the type <s,<<s,t>,t>>> corresponding to functions from situations to sets of propositions, and wh-quantifiers must combine with functions of such type. This derives



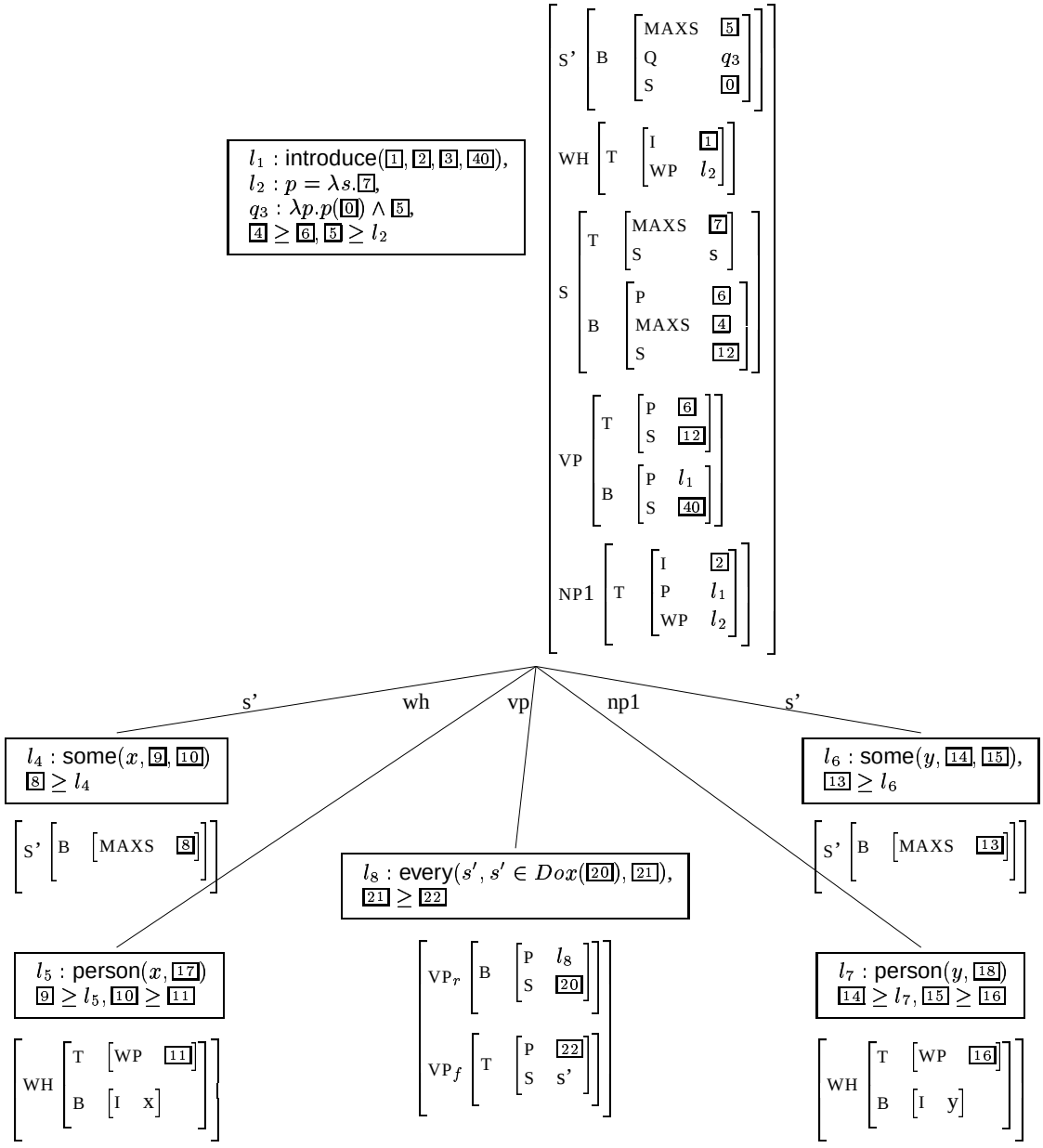


Figure 5: Abriged derivation and derivation tree with semantics (without quantifier *everybody*) for (6) *who seemed to introduce who to everybody*

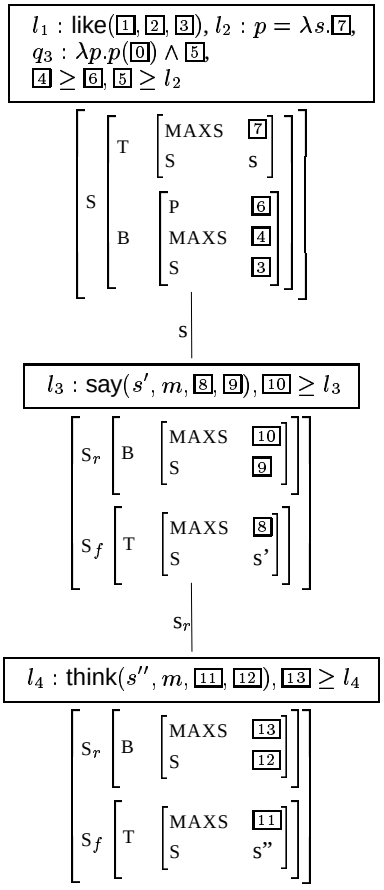


Figure 7: Abriged derivation tree and semantics for (19)

the effect that all wh-quantifiers must scope over all the non-wh-quantifiers.

A comparable approach using semantic features is developed in (Ginzburg and Sag, 2000), who make an ontological distinction between states-of-affairs (SOAs) and propositions. A verb introduces a SOA, which is the original building block from which later one builds propositions, questions, outcomes and facts. The idea is that a non-wh-quantifier has a SOA as its nuclear scope, and a wh-phrase has a proposition as its nuclear scope. Hence, wh-phrases necessarily have wider scope than non-wh-quantifiers in their clause.

The present approach provides an account of the scopal properties of wh- and non-wh-quantifiers within a 'flat' semantics framework in the style of MRS (Copestake et al., 1999) without invoking finer ontological distinctions. The semantic contribution of each elementary and auxiliary tree is a set of formulae (type t, the extensional version of propositions). Such a flat approach simplifies the design of algorithms for semantic computation as explained in (Copestake et al., 1999). Since the semantic material that will end up in the nuclear scope of a wh- and non-wh-quantifier is invariably introduced as a formula, no type distinction can be made to which the scopal properties of wh- and non-wh-quantifiers could relate. Furthermore, no ontological distinction between state-of-affairs and propositions is used to make scope follow from selectional properties. Instead, the present account proposes to define appropriate scope windows using the features MAXS, P and WP and feature unification.[5]

4 Embedded interrogatives

We have seen that the elementary tree for verbs includes formulae with situation arguments, e.g. l_1 : laugh(1, 2) in Fig. 3 and l_1 : introduce(1, 2, 3, 40) in Fig. 5. When no operator binds that variable, it defaults to the utterance situation s_0, as we saw for 2 in *laugh* in (16), section 3.1. Otherwise, the situation variable must be bound by some operator, using feature unification: e.g., 40 is bound by the $\forall s'$-quantifiction introduced by *seems* in Fig. 5 (40 $= s'$ by adjunction of *seems* to VP).

In the case of $q_3 : \lambda p.p($0$) \wedge$ 5 in an interrogative verb tree, we also have a situation variable 0 that, if unbound, will default to s_0, as noted for (18). The issue is how this situation variable becomes bound when the interrogative clause is embedded under, e.g., *know*. Note that, in the final semantics for *John knows who called* in (12), repeated as (20) below, the semantic contribution of the embedded interrogative has to be used twice, once evaluated for the doxastic situation s' and once for the utterance situation s_0. But, if we take $q_3 : \lambda p.p($0$) \wedge$ 5 in any of the derivations above and we simply perform feature unification to the extent that 0 $= s'$, q_3 will invariably amount to $\lambda p.p(s') \wedge$ 5 all the times it is used. The question is, thus, how to achieve the effect that 0 is replaced by s' in one occurrence of the formula and by s_0 in another.

(20) $\lambda s_0.\forall s'_s \in Dox_j(s_0)\forall p_{<s,t>}$
$[\exists x[\text{person}(x, s') \wedge p(s') \wedge p = \lambda s.\text{call}(x, s)]$
$\leftrightarrow \exists x[\text{person}(x, s_0) \wedge p(s_0) \wedge p = \lambda s.\text{call}(x, s)]]$

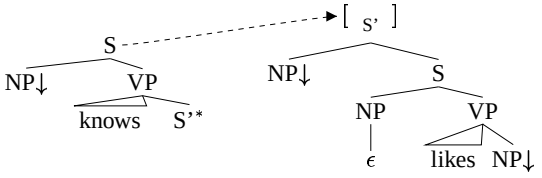


Figure 8: Derivation of (4)

Our analysis of (4) *John knows who likes everybody* is given in Fig. 8 and Fig. 9. To obtain the desired effect, we propose that the semantics of the verb tree for *know* includes a $\lambda s''$ that will bind 0 in both occurrences of

[5] A third approach treats wh-phrases, along with indefinites, as open formulae whose variable is bound by an unselective binder (Berman, 1991). As we treat indefinites as contributing their own quantificational force, we do the same for wh-phrases.

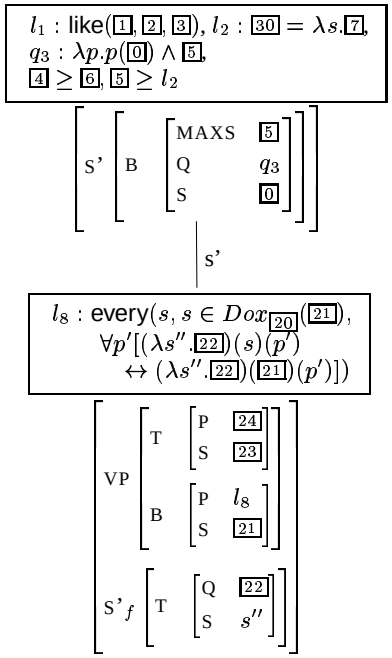


Figure 9: Abriged derivation tree and semantics for (4)

q_3. This is achieved by adding the situation feature S 0 at the S' position of interrogative *like*, which will unify with the feature S s'' at the foot of *know*. As a result, within l_8 of *know* we have the newly created expression $\lambda s'' \lambda p.p(s'') \wedge$ 5, arising from $\lambda s''.$22 and from 22 $= q_3$ by adjunction of *know* to the S' of *like*. Then, l_8 includes the new λ-expression twice: once it applies it to the doxastic sitation s, and once it combines it with the situation index 21. Index 21 (and 17 below) is left unbound and will thus default to the situation s_0 of the whole proposition. Finally, by substitution of *John*, 20 is identified with x. The result of the computation is given in (21).[6]

(21)
john(x), l_1 : like(x, y, s)
l_2 : $p = \lambda s.$7, q_3 : $\lambda p.p(s'') \wedge$ 5,
l_4 : some($x,$ 9, 10), l_5 : person(x, s_0),
l_6 : every($y,$ 14, 15), l_7 : person($y,$ 18)
l_8 : every($s, s \in Dox_x(s_0)$,
$\forall p'[(\lambda s''.q_3)(s)(p')$
$\leftrightarrow (\lambda s''.q_3)(s_0)(p')]$),
7 $\geq l_1$, 5 $\geq l_2$, 5 $\geq l_4$, 9 $\geq l_5$, 10 $\geq l_2$
7 $\geq l_6$, 14 $\geq l_7$, 15 $\geq l_1$

5 Conclusion

In sum, we have proposed an account for the semantics of wh-questions in LTAG that captures the different scope properties of wh- and non-wh-quantifiers and that derives the adequate semantics for embedded interrogative clauses.

[6] In the case of direct questions Q, we can assume that their truth-conditional content amounts to the proposition expressed by *I want to know* Q. For weaker degrees of exhaustivity of direct and embedded questions compatible with the present approach, see (Beck and Rullmann, 1999) and (van Rooy, 2003).

Acknowledgments

For many fruitful discussions we would like to thank Aravind K. Joshi and all members of the XTAG Group at the University of Pennsylvania. Furthermore, we are grateful to two anonymous reviewers for their useful comments.

References

Sigrid Beck and Hotze Rullmann. 1999. A Flexible Approach to Exhaustivity in Questions. *Natural Language Semantics*, (7):249–298.

Steven Berman. 1991. *The Semantics of Open Sentences*. Ph.D. thesis, UMass, Amherst.

Gennaro Chierchia. 1993. Questions with quantifiers. *Natural Language Semantics*, (1):181–234.

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 1999. Minimal Recursion Semantics. An Introduction. Manuscript, Stanford University.

Claire Gardent and Laura Kallmeyer. 2003. Semantic Construction in FTAG. In *Proceedings of EACL 2003*, Budapest.

Jonathan Ginzburg and Ivan A. Sag. 2000. *Interrogative Investigations. The Form, Meaning, and Use of English Interrogatives*. CSLI.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2003. Flexible Composition in LTAG: Quantifier Scope and Inverse Linking. In Harry Bunt, Ielka van der Sluis, and Roser Morante, editors, *Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5*, pages 179–194, Tilburg.

Laura Kallmeyer and Aravind K. Joshi. 2003. Factoring Predicate Argument and Scope Semantics: Underspecified Semantics with LTAG. *Research on Language and Computation*, 1(1–2):3–58.

Laura Kallmeyer and Maribel Romero. 2004. LTAG Semantics with Semantic Unification. In *Proceedings of TAG+7*, Vancouver.

Lauri Karttunen. 1977. The syntax and semantics of questions. *Linguistics and Philosophy*, (1):3–44.

Anthony S. Kroch. 1987. Unbounded dependencies and subjacency in a Tree Adjoining Grammar. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 143–172. John Benjamins, Amsterdam.

Uptal Lahiri. 1991. *Embedded Interrogatives and Predicates That Embed Them*. Ph.D. thesis, MIT.

Orin Percus. 2000. Constraints on some other variables in syntax. *Natural Language Semantics*, (8):173–229.

Robert van Rooy. 2003. Quetioning to Resolve Decision Problems. *Linguistics and Philosophy*, (26):727–763.

Assigning XTAG Trees to VerbNet

Neville Ryant, Karin Kipper
University of Pennsylvania
200 South 33rd Street
Philadelphia, PA 19104 USA
nryant@unagi.cis.upenn.edu
kipper@linc.cis.upenn.edu

Abstract

This paper presents the mappings between the syntactic information of our broad-coverage domain-independent verb lexicon, VerbNet, to Xtag trees. This mapping between complementary resources allowed us to increase the syntactic coverage of our verb lexicon by capturing transformations of the basic syntactic description of the verbs present in VerbNet. In addition, having these two resources mapped allows the semantic predicates present in our lexicon to be used to disambiguate Xtag verb senses.

1 Introduction

The limited availability of large-scale lexical resources has restricted natural language applications to specific domains. We propose to fill this gap by creating VerbNet (Kipper et al., 2000a; Dang et al., 2000), a freely available broad-coverage verb lexicon. VerbNet includes mappings to other known resources so that they can be used as extensions of each other.

VerbNet is a domain-independent verb lexicon with explicit syntactic and semantic information for over 4,000 English verbs. The verbs are organized in classes according to Levin's classification (Levin, 1993). In order to retain common syntactic and semantic properties for all members of a class, our verb classes are hierarchically organized, with 74 new subclasses added to the original classes. The syntactic frames represent the surface structure of constructions such as transitives, intransitives, prepositional phrases, resultatives, and other alternations listed in Levin.

The verbs in our lexicon have been mapped to WordNet (Miller, 1985; Fellbaum, 1998) and more recently to FrameNet (Baker et al., 1998). The syntactic coverage of VerbNet has been tested against the frames found in PropBank (Kingsbury and Palmer, 2002) through a systematic mapping between the two resources. The syntactic frames in our verb lexicon account for over 78% exact matches to the frames found in PropBank (Kipper et al., 2004).

A natural extension of VerbNet's syntactic frames is to incorporate the possible transformations of each frame. The Xtag grammar (XTAG Research Group, 2001) presents a large existing grammar for English verbs that accounts for just that richness of constructions. Mapping our syntactic frames to the Xtag trees greatly increases the robustness of our resource by capturing such transformations.

2 Levin Classes

Levin verb classes (Levin, 1993) are based on the ability of a verb to occur in pairs of syntactic alternations which preserve the intended meaning. The fundamental assumption of Levin classes is that the syntactic behavior of verbs is a direct reflection of the underlying semantics. This is a not uncontroversial thesis in its strongest form, but it is indisputable that meaning can have great predictive ability. Hale and Keyser (1987) discuss the predictive ability of lexical semantic knowledge using the archaic whaling term *gally*, which might be interpreted to mean *see* or possibly *frighten.* Depending on the assumption made about galley's meaning, speakers can make conflicting judgments about the verb's syntactic behavior. For the speaker interpreting it to mean *see,* the middle construction is disallowed "*Whales gally easily" (paralleling "Whales see easily"), while for the speaker who interprets it as *frighten*, the middle construction is allowed. For an example from Levin, consider the classes of the *break* verbs and the *cut* verbs which are similar in the ability of their members to participate in the transitive and middle constructions. Additionally, *break* verbs may appear in the simple intransitive construction while *cut* verbs may appear in the conative construction. The ex-

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 194-198.

planation given by Levin is that the *cut* verbs describe a series of actions with the goal of separating some entity into pieces. Whether the goal is achieved or not, the action can still be performed, as recognized by "John cut at the loaf." For the break verbs, the verb specifies the manner that a resultant change of state occurs. The action of breaking cannot be attempted if no result is achieved and so these verbs disallow "*John broke at the window".

3 VerbNet

VerbNet is a verb lexicon with syntactic and semantic information for English verbs, referring to Levin verb classes (Levin, 1993) to construct the lexical entries. It exploits the systematic link between syntax and semantics that motivates these classes, and thus provides a clear and regular association between syntactic and semantic properties of verbs and verb classes (Kipper et al., 2000a; Dang et al., 2000). Each node in the hierarchy is characterized extensionally by its set of verbs, and intensionally by syntactic and semantic information about the class and a list of typical verb arguments. The argument list of each entry consists of thematic labels and possible selectional restrictions on the arguments expressed using binary predicates. The syntactic information in each verb's entry maps the list of thematic arguments to the deep-syntactic arguments of that verb (normalized for voice alternations, and transformations). The semantic predicates list the participants during various stages of the event described by the syntactic frame.

The syntactic frames act as a short-hand description for the surface realizations allowed for the members of the class. They describe constructions such as transitive, intransitive, prepositional phrase complement, resultative, and a large set of Levin's alternations. A syntactic frame consists of the verb itself, the thematic roles in their preferred argument positions around the verb, and other lexical items which may be required for a particular construction or alternation. Additional restrictions may be further imposed on the thematic roles (quotation, plural, infinitival, etc.). Examples of syntactic frames are *Agent V Patient* (e.g., John hit the ball), *Agent V at Patient* (e.g., John hit at the window), and *Agent V Patient[+plural] together* (e.g., John hit the sticks together).

The semantic information for the verbs is expressed as a conjunction of semantic predicates, such as *motion, contact, transfer_info*. For the same verb, each different alternation typically has a slightly different set of semantic predicates, although there is usually a substantial overlap within a class. The predicates can take arguments over the verb complements, as well as over implicit existentially quantified event variables.

4 Compositional Semantics for VerbNet

Several attempts have been made to use LTAG derivation trees to compute compositional semantics. Stone and Doran (1997) describe a system for incorporating semantics into TAG trees by a system that simultaneously constructs the semantics and syntax of a sentences using LTAGS. Each lexical item anchors a tree or family of trees and associates with each tree a logical form representing the semantic and pragmatic information for that lexical item and tree. The meaning of a sentence is computed by the conjunction of the meaning of the elementary trees used in the derivation.

Joshi and Vijay-Shanker (1999) and Kallmeyer and Joshi (1999) describe the semantics of the derivation tree as a set of attachments to trees. For each attachment, the semantics are defined as a conjunction of formula in a flat semantic notation. They provide an explicit methodology for composing semantic representations.

Kipper et al (2000) present a method for deriving compositional semantic interpretations from sentences using VerbNet. The mappings discussed here are a step closer to that proposal.

5 Extending VerbNet with XTAG

VerbNet, while providing an explicitly constructed verb lexicon with syntax and semantics, offers limited syntactic coverage since it describes only the declarative frame for each syntactic construction or alternation. The Xtag grammar, on the other hand, is a lexical resource with well-characterized syntactic descriptions for lexical items but makes no distinctions between verb senses and currently has contains no explicit semantics. An obvious way to extend VerbNet's syntactic coverage is to incorporate the coverage of Xtag, accounting for the possible transformations of each declarative frame. Presumably, transformations of VerbNet's syntactic frames are recoverable by mapping onto elementary trees of TAG tree families. Then, for any verb in VerbNet each thematic role can be mapped to an indexed node in the basic syntactic tree and the selectional restrictions on VerbNet thematic roles to features on the nodes. In addition to increasing the coverage of VerbNet, this provides us with a pre-existing parser for computing derived and derivation trees to which our semantic predicates can be added and therefore sense distinctions can be made more explicit.

5.1 Mapping VerbNet frames to XTAG

Each frame in VerbNet is described by 4 components: 1) a brief text description (such as *Transitive, Resultative*), 2) an example sentence, 3) a syntactic frame, 4) a semantic description using a set of semantic predicates. Text descriptions and syntactic frames are very much interrelated, but the text description is independent of the roles

assigned to the verb's arguments. These text descriptions consist of both primary and secondary descriptions which were made completely consistent for the whole VerbNet lexicon prior to these mappings. Examples of primary descriptions include *Transitive, Material/Produce Alternation, and Ergative*. Secondary descriptions provide additional information about the semantics and/or syntax. These might specify the types of prepositional phrases that a verb may take or the existence of restrictions on a complement (often secondary descriptions are used to distinguish between different types of sentential complements). Examples (1) and (2) show how the first three components of VerbNet frames are described:

(1) *Material/Product Alternation Intransitive (Material Subject)*
"That acorn will grow into an oak tree."
Material V Prep(into) Product

(2) *Material/Product Alternation Intransitive (Product Subject)*
"An oak tree will grow from that acorn."
Product V Prep(from out of) Material

Because secondary descriptions sometimes refer to variants of a frame that correspond in Xtag to an entirely different tree family than the original frame, it is necessary to consider both these descriptions in order to uniquely identify frames. For instance the Benefactive Alternation in VerbNet has two variants as shown in examples (3) and (4):

(3) *Benefactive Alternation (for variant)*
"Martha carved a piece of wood for the baby"
Agent V Material Prep(for) Beneficiary

(4) *Benefactive Alternation (double object)*
"Martha carved the baby a toy out of a piece of wood"
Agent V Beneficiary Product Prep(from out of) Material

In the current Xtag grammar example (3) corresponds to the tree family of Ditransitives with a PP complement (Tnx0Vnx1pnx2), derived from the simple Transitive tree family with the PP anchored by *for* adjoined into the tree at the VP node, whereas example (4) corresponds to the Ditransitive tree family (Tnx0Vnx2nx1). VerbNet however can only discriminate between the two frames with both the primary and secondary descriptions. Each syntactic frame, then, is assumed to be uniquely specified by its primary and secondary descriptions. Generally, the VerbNet syntactic frame specified by a full description corresponds to the surface syntactic realization of an Xtag elementary tree. Mappings between VerbNet syntactic frames and Xtag tree families were done manually, using the latest frozen release of the XTAG grammar and the latest version of VerbNet. Each VerbNet syntactic frame was mapped to a corresponding Xtag tree family, with the index of the tree family recorded in the VerbNet entry. In theory we should be able to annotate each unique VerbNet syntactic frame with a mapping to an Xtag elementary tree. However, there currently are two impediments to doing this:

1. Many VerbNet syntactic frames specify surface realizations of trees that would not be regarded as initial trees in the Xtag framework (though it is possible to regard them as such by violating certain fundamental assumptions of the grammar). The canonical example is where VerbNet includes as part of a frame a PP that Xtag would analyze as an adjunct.

2. Not all VerbNet syntactic frames correspond to an Xtag elementary tree.

The first issue includes certain verbs appearing in the *Induced Action* alternation, for example, and many of the transitive frames that additionally specify a path PP to indicate the direction of the action. In Example (5) the Xtag grammar analyzes 'over the fence' as an adjunct, this analysis is based on the fact that this PP is optional for the grammaticality of the sentence. Consequently, verbs taking this frame should map to the Transitive tree family, the tree corresponding to the VerbNet frame's overt syntax being derived by adjunction into the elementary tree of the auxiliary tree of 'over the fence.' As an example, consider the two frames in (5) and (6), both of which have PP adjuncts under the Xtag analysis:

(5) *Induced Action (with accompanied motion and path PP)*
"Tom jumped the horse over the fence"
Agent V Theme Prep[+spatial] Location

(6) *Transitive (+ path PP)*
"Jackie accompanied Rose to the store" Agent V Theme Prep[+loc OR +path]

This is similarly an issue with intransitives followed by a PP. Xtag grammar guidelines specify that no verb should appear both in Tnx0v (the tree family for purely intransitive verbs that can be followed by a prepositional phrase but do not require one to be grammatical) and also in Tnx0Vpnx1 (the tree family for intransitive verbs that must be followed by a prepositional phrase to be grammatical). In VerbNet many verbs participate in the *Conative* Alternation, in which the a transitive frame alternate with an intransitive frame in which the NP object is replaced with a PP fronted by 'at.' Examples (7) and (8) show a conative frame in VerbNet and its transitive equivalent respectively:

(7) *Conative*
"Carol cut at the bread"

Agent V at Patient

(8) *Basic Transitive*
"Carol cut the bread"
Agent V Patient

While most of the VerbNet verbs that take the conative do not have intransitive forms that are grammatical when not followed by a PP, many can appear in such frames. For example, the verbs *cut, hack, hew, scrape, scratch, shovel, and dust* constitute a partial listing of the verbs taking the conative and that can also appear as bare intransitives. There remains the question, then, of how the conative alternation should be handled by VerbNet. In instances such as these, as we are interested merely in recovering the various transformationally related forms of frames, we simply ignore the constraints of the Xtag grammar, in the case of (5) and (6) mapping to the elementary tree anchoring tree family Tnx0Vnx1Pnx2, the tree family of verbs taking an NP complement followed by a PP complement headed by a particular preposition. In the case of (7) and (8), mapping to Tnx0Vpnx1, the tree family of intransitives with PP complements.

With regard to the second of these considerations, not all VerbNet syntactic frames correspond to some Xtag elementary tree. A number of VerbNet classes contain syntactic frames that specify multiple adjuncts. As a case in point consider VerbNet class `turn-26.6`, with members *alter, metamorphose, transform, transmute, change, convert, and turn*. Each of these can appear in the two frames presented in (9) and (10):

(9) *Causative/Inchoative Alternation (causative, + Material + Product)*
"The witch turned him from a prince into a frog"
Agent V Patient Prep(from) Material Prep(into) Product

(10) *Causative/Inchoative Alternation (inchoative, + Material + Product)*
"He turned from a prince into a frog"
Patient V Prep(from) Material Prep(into) Product

In the Xtag grammar, the frame presented in (9) corresponds to no elementary tree of any tree family. One might disagree over what elementary tree it is derived from. For instance, (9) can be seen as a transitive sentence with PP adjuncts (and thus belonging to tree family Tnx0Vnx1), as a ditransitive taking a PP complement with another PP adjunct (tree family Tnx0Vnx1pnx2), or as a resultative with a PP anchor and an additional PP adjunct (and thus belonging to tree family TRnx0Vnx1Pnx2). Similarly, the frame presented in (10) can be seen as either an intransitive sentence with optional PP adjuncts (Tnx0V), or as a resultative with ergative verb and PP anchor (TREnx1VPnx2). In the current version of VerbNet we have 18 syntactic frames that fall into this category. Some of these are frames that have been added to VerbNet during attempts at expanding syntactic coverage. Others, such as the *middle construction*, are based on the original alternations proposed by Levin. Currently, these frames are also mapped to the Xtag elementary tree from which they are derived, but it is noted that they are not initial trees. In the future, some of these frames may be removed (in the cases where they are not crucial to characterizing the structure of classes) and for others, it should be specified if they are derived from an initial tree.

5.2 Coverage

As of the latest release of VerbNet, there are 196 unique frames (as distinguished by primary and secondary description). Of these, all but 18 correspond exactly to some Xtag elementary tree (the exceptions are discussed above). For these 168 VerbNet syntactic frames that map exactly to an Xtag elementary tree, only 16 of the 57 Xtag elementary trees were used. A detailed inspection on the 41 Xtag tree families with no corresponding VerbNet frame, revealed that 22 of them deal with small clauses, 8 with idiomatic expressions, and 9 with other various classes. However, some of VerbNet's syntactic frames quite simply are not able to be parsed by the Xtag grammar. The current Xtag analysis for PPs analyzes PP complements with an expanded PP structure rather than as a PP substitution node contrasts these approaches. This is done as expansion of the PP makes the NP node of the PP available to the metarules for creating the trees for extraction so that sentences such as (12) are derivable from (11).

(11) *Jill placed her handbag on the table.*

(12) *What 1 did Jill put her handbag on t1?*

However, Xtag's explicit realization of NPs in complement PPs precludes handling of incidences of exhaustive PP substitution. Thus, Xtag does not handle verbs that take an exhaustive PP such as 'here' (or 'there,' 'somewhere,' etc) as an argument. As such, sentences such as (13) currently cannot be handled, and therefore certain frames in VerbNet (namely, the Transitive (+ here/there)) construction simply have no Xtag mapping.

(13) *I spooned the sauce there.*

6 Conclusion

We presented a detailed account of our mappings between our broad-coverage verb lexicon with explicit semantics, VerbNet, and a syntactically rich lexical resource, the Xtag grammar. By incorporating the transformations of the basic frames from Xtag to our syntactic

frames in VerbNet we are able to greatly increase the robustness of our resource by indirectly providing a much larger syntactic coverage. In addition to increasing the coverage of VerbNet, these mappings supply us with a pre-existing parser for computing derived and derivation trees to which our semantic predicates can be associated thus helping the task of verb sense disambiguation.

Acknowledgements

This research was partially supported by NSF Grants IIS-9900297, IIS-0325646, DARPA Tides Grant N66001-00-1-891 and ACE Grant MDA904-00-C-2136.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING/ACL-98)*, pages 86–90, Montreal. ACL.

Hoa Trang Dang, Karin Kipper, and Martha Palmer. 2000. Integrating compositional semantics into a verb lexicon. In *Proceedings of the Eighteenth International Conference on Computational Linguistics (COLING-2000)*, Saarbrücken, Germany, July-August.

Christiane Fellbaum, editor. 1998. *WordNet: An Eletronic Lexical Database*. Language, Speech and Communications. MIT Press, Cambridge, Massachusetts.

Kenneth Hale and Samuel Jay Keyser. 1987. *A view from the Middle*. Lexicon Project Working Papers 10, MIT, Cambridge, MA.

Aravind K. Joshi and K. Vijay-Shanker. 1999. Compositional Semantics with Lexicalized Tree-Adjoining Grammar: How Much Under-Specification Is Necessary? . In *Proceedings of the Third International Workshop on Computational Semantics (IWCS-3)*, pages 131–145, Tilburg, The Netherlands, January.

Laura Kallmeyer and Aravind K. Joshi. 1999. Underspecified Semantics with LTAG . In *Proceedings of Amsterdam Colloquium on Semantics*.

Paul Kingsbury and Martha Palmer. 2002. From treebank to propbank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*, Las Palmas, Canary Islands, Spain.

Karin Kipper, Hoa Trang Dang, and Martha Palmer. 2000a. Class-based construction of a verb lexicon. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, July-August.

Karin Kipper, Hoa Trang Dang, William Schuler, and Martha Palmer. 2000b. Building a class-based verb lexicon using tags. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 147–154, Paris, France, May.

Karin Kipper, Benjamin Snyder, and Martha Palmer. 2004. Extending a verb-lexicon using a semantically annotated corpus. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-04)*, Lisbon, Portugal.

Beth Levin. 1993. *English Verb Classes and Alternation, A Preliminary Investigation*. The University of Chicago Press.

George Miller. 1985. Wordnet: A dictionary browser. In *Proceedings of the First International Conference on Information in Data*, Waterloo, Ontario.

Matthew Stone and Christine Doran. 1997. Sentence Planning as Description Using Tree Adjoining Grammar. In *Proceedings of ACL-EACL '97*, Madrid, Spain.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Nondeterministic LTAG Derivation Tree Extraction

Libin Shen
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
libin@linc.cis.upenn.edu

Abstract

In this paper we introduce a naive algorithm for nondeterminisctic LTAG derivation tree extraction from the Penn Treebank and the Proposition Bank. This algorithm is used in the EM models of LTAG Treebank Induction reported in (Shen and Joshi, 2004). Given the trees in the Penn Treebank with PropBank tags, this algorithm generates shared structures that allow efficient dynamic programming in the EM models.

1 Introduction

In recent years, the statistical approach has been successfully used in natural language processing (NLP). No matter which statistical model people use, a generative model or statistical machine learning, large corpora are always needed to train the models. For example, after the introduction of the Penn Treebank (PTB) (Marcus et al., 1994), a serial of improvements has been achieved on natural language parsing and shallow parsing tasks. In the field of Lexicalized Tree Adjoining Grammar (LTAG), the statistical approach has also been successfully employed in many LTAG-based NLP tasks, such as LTAG parsing (Chiang, 2000; Shen et al., 2003) and Supertagging (Joshi and Srinivas, 1994).

However, the lack of very large corpora based on LTAG prevents the statistical approach from being widely used in the field of LTAG. As we know, very large corpora are crucial to statistical NLP. In previous works, people managed to induce LTAG style grammars and LTAG based corpora from the PTB, and use them in their applications.

Joshi and Srinivas (1994) first implemented a supertag corpus by extracting it from the PTB, using heuristic rules. Due to various limitations of this system, extracted supertags of the words in a sentence cannot always be successfully put together. Xia (2001) and Chen (2001) described deterministic systems that extract LTAG-style grammars from PTB. In their systems, the *head percolation table* (Magerman, 1995) and the PTB functional tags were used to solve ambiguities in extraction. Chiang (2000) reported a similar method to extract an LTAG like treebank from PTB, and used it in a statistical parser. Shen et al. (2003) employed a similar technique to induce an LTAG treebank, to be used in a parse reranking system.

In these LTAG grammar and treebank induction systems, deterministic rules were used to solve the ambiguities in the elementary tree extraction process. However, it is clear that deterministic rules are not enough to solve ambiguity in extraction, especially in the case of the argument-adjunct distinction (Paola and Leybold, 2001).

2 A Statistical Model

In (Shen and Joshi, 2004), we have proposed a statistical model for LTAG Treebank induction using the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). The EM Algorithm is a general iterative method of search to find the maximum-likelihood estimate of the parameters of the hidden data from the observed data.

If we take the PTB as the observed data, then the LTAG derivation trees for the PTB trees can be treated as the hidden data. Then our goal is the find out the hidden structures, or LTAG derivation trees, with maximum-likelihood. Similar idea was previously employed in (Chiang and Bikel, 2002) in statistical parsing.

In (Shen and Joshi, 2004), several EM models were proposed for LTAG Treebank induction. In these models, linguistic knowledge is used to overcome EM's weakness that EM cannot guarantee to find a global optimum. By using linguistic knowledge, we can not only start the EM iteration from the point close to the global optimum in the first iteration, but also limit the search space of hidden derivation trees in the following rounds.

However in that paper, we did not give the details on how to employ the linguistic knowledge to constrain the search space. In this paper, we will introduce a novel

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 199-203.

algorithm that searches the space the derivation trees, respecting the linguistic constraints and maintaining the ambiguities in elementary tree extraction, for each given PTB tree. In this section, we first analyze the ambiguities existing in LTAG elementary tree extraction from PTB, and then we illustrate the linguistic information to be used in nondeterminisctic derivation extraction.

2.1 Ambiguities

Our analysis is mainly based on the work of (Xia, 2001) and (Chen, 2001). There are two kinds of ambiguities in LTAG elementary tree extraction, which are *head competition* and *argument-adjunct distinction*.

Given a deduction rule of Context Free Grammar (CFG) in the PTB, we need to find out which item on the right hand side of the rule is the *lexical head* of the item on the left hand side. For example, for rule VP $\rightarrow$ V NP, V is the head of VP. In (Xia, 2001) and (Chen, 2001), the so called head percolation table (Magerman, 1995) is used to determine the head item of each CFG rule in the PTB. Following the lexical heads in a tree, we can get the spines of elementary trees.

Argument-adjunct distinction is mainly to distinguish the arguments and adjuncts of a predicate. In (Xia, 2001) and (Chen, 2001), argument-adjunct distinction is solved with respect to the constituent tags and function tags in the PTB.

In their systems, these two kinds of ambiguities are solved deterministically as we described above. However, there exists a lot ambiguities that can not been solved easily. For example, in the sentence ... *was named a nonexecutive director of this British industrial conglomerate* extracted from the PTB, the NP dominating *director* has a function tag PRD, which means predicate. In this case there exists a head competition between *director* and *named*. As far as argument-adjunct distinction is concerned, the ambiguities are ubiquitous, which can only be solved with a lexicon, i.e. the hand-crafted XTAG English Grammar (XTAG-Group, 2001), with statistical methods.

2.2 Linguistics Information

2.2.1 Penn TreeBank

The input of our algorithm is a full bracketed PTB tree, as shown in Figure 1. The structure information is the main source of derivation tree extraction. We can simply regard a PTB tree as a derived tree in LTAG.

Besides the structure information, The Penn Treebank provides more information useful in LTAG derivation tree extraction, such as special information for the predicate-argument structures. Although the PTB does not try to distinguish *arguments* and *adjuncts*, and treats them as arguments in general, it assigns functional tags for these arguments which help to to distinguish arguments and

```
TOP
   S
      NP-SBJ
         NP
            NNP Pierre
            NNP Vinken
         , ,
         ADJP
            NP
               CD 61
               NNS years
            JJ old
         , ,
      VP
         MD will
         VP
            VB join
            NP
               DT the
               NN board
            PP-CLR
               IN as
               NP
                  DT a
                  JJ nonexecutive
                  NN director
            NP-TMP
               NNP Nov.
               CD 29
      . .
```

Figure 1: PTB tree

adjuncts. For example, in Figure 1, functional tag *SBJ* means subject, *TMP* means temporal phrase, and *CLR* means closely related. This information was used in previous LTAG extraction systems and will also be used in our system too.

2.2.2 PropBank

We will also use the Penn Proposition Bank (PropBank) (Kingsbury and Palmer, 2002) in our LTAG derivation tree extraction algorithm. The PropBank provides more information on the predicate-argument structures for the PTB data.

In the PropBank, each predicate is assigned with a tag of *major sense* defined on usage of the predicate. Each argument of this predicate is assigned with an argument ID with respect to the *major sense* of this predicate, as shown in Figure 2. We will use the argument tags in our extraction algorithm too.

What is worth mentioning is that, like the PTB, the PropBank does not distinguish arguments and adjuncts, and both are called arguments in the PropBank. Using

```
TOP
   S
      [join:0_0_0:ARG0]NP-SBJ
         NP
            NNP Pierre
            NNP Vinken
         , ,
         ADJP
            NP
               CD 61
               NNS years
            JJ old
         , ,
      VP
         [join:1_0_0:ARGM]MD will
         VP
            [join:2_0_0:rel]VB join
            [join:3_0_0:ARG1]NP
               DT the
               NN board
            [join:4_0_0:ARGM]PP-CLR
               IN as
               NP
                  DT a
                  JJ nonexecutive
                  NN director
            [join:5_0_0:ARGM]NP-TMP
               NNP Nov.
               CD 29
      . .
```

Figure 2: PTB tree with PropBank tags

the PropBank tags, we can easily get all the arguments and adjuncts of a given predicate, but how to distinguish them is still a problem. This is the reason why we introduce the nondeterminisctic method in derivation tree extraction. The EM algorithms in (Shen and Joshi, 2004) are supposed to find out the global optimum over various selections.

3 Nondeterministic Derivation Tree Extraction

In this paper we will propose an algorithm to maintain the ambiguities in elementary tree extraction. For a given PTB tree, the derivation tree candidates serve as the search space used in the EM algorithm. A naive way is to represent all the candidate derivation trees one by one. However, a more efficient way to do this is to use shared structures. Therefore, we need an efficient way to represent all the LTAG derivation trees that meet the linguistic constraints, given a PTB tree. Then the EM algorithm can re-estimate expectation over all derivation tree candidates with less computational complexity by inside-outside algorithm as described in (Shen and Joshi, 2004).

3.1 Idea

We first give the idea of the extraction algorithm. The extraction consists of two phases, the bottom-up head competition phase and the top-down argument-adjunct distinction phase.

In the head-competition phase, we visit all the nodes throughout a PTB tree from bottom and look for the head candidates for every internal node. For each internal node, there will be several head candidates with respect to different triggering rules. For example, in the example given above, ... *was named a nonexecutive director of this British industrial conglomerate*, either the VP node anchored on *named* or the NP node anchored on *director* can be the head of the main S node with respect to different analyses.

If any head candidate is selected as the head for the parent node, the rest children nodes serve as

- a leaf node to which an initial tree substitutes,
- a leaf node to which an auxiliary tree adjoins, or
- an internal node of the elementary tree for the head node.

No matter what it will be, these elementary trees are beneath the elementary tree for the head node, in any *derivation* tree for this PTB tree. Thus, we can use shared structure to represent all these situations; each non-head child nodes can be used in three way as described above.

To sum up, for each head candidate, we can use an *index* structure to represent all the possible sub-structures beneath this elementary tree in the derivation tree, and this *index* structure will be further used for one or several times by the upper nodes. So the goal of the head competition phase is to search for all the head candidates for each internal node and to generate an *index* structure the represent all the sub-structures beneath this point.

In the argument-adjunct distinction phase, we visit *index* structures in a top-down style, starting from the *index* structures of the root node. Keeping track of the head child, we first get the spine of the elementary for the main predicate. Then we use linguistic information to get all the possible operations with which other sub-trees are attached to the sister nodes on different levels along the spine. The possible attachment methods, i.e. substitution, adjunction or internal node, are recorded in the index structures. Then we do argument-adjunct distinction recursively on all the subtrees rooted on these sister nodes.

Now we explain what an *index* structure is. Each node in a PTB tree is associated with a set of *index* structures.

An *index* structure stores the following information: its lexical head, a set of attachment points, and a set of the references of index structures of attached subtrees.

After the head competition phase and argument-adjunct distinction phase, we get a shared forest composed of *index* structures, with which we can get all the possible derivation trees of a PTB tree as well as the corresponding elementary trees.

3.2 Algorithm

The following algorithm is used to find all candidate derivation trees and store them in shared structures. The input is a PTB tree with PropBank tags, and the result of the algorithm is a shared forest that represents all candidate derivation trees for this input PTB tree.

1. head competition (node n)
   - 1.1 if (head-comp-done) return;
   - 1.2 for each child of n, call head competition;
   - 1.3 look for the candidate heads using linguistic constraints;
   - 1.4 for each candidate, generate an *index* structure and associate it with n;
   - 1.5 head-comp-done = true;
2. argument-adjunct distinction (node n)
   - 2.1 if (arg-adj-done) return;
   - 2.2 for each *index* structure of n, for each attachment candidate
     - refine attachment types;
     - call argument-adjunct distinction on the attached subtree;
   - 2.3 arg-adj-done = true;

3.3 Example

Figure 3 shows the results after head competition. For this case, there is no ambiguity on head competition, so there is only one output. Otherwise shared structures are used to represent all the results.

It is shown in Figure 3 that the result of the head competition does not distinguish arguments and adjuncts. The ambiguities are maintained in the single output in this case. Specifically, the PP subtree anchored on *as* can be either an argument or an adjunct. So do the NP subtrees for the subject, the object and the temporal phrase.

After the argument-adjunct distinction phase, ambiguities on the subject, the object and the temporal phrase are solved with respect to the templates defined on context, in a way similar to (Xia, 2001; Chen, 2001). However, the argument-adjunct ambiguity on the PP node is still maintained in the final output of the algorithm.

```
TOP
   [join:2_0_0:rel]S
      [join:0_0_0:ARG0]NP-SBJ
         %NP
            NNP Pierre
            %NNP Vinken
         , ,
         ADJP
            NP
               CD 61
               %NNS years
            %JJ old
         , ,
      %[join:2_0_0:rel]VP
         [join:1_0_0:ARGM]MD will
         %[join:2_0_0:rel]VP
            %[join:2_0_0:rel]VB join
            [join:3_0_0:ARG1]NP
               DT the
               %NN board
            [join:4_0_0:ARGM]PP-CLR
               %IN as
               NP
                  DT a
                  JJ nonexecutive
                  %NN director
            [join:5_0_0:ARGM]NP-TMP
               NNP Nov.
               %CD 29
      . .
```

Figure 3: PTB tree with head annotated. % stands for *head*

3.4 Discussion

In the previous sections, we did not cover the following two special situations in derivation tree extraction.

- Auxiliary predicate
- Coordination

In order to handle auxiliary predicate structure, we introduce a stack structure to maintain a head chain as described in (Chen, 2001). Since we have used the PropBank tags, we can easily recognize the predicate-argument relation between an argument associated with a sister node and a head in the stack.

Coordination is another important case in derivation extraction. In our current implementation, we use the first item in a coordination phrase as the head, and treat the rest items as adjuncts. For example, in the phrase *red and blue*, *red* is the head, *blue* attaches to *and*, and *and* adjoins to *red*. We are still working on the treatment of

coordination. One solution is to follow the approach described in (Sarkar and Joshi, 1996).

Theoretically, the number of the *index* structures grows exponentially with respect to the height of a tree. However, our experiments show that it does not grow that fast, thanks to the linguistic constraints used in head competition.

By using rich PropBank features, we can almost solve the ambiguities in head competition. The only thing that we need to take care of is the different analyses of head word in PropBank and LTAG. In some cases, PropBank and LTAG select different words as head. However, this problem can be solved with templates. For example, in the following example, *thought* is the head for the whole sentence in PropBank, while *come* is the head in LTAG analysis.

- *John thought Mary didn't come yet.*

4 Experiments

The nondeterminisctic LTAG derivation extraction algorithm was used in the EM models reported in (Shen and Joshi, 2004). With the algorithm given in that paper, about 12,000 elementary trees were extracted from the Penn Treebank. The experiments in (Shen and Joshi, 2004) showed that the number of the elementary trees was reduced to about 10,000 with several rounds of EM training.

It also noted in (Shen and Joshi, 2004) that some simple EM models reported in that paper prefer elementary trees of lower frequency, which is undesirable for grammar extraction. In our future research, we will incorporate the hand crafted XTAG English Grammar (XTAG-Group, 2001) in the EM models. Some XTAG Grammar based EM models were proposed in (Shen and Joshi, 2004) as future work.

5 Conclusions

In this paper we have introduced a naive algorithm for nondeterminisctic LTAG derivation tree extraction from the Penn Treebank and the PropBank, which is an extension of the deterministic methods in (Xia, 2001) and (Chen, 2001). The algorithm will be used in the EM models of LTAG Treebank Induction. The shared structures generated by this nondeterminisctic algorithm allow efficient expectation computation via dynamic programming in the EM algorithm.

References

J. Chen. 2001. *Towards Efficient Statistical Parsing using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

D. Chiang and D. M. Bikel. 2002. Recovering latent information in treebanks. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan.

D. Chiang. 2000. Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar. In *Proceedings of ACL-2000*.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B*, 39(1):1–38.

A. Joshi and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING'94*.

P. Kingsbury and M. Palmer. 2002. From treebank to propbank. In *Proceedings of the 3rd LREC*.

D. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd ACL*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1994. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

M. Paola and M. Leybold. 2001. Automatic distinction of arguments and modifiers: the case of prepositional phrases. In *Proceedings of the Fifth Workshop on Computational Natural Language Learning (CoNLL-01)*, Toulouse, France.

A. Sarkar and A. K. Joshi. 1996. Coordination in tree adjoining grammars: Formalization and implementation. In *Proceedings of COLING 1996*.

L. Shen and A. K. Joshi. 2004. Extracting Deeper Information from Richer Resource: EM Models for LTAG Treebank Induction. In *Proceedings of IJCNLP 2004*.

L. Shen, A. Sarkar, and A. K. Joshi. 2003. Using ltag based features in parse reranking. In *Proceedings of EMNLP 2003*.

F. Xia. 2001. *Automatic Grammar Generation From Two Different Perspectives*. Ph.D. thesis, University of Pennsylvania.

XTAG-Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report 01-03, IRCS, Univ. of Pennsylvania.

Deriving Syntactic Structure inside Ellipsis

Will Thompson
Northwestern University & Motorola Labs
wkt@motorola.com

Abstract

The proper linguistic representation of ellipsis has been a source of debate for years (Hankamer and Sag, 1976), with ellipsis theories broadly categorizable as being either *syntactic* or *semantic*, depending on whether or not an elided constituent is held to contain articulated syntactic structure. In this paper, I combine ideas from both syntactic and semantic theories in order to (1) account for the data that suggest there is syntactic structure within elided constituents, and (2) do so in a manner that preserves one of the prime advantages of existing semantic theories, namely straightforward declarative and procedural intepretations. This is accomplished by stating both semantic and syntactic identity conditions on ellipsis. The syntactic condition is formulated within a *desription theory* approach to grammar, as in the formalisms proposed in (Vijay-Shanker, 1992), (Rambow et al., 2001) and (Muskens, 2001).

1 Introduction

The proper linguistic representation of ellipsis has been a source of debate for years (Hankamer and Sag, 1976), with ellipsis theories broadly categorizable as being either *syntactic* or *semantic*, depending on whether or not an elided constituent is held to contain articulated syntactic structure. Recently the scales have tipped strongly in favor of syntactic approaches (see (Kennedy, 2003) for an overview). For example, the data in (1) ((Ross, 1969); see (Merchant, 2001) for extensive discussion) show that the wh-remnant of IP ellipsis ("sluicing") must bear the same case marking as its correlate in the antecedent, in those languages with overt case marking.

(1) (a) *Er will jemandem schmeicheln, aber*
He wants someone.dat flatter, but
sie wissen nicht wem.
they know not who.dat.
'He wants to flatter someone, but they don't know whom'

(b) *Er will jemanden loben, aber sie*
He wants someone.acc praise, but they
wissen nicht wen.
know not who.acc.
'He wants to praise someone, but they don't know who'

Other data strongly suggesting that elided constituents contain internal syntactic structure include filler-gap constructions, where the gap is contained in the elided constituent (2a). Such cases include island violations (2b).

(2) (a) John greeted every person who Bill did.
(b) * John greeted every person who Bill wondered why Sam did.

Facts such as these are difficult to account for in a purely semantic theory of ellipsis resolution, such as the one proposed in (Dalrymple et al., 1991). Given the strong evidence for the existence of syntactic structure within elided constitutents, the question arises of how to correctly infer the required syntactic structure, since this structure is not directly associated with overt phonological material.

In this paper, I will sketch an analysis of ellipsis employing the mechanisms of a description theory approach to grammar, such as in the D-Tree Grammar (DTG)(Vijay-Shanker, 1992) and D-Tree Substitution Grammar (DSG) (Rambow et al., 2001) formalisms. A description approach to grammar, in combination with a number of other assumptions, provides the right means for both declaratively characterizing the syntactic structure that exists within an elided constituent and for a pro-

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 204-210.

cedural interpretation that transparently leads to an incremental processing model.

2 Background

2.1 Semantic approaches to ellipsis

Existing semantic approaches to ellipsis resolution, such as the ones in (Dalrymple et al., 1991) and (Egg and Erk, 2002) are attractive because they provide a good account of a variety of semantic phenomena (including interactions with scope ambigiuites and anaphora), and they have both a declarative and and a procedual interpretation. However, these analyses have paid scant attention to the syntactic data mentioned above, and have failed to provide an adequate account of the syntax-semantics interface for sentences containing elided constituents. For example, in (Egg and Erk, 2002), the auxiliary verb left behind by VP ellipsis is treated as a kind of pro-verb, which "discharges" the ellipsis potential of an antecedent clause. The question remains as to why constituents containing these "pro-verbs" show evidence of having further underlying syntactic structure, as shown in (2).

The most influential of the semantic approaches to ellpsis has been the higher-order unification approach proposed in (Dalrymple et al., 1991). Here, the semantic representation of a clause containing an ellipsis (the "target" clause) contains a higher-order variable (3a), and this variable receives a value by solving an ellipsis equation (3b) involving the antecedent (or "source") clause.

(3) From (Dalrymple et al., 1991):

(a) $S \wedge R(T_1, ..., T_n)$
(b) $R(S_1, ..., S_n) = S$

In (3a), the elided utterance is represented as containing a free variable R, which is applied to the semantic values of the target elements which are parallel to a sequence of elements contained in the source utterance. Solving the equation in (3b) using higher-order unification causes R to become bound to a lambda expression, which can then be applied to the semantic values of the target parallel elements to construct a semantic representation for the target utterance as a whole. A simple example of how this works is shown in (4).

(4) (a) George listened to Beethoven's Ninth, and Sam did too.
(b) $listen(george, b9th) \wedge R(sam)$
(c) R(g) = listen(george, b9th)
(d) $\{R \rightarrow \lambda x.listen(x, b9th)\}$
(e) listen(sam, b9th)

In (4a) the parallel elements are 'George' and 'Sam', the semantic representation of source and target are shown in (4b), and the desired solution for the equation (4c) is given in (4d). Applying the lambda expression in (4d) to the semantic value of 'Sam' provides the (intuitively correct) meaning (4e) for the target containing the VP ellipsis.

While applying (3) to (4a) does derive the correct meaning, it provides no independent, compositionally determined representation of the semantics of the target clause containing the ellipsis, and no indication of the syntactic structure associated with the ellipsis clause. In (Gardent, 1999) and (Gardent, 2000) this defect is partially remedied. She demonstrates how replacing (3) with the equational setup in (5) both extends the empirical scope of the HOU account and provides a more principled account of the compositional semantics of elliptical sentences.

(5) From (Gardent, 1999):

(a) $S = C(X_1, ..., X_n)$
(b) $T = C(Y_1, ..., Y_n)$

In the approach of (Gardent, 1999), the semantics of the source and target sentences are derived from the normal compositional semantic construction process. An elided constituent is represented semantically with a free variable of the proper type, i.e. the type it would normally receive based on its syntactic category. This differentiates her analysis from the one in (Dalrymple et al., 1991). Futhermore, there are two equations (5a,b) rather than one (3b), which together introduce a free variable C representing the common background relation shared by the source and target clauses. Resolving these equations causes the free variable introduced by ellipsis to be resolved as a side effect. Thus, ellipsis resolution is driven by the general process of establishing a redundancy relation between two clauses in a discourse. How this works for sentence (4a) is shown in (6).

(6) (a) $listen(george, b9th) \wedge R(sam)$
(b) $C(george) = listen(george, b9th)$
(c) $C(sam) = R(sam)$
(d) $\{C \rightarrow \lambda x.listen(x, b9th), R \rightarrow \lambda x.listen(x, b9th)\}$

For this particular example, the approaches of (Dalrymple et al., 1991) and (Gardent, 1999) obtain the same result. However, as explained in (Gardent, 1999) and (Gardent, 2000) the equational setup in (5) not only provides a clearer picture of the syntax-semantics interface, it also opens the door to using the HOU analysis to explain other phenomena, such as focus, deaccenting, and strict/sloppy readings in both ellipsis and non-ellipsis contexts.

While (5) improves on (3), it still leaves open the question of how to syntactically represent elided constituents

which appear to contain internal syntactic structure, as shown in (1) and (2) above. In what follows I will adopt the equations in (5) as a semantic condition on ellipsis representations. However, I will augment it with a syntactic condition, which will be used to generate syntactic structure within elided constituents, and I will show how the syntactic and semantic conditions can be made to work together.

2.2 Syntactic approaches to ellipsis

Syntactic approaches to ellipsis start with the idea that elided constituents contain full syntactic structure at some level of a syntactic representation. They differ on which level of representation the syntactic structure is present. For example, in (Lobeck, 1995) and (Chung et al., 1995) an elided constituent is initially syntactically null, and receives syntactic structure by copying it from the antecedent clause at the level of LF. However, such "copying" approaches to ellipsis run into some of the same problems as semantic approaches to ellipsis, since special mechanisms need to be posited in order to account for the connectivity effects shown in (1) and (2). For example, (Chung et al., 1995) posit three separate special ellipsis operations ("recycling", "merger", and "sprouting") for this reason [1].

The other tack taken by syntactic approaches is to posit full syntactic structure at initial levels of syntactic representation, generated in the normal fashion (Ross, 1969) (Merchant, 2001). The difference between elided and non-elided constituents is that elided constituents have no overt PF material associated with them. Such "PF-deletion" approaches have the advantage that they straightforwardly account for the connectivity effects shown in (1) and (2).

The most successful and extended defence of the PF-deletion approach to ellipsis is given by (Merchant, 2001). In his analysis, PF-deletion is triggered by a syntactic feature on heads, labelled "E". When a head contains this feature, it instructs the PF component of syntax to ignore its complement, i.e., the complement receives no PF interpretation. Futhermore, the E-feature is associated with a semantic identification condition. This condition in essence states that the focus semantic values of the antecedent and ellipsis constituents must be semantically equivalent [2] (cf. (Rooth, 1992)). By stating this condition in terms of semantic entailment, (Merchant, 2001) avoids some of the problems associated with syntactic approaches that require syntactic isomorphism between source and target sentences (e.g., (Fiengo and May, 1994)), as illustrated in (7) and (8) (from (Merchant, 2001)).

(7) (a) Abby was reading, but I don't know what.
(b) Ben called – guess when!

(8) (a) They arrested Alex$_i$, though he$_i$ thought they wouldn't
(b) They arrested [the guy who lives over the garage]$_i$, though he$_i$ thought they wouldn't.
(c) *He$_i$ thought they wouldn't arrest Alex$_i$.
(d) *He$_i$ thought they wouldn't arrest [the guy who lives over the garage]$_i$.

Examples (7a) and (7b) show that a wh-trace can appear in an elided constituent even when no corresponding syntactic argument appears in the antecedent. The examples in (8) show that Condition-C violations do not occur in ellipsis clauses, counter to what would be expected were the elided constituent to be completely isomorphic to its antecedent.

Due to the success of this PF-deletion theory of ellipsis, and in particular its ability to account for both connectivity effects as well as the kind of syntactic"mismatches" shown in (7) and (8), I adopt much of its outline. One significant drawback, however, is that the theory (as stated) does not lead transparently to a processing model for ellipsis, unlike the semantic approaches outlined above. In particular, the formulation of the semantic identification condition given in (Merchant, 2001) leaves it unclear as to how a processor is to generate the requisite ellipsis-internal syntactic structure. The goal of the rest of this paper is to combine the demonstrated empirical advantages of the PF-deletion approach with the formal advantages of the HOU approach.

3 Analysis

Here is a brief outline of the phonological, syntactic, and semantic components of my analysis. First, (descriptions of) elementary trees are anchored by lexical items which consist of bundles of syntactic and semantic features, but not of phonological features. Phonological features are added independently by a set of "spell-out rules" (cf. (Halle and Marantz, 1993)(Ackema and Neeleman, 2003)), which map from bracketed sequences of syntactic feature bundles to sequences of bundles of phonological features. Second, loosely following (Merchant, 2001), certain syntactic categories (in English, NP, VP, and IP) may optionally bear an E-feature, which triggers spell-out rules that generate the empty string. Third, again following (Merchant, 2001), I will associate the E-feature with a semantic constraint on the content of the elided constituent. Unlike (Merchant, 2001), however, the semantic constraint is formulated in terms of the equations

[1] Some empirical problems with the (Chung et al., 1995) approach are detailed in (Merchant, 2001)

[2] This is a simplification of the actual statement of this condition

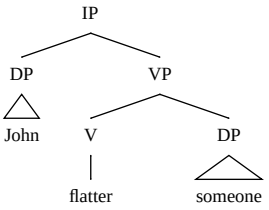
in (5). Additionally, I associate a *syntactic* identity condition with the E-feature, which regulates the syntactic structures appearing within elided constituents. [3] Combining the semantic and the syntactic conditions has the advantage that they together provide a declarative characterization of the syntax and semantics of ellipsis. It is also straightforward to assign these conditions a procedural interpretation. In the rest of this paper, I further elaborate the three components of my analysis and conclude with some open problems that require further research.

3.1 Syntax of Ellipsis

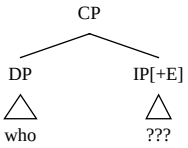
Here is a simple sluicing example:

(9) (a) John flattered someone, but I don't know who.

(b) source:



(c) target:



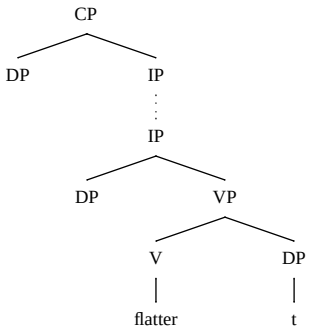
Of particular interest here is the tree in (9c), representing a (partial) syntactic analysis of the target ellipsis clause. Note that the IP node in this tree bears the E-feature, which means that its PF content is absent. However, by hypothesis, the underlying syntactic structure is not. It is for this reason that the question marks appear under IP. The question here is how to generate the required syntactic structure, while avoiding some of the problems associated with approaches that require syntactic isomorphism with the source clause (Rooth, 1992) (Fiengo and May, 1994).

Imagine that (9c) is output by a parser, and represents a description of what the parser has seen so far. Assuming a lexicalized grammar (Joshi and Schabes, 1991), the DP slot into which the wh-phrase has been substituted must be part of an elementary tree anchored by some (missing) lexical item. I propose that we add this missing information to the tree description in (9c) through an application of the constraint in (10). [4]

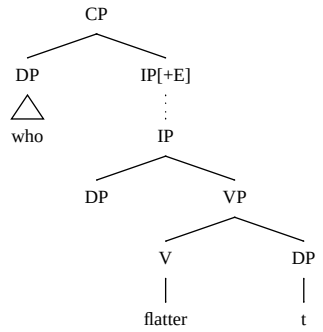
(10) Let $par(T)$ be the set of constituents in the target ellipsis clause that are parallel to a set of constituents $par(S)$ in the source clause. For each t $\in par(T)$ and matching s $\in par(S)$, $\alpha^{-}(root(t))$ = $\alpha^{-}(root(s))$

In (9c) assume that the DP dominating 'who' in the target is matched with the DP dominating 'someone' in the source. The negative lexical anchor of 'someone' is 'flatter'. Therefore, to satisfy constraint (10) in (9c), we must select an elementary tree from the lexicon that is anchored by 'flatter', and which is compatible with the existing description of the target ellipsis clause in (9c). The elementary tree in (11a) fits the bill. [5] We add the description representing this elementary tree to the desription of the target clause already generated by the parser, resulting in (11b), which contains syntactic structure derived within the ellipsis site.

(11) (a)



(b)



It is important to note that the syntax inside the ellipsis in (11) is incomplete, in the technical sense defined in (Rambow et al., 2001). That is, there are frontier nodes that are labelled with non-terminal symbols (i.e., substitution nodes). The constraint in (10) says nothing about these nodes. There are a couple of ways to approach this issue. First, one might posit another syntactic constraint which "fills in" these empty argument positions with pronouns of the appropriate sort. This would be play a role similar to that of the "Vehicle Change" analysis proposed in (Fiengo and May, 1994). Another way to approach this

[3] This syntactic identity condition has the effect of constraining the possible meanings that can be associated with the elided constituent. These syntactic constraints therefore play a role similar to the one played by "syntactic presuppostions" in (Ginzburg, 1999).

[4] In (10), I borrow some terminology from (Muskens, 2001). The *negative lexical anchor* of the root node r of a tree τ_1 (denoted by $\alpha^{-}(root(\tau_1))$) is the lexical anchor of the substitution node n of tree τ_2 into which tree τ_1 is substitued.

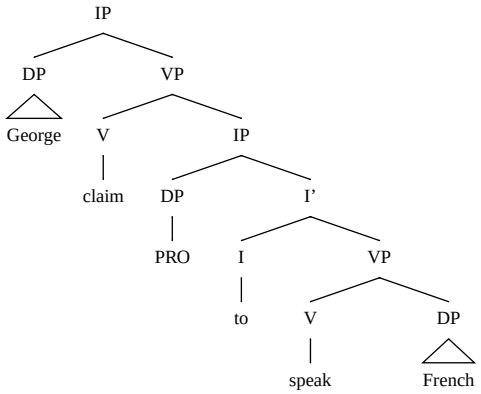
[5] Here I am using the standard notation where solid lines indicate immediate dominance, and dotted lines indicate (possibly non-immediate) dominance.

issue is to allow the positions to remain unfilled. Normally, it is forbidden for a non-terminal to remain on the fringe of a derived tree at the end of a derivation. Linguistically, this can be viewed as a consequence of PF requirements and of the requirement of full interpretation of syntactic structure. However, in the case of ellipsis, PF requirements are fulfilled by mapping the elided structure to an empty string, and the structure containing the ellipsis receives full interpretation by solving the ellipsis equations (as described below). Hence the need to fill the argument slots with lexical material disappears.

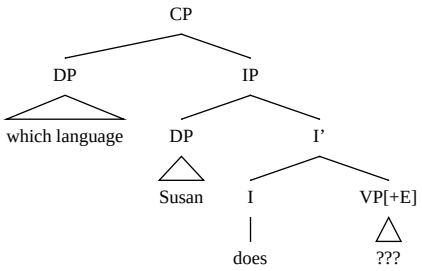
The current approach can also handle examples of VP ellipsis. This includes VP ellipsis from which a wh-phrase has been extracted, as shown in (12).

(12) (a) George claims to speak French, but I don't know which language Susan does.

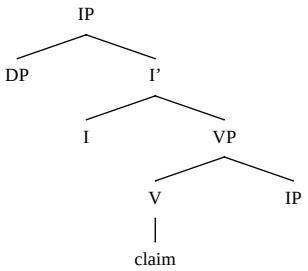
(b) source:



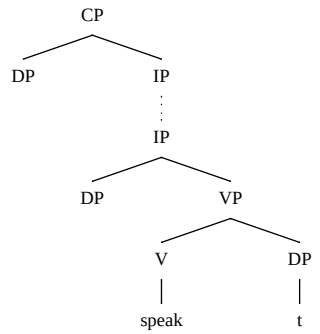
(c) target:



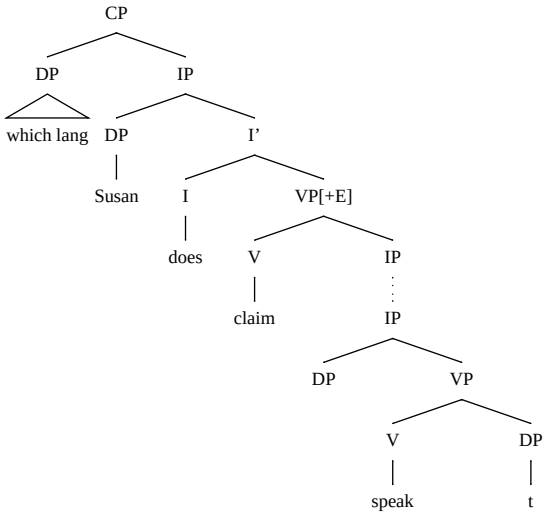
(d)



(e)



(f)

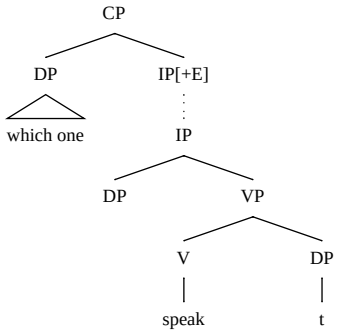


In (12), the target contains multiple ellipsis remnants ('which language' and 'Susan'), which match parallel elements in the source clause ('French' and 'George'). Employing condition (10) allows us to add the elementary tree descriptions in (12d) and (12e) to the desription shown in (12c). The resulting tree description is shown in (12f).

Another case of sluicing is shown in (13) (this time, only the syntactic structure of the target clause is shown, after applying (10)).

(13) (a) George claims to speak an exotic language, but I don't know which one.

(b) target:



This example is notable because the matrix verb in the source clause is entirely unrepresented in the target, un-

like the previous two examples. The issue of how to assign this ellipsis the correct semantics despite this fact is addressed in the next section.

3.2 Semantics of Ellipsis

As noted above, I adopt (5) from (Gardent, 1999) as the semantic condition on ellipsis. More accurately, (5) is to be viewed as a general redundancy condition on two clauses entering into a discourse relation, and when applied to a clause containing an ellipsis, the ellipsis is resolved as a side effect. How this works for (9) is shown in (14).

(14) (a) $flatter(j, f(person)) = C(f(person))$
(b) $flatter(x, g(person)) = C(g(person))$
(c) $\{C \rightarrow \lambda y.flatter(j, y), x \rightarrow j\}$

The left hand sides of the equations in (14a) and (14b) contain (simplified) semantic representations of the source and target clauses, respectively. [6] Note that the fringe non-terminal DP in (11b) is represented in the (14b) as a free variable. This variable is resolved by solving the equations, as shown in (14c)

The resolution of (12) proceeds in a similar fashion, as shown in (15).

(15) (a) $claim(g, speak(g, fr)) = C(g, fr)$
(b) $claim(s, speak(x, f(lang))) = C(s, f(lang))$
(c) $\{C \rightarrow \lambda y \lambda z.claim(y, speak(y, z)), x \rightarrow s\}$

Once again, the fringe non-terminal DP in (12f) is translated as a free variable, which gets resolved by solving the equations in (15c).

More challenging is (13). Here we must say something more in order to derive the correct semantics, due to the fact that the matrix verb of the source is absent from the syntactic representation of the target. For this case, it is possible to take advantage of the fact that the two IPs in the target syntactic representation are related to each other by a dominance relation (as indicated by the dotted line), rather than an immediate dominance relation. For cases such as these, (Pinkal, 1996) proposes the constraint in (16).

(16) For each pair of nodes P_i and P_j for which the dominance relation is stated, we add the constraint $X_i = C(X_j)$. [where X_i is the semantic value of P_i, X_j is the semantic value of P_j, and C is a free higher-order variable]

Applying (16) to (13b) allows us to derive the semantic representation on the left hand side of the equation in (17b). Solving the equations then proceeds as normal in (17c)

(17) (a) $claim(g, speak(g, f(lang)) = C(f(lang)))$
(b) $R(speak(x, h(lang))) = C(h(lang))$
(c) $\{C \rightarrow \lambda y.claim(g, speak(g, y)), x \rightarrow g, R \rightarrow \lambda P.claim(x, P)\}$

Of course the analysis of (13b) goes through only if we do not conflate the two IPs in this representation, as would occur for example if we applied the "reading off" algorithm in (Rambow et al., 2001). There are both technical and conceptual issues here that remain to be taken care of.

3.3 Phonology of Ellipsis

Some mechanism is required in order to prevent the phonological realization of lexical items contained in elided constituents. If elided constituents contain syntactic structure, and syntactic structures are anchored to lexical items, then an elided constituent contains at least one lexical item, as for example in (11b). One possible means for ensuring that such lexical items remain unpronounced is to assume that they do not contain phonological material at the relevant level of representation. Instead, lexical items consist of sets of syntactic and semantic features only. It will then be necessary to formulate a theory for how these features are "spelled-out" (cf. (Halle and Marantz, 1993)(Ackema and Neeleman, 2003)). The theory might consist of a set of mapping rules from syntactic representations to phonological representations. These mapping rules could be made sensitive to the presence or absence of the hypothesized E-feature. A schematic formulation of an "ellipsis" spell-out rule is shown in (18). This mapping schema simply states that an XP bearing the E-feature is mapped to the empty string at PF.

(18) $[\text{XP}_{+E}] \mapsto \epsilon$ (where XPϵ{NP,VP,IP})

For now this remains merely a rough sketch of how an account of the phonology of ellipsis could be made to work. Clearly much more needs to be said to make this component of the analysis precise.

4 Conclusion

In this paper, I have sketched an analysis of ellipsis which combines ideas from PF deletion theories with ideas from semantic theories. This combination of ideas is enabled by adopting a description approach to grammar, as in the formalisms proposed in (Vijay-Shanker, 1992), (Rambow et al., 2001) and (Muskens, 2001). This has resulted in an approach that can handle some cases of ellipsis where there is evidence for syntactic structure within the elided constituent. The approach also lends itself to a straightforward procedural interpretation, making it possible to develop an expicit processing algorithm, once it has been made sufficiently explicit.

[6] The parallel elements are represented as choice functions.

However, many technical and conceptual issues remain to be resolved, such as how it is possible to licence incomplete tree descriptions under ellipsis. Empirical issues remain as well. Here, I discuss only one example. The evidence for syntactic structure within ellipsis is more complicated than has been suggested above. It turns out that IP ellipsis can repair island violations (Ross, 1969)(Merchant, 2001), while VP ellipsis does not. This is shown in (19) and (20) (from (Merchant, to appear)).

(19) (a) They want to hire someone who speaks a Balkan language, but I don't remember which.

(b) * I don't remember which (Balkan language) they want to hire someone who speaks.

(c) Bob ate dinner and saw a movie that night, but he didn't say which.

(d) * He didn't say which movie he ate dinner and saw that night.

(20) (a) * Abbey does want to hire someone who speaks Greek, but I don't remember what kind of language she doesn't.

(b) * They got the president and 37 Democratic Senators to agree to revise the budget, but I can't remember how many Republican ones they didn't.

The challenge here is to generate sufficient syntactic structure for the VP ellipsis cases in (20) to account for the island violations, yet in a manner that doesn't generate island violations for the IP ellipsis cases in (19). Accounting for this puzzle provides a challenge for any unified theory of ellipsis constructions, including the present one. Doing so remains a goal for future research.

References

Peter Ackema and Ad Neeleman. 2003. Context-sensitive spell-out. *Natural Language and Linguistic Theory*, 21:681–735.

Sandra Chung, William Ladusaw, and James McCloskey. 1995. Sluicing and logical form. *Natural Language Semantics*, 3:239–282.

Mary Dalrymple, Stuart Shieber, and Fernando Pereira. 1991. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452.

Markus Egg and Katrin Erk. 2002. A compositional account of vp ellipsis. In Frank Van Eynde, Lars Hellan, and Dorothee Beermann, editors, *Proceedings of the 8th International Conference on Head-Driven Phrase Structure Grammar*, pages 162–179. CSLI Publications.

Robert Fiengo and Robert May. 1994. *Indices and Identity*. MIT Press.

Claire Gardent. 1999. Unifying parallels. In *Proceedings of the 27th Annual Meeting of the ACL*, pages 188–207.

Claire Gardent. 2000. Deaccenting and higher-order unification. *Journal of Logic Language and Information*, 9(3):313–338.

Jonathan Ginzburg. 1999. Semantically-based ellipsis resolution with syntactic presuppositions. In Harry Bunt and Reinhard Muskens, editors, *Computing Meaning*, volume 1, pages 255–279. Kluwer Academic Publishers.

Morris Halle and Alec Marantz. 1993. Distributed morphology and the pieces of inflection. In K. Hale and S.J. Keyser, editors, *The View from Building 20*, pages 111–176. MIT Press.

Jorge Hankamer and Ivan Sag. 1976. Deep and surface anaphors. *Linguistic Inquiry*, pages 391–428.

Aravind Joshi and Yves Schabes. 1991. Tree-adjoining grammars and lexicalized grammars. In *Definability and Recognizability of Sets of Trees*. Elsevier.

Chris Kennedy. 2003. Ellipsis and syntactic representation. In Kerstin Schwabe and Susanne Winkler, editors, *The Interfaces: Deriving and Interpreting Omitted Structures*, pages 29–53. John Benjamins.

Anne Lobeck. 1995. *Ellipsis: Functional heads, licensing, and identification*. Oxford University Press.

Jason Merchant. 2001. *The Syntax of Silence: Sluicing, Islands, and the Theory of Ellipsis*. Oxford University Press.

Jason Merchant. to appear. Variable island repair. In Kyle Johnson, editor, *Topics in Ellipsis*. Cambridge University Press.

Reinhard Muskens. 2001. Talking about trees and truth-conditions. *Journal of Logic, Language, and Information*, 10(4):417–455.

Manfred Pinkal. 1996. Radical underspecification. In *Proceedings of the 10th Amsterdam Colloquium*, pages 587–606.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-tree substitution grammars. *Computational Linguistics*, 27(1):87–121.

Mats Rooth. 1992. Ellipsis redundancy and reduction redundancy. In Steve Berman and Arild Hestvik, editors, *Proceedings of the Stuttgarter Ellipsis Workshop*, number 29 in Arbeitspapiere des Sonderforschungsbereichs 340.

John Ross. 1969. Guess who? In R. Binnick, A. Davison, G. Green, and J. Morgan, editors, *Papers from the 5th Regional Meeting of the Chicago Linguistics Society*, pages 252–86.

K. Vijay-Shanker. 1992. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–518.

Using a Meta-Grammar for LTAG Korean Grammar

SinWon Yoon
UFRL, University Paris 7
2 place Jussieu, Case 7003, 75251 Paris Cedex 05
swyoon@linguist.jussieu.fr

Abstract

Generating elementary trees for wide-coverage Lexicalized Tree Adjoining Grammars (LTAG) is one of the great concerns in the TAG project. We know that the Korean LTAG developed in (Han C.-H. et al., 2000) was not sufficient to handle various syntactic structures. Therefore, a Korean Meta-Grammar (KMG) is proposed to generate and maintain a large number of elementary tree schemata. Describing Korean MG with more precise tree families and with class encoding Korean syntactic properties leads to a larger coverage capacity for Korean LTAG.

1 Motivations for this work

The first development of LTAG Korean Grammar (KTAG) was proposed in (Han C.-H. et al., 2000). Few grammars for Korean exist, the one for TAG is quite small with limited coverage. Our goal is to generate a wider-coverage KTAG, using a now well-established grammar development technique. We propose using the Meta-Grammar for KTAG :

a) The MG was successfully used to generate wide-coverage grammars for French and medium size TAG for Italian (Candito 1996; Candito 1999), within the FTAG project at the Univ. of Paris 7 (Abeillé, 2002). So the use of the MG to generate real-size grammars has already been established [1] .

b) In addition, the MG was also used to generate wide-coverage grammars for frameworks like LFG (Clement and Kinyon, 2003). This stronly suggests that the MG is more portable to non-TAG frameworks, unlike other approaches such as Metarules

c) The MG was also used to generate test-suite sentences for German (Kinyon and Rambow, 2003), as well as a medium-size grammar for German (Gerdes, 2002). This specific use of the MG for text-generation shows over-generation is not a real issue. i.e., not more than for any standard grammar development technique[2]. Moreover, the MG is particularly appropriate to handle relatively "free-word order" languages such as Korean and German, because of underspecification. This mechanism is used for handling phenomena such as scrambling.

2 What is a Meta Grammar

The notion of Meta Grammar (MG) was originally presented to automatically generate wide-coverage TAGs for French and Italian, using a hierarchical-level and compact layer of linguistic description which imposes a general organization for the syntactic information, shared by the different elementary tree families, in a three dimensional inheritance network. The elementary structures of a MG are the classes organized in the Inheritance Graph. The classes in a graph order from more general classes to more specific classes, e.g., the class TRANSITIVE-VERB inherits information from one general class VERB. The three dimensional hierarchies in a MG represent the following information (Candito, 1999):

- In Dimension 1, each terminal class encodes an initial sub-categorization, i.e., a list of arguments associated with a given head with an initial syntactic function for each, e.g., a subject and an object for a transitive verbal anchor.
- In Dimension 2, each terminal class encodes a list of final function, i.e., a possible change in the initial grammatical function from dimension 1, including the possibility to increase or decrease the number of syntactic functions to be realized, e.g., adding an

[1] For French, the MG is also used for the syntax of nouns and adjectives (see (Barrier and Barrier, 2003);(Barrier and Barrier, 2004))

[2] Even the 5000 tree FTAG was successfully used in the G-TAG text generation project

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 211-218.

argument for the causative, and erasing an argument for passive with no agent.

- In Dimension 3, each terminal class encodes the surface realization of a final syntactic function. The category and the word order are selected.

Each class in the hierarchy is associated with a partial description of a tree. These partial descriptions of trees, called *quasi-trees*, encode *father, dominance, equality* and *precedence* relations between tree nodes. A well-formed tree is generated by inheriting information from exactly one terminal class from dimension 1, one terminal class from dimension 2, and *n* terminal classes from dimension 3. For instance, in order to generate the elementary tree for *By whom will Mary be accompanied ?*, a MG compiler creates one crossing class which is inherited from a *strict-transitive* class in dimension 1, from a *personal-full-passive* class in dimension 2, and from a *Wh-questioned-By-complement* class in dimension 3.

3 Hierarchical Descriptions in Korean Meta-Grammar (KMG) for LTAG

The Korean LTAG (Han C.-H. et al., 2000) consists of 15 tree families (see Fig.(1)). The 289 elementary trees

	Tree Families
8 for Verbs	Tnx0V, Tnx0nx1V, Tnx0nxp1V, Tnx0nxp1nx2V Tnx0s1V, Tnx0nxp1s2V, Tnx0nxNOM1V, Tnx0nx1CO
3 for Adjectives	Tnx0A, Tnx0nxp1A, Tnx0nxNOM1A
4 for Structures	Declarative and Relative Constructions, Gerund and Adverbial Clauses

Figure 1: Tree Families in (Han C.-H. et al., 2000)

have been created. Han C.-H. et al, 2000 said that it was expected to increase the number of elementary trees in order to handle more syntactic phenomena : passive, causative, resultative, light verb construction, coordination construction, and scrambling. In particular, the most important concern about the coverage capacity for a Korean grammar is the ability to handle the scrambling phenomenon. Because free-word order probably leads to an enormous expansion in the number of elementary trees due to permutations of arguments.

3.1 Initial Syntactic Functions in KMG

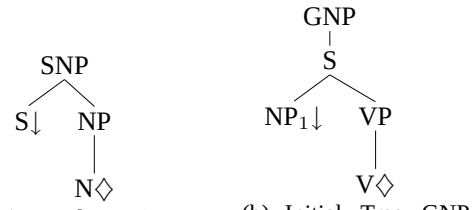
Lexicalized TAG elementary trees represent extended projections of lexical items and encapsulate all syntactic arguments of a lexical anchor. We describe the initial subcategorization frames for Korean verbs, which will be encoded in each elementary tree. Tree families proposed here cover those of (Han C.-H. et al., 2000).

Before representing initial subcategorization frames, we explain the linguistic choice for KMG : As defective verbs, auxiliary verbs, causative and/or passive auxiliary verbs, raising verbs are not represented by sentential structures, i.e., they have a reduced projection to VP and not to S. We use the syntactic category SNP (sentential noun phrase) for the complex noun phrase, and the syntactic category GNP (gerund noun phrase) for the gerund construction. When sentential clauses appear in an argument position, they become either like complex noun phrases as in (1), or like gerund noun phrases as in (2). Head items in SNP and GNP take a case marker such as a lexical head noun in NP[3]. SNP and GNP behave as nouns as a whole. But in contrast to NP nodes, modifiers for nouns can not adjoin at a SNP (complex NP) or a GNP (gerund NP) node. We have specified SNP↓ and GNP↓ nodes in tree families of predicates for which subcategorize.

Complex NPs are represented by an initial tree, whose root node is SNP. It is anchored by the head dependant noun and it has a substitution node S for the clause that modifies the head noun in Fig. 2(a). Gerund NPs are represented by an initial tree, whose root node is GNP, that is anchored by the head verb that represents appropriate subcategorization frames in Fig. 2(b).

(1) Minho-ga $[_{snp}$ yaksok-e neujossda-n-sasil-eul]
Minho$_{nom}$ appointment$_{pp}$ be.late$_{adn.FACT.acc}$
arassda.
realize
'Minho realizes that he is late for the appointment'

(2) Minho-ga $[_{gnp}$ sakwa-reul meok-gi-reul]
Minho$_{nom}$ apple$_{acc}$ eat$_{nominalizer.acc}$
silheohanda.
dislike
'Minho does not like to eat apples.'



(a) Initial Tree SNP (b) Initial Tree GNP (intransitive)

Figure 2: Trees that anchor complex and gerund NPs

We have defined 26 tree families for verbs (see Tab. (5)), and 9 tree families for the adjectives are defined (see Tab.(5)). We know that, in Korean, the syntactic functions of an argument are assigned by the markers. I.e., an

[3]i.e., a head noun in a complex NP and a head verb in gerund NP inflected with a nominalizer marker (*-gi/-eum*) are inflected with one of the case markers.

argument function will be fixed depending on the marker taken by an argument. For example, in order to realize the bare noun *Minho* as a subject in a sentence, this bare noun 'Minho' is marked with the nominative case *-i/-ga*, e.g., *Minho-ga* as a subject. According to our subcategorization frames, we proposed 10 initial syntactic functions to realize each constituent in its phrase-structure in the surface realization dimension.

1. *nom1* function for the subject argument
2. *nom2* function for the second subject argument in the double nominative construction, which can not be permuted, and for the causee with nominative in the causative
3. *acc1* function for the object argument
4. *acc2* function for the second object argument in the double accusative construction, which can not be permuted and can not be promoted to a subject position in a passive
5. *dat* function for the dative argument
6. *obli1* function for the obligatory argument marked by a postposition
7. *obli2* function for the facultative argument marked by a postposition
8. *adv* function for the adverbial argument
9. *loc* function for the locative argument of movement verbs
10. *noun-scomp* function for the complex noun phrase and for the gerund.

3.2 Redistribution of Functions in KMG

The initial functions for a predicate can be changed by a series of redistribution of functions. The passive and causative constructions determine the redistribution of grammatical functions (Suh J. -S., 1996; Lee S. -O., 1999).

Passive : Korean has two passive types. -i) A predicate is inflected with one of passive morphemes *i, hi, ri, gi*. This is the morphological passive as in (3). -ii) A predicate is realized with a passive auxiliary verb *-eo jida, -ge doeta* as in (4). This is the analytical passive.

(3) a. saengjwi-ga goyangi-reul mureossda.
mouse$_{nom}$ cat$_{acc}$ bit
'A mouse bit a cat.'

b. goyangi-ga saengjwi-ege mur*i*eossda.
cat$_{nom}$ mouse$_{pp}$ be.bitten
'A cat was bitten by a mouse.'

(4) a. saramdeul-i geu norae-reul pureunda.
people$_{nom}$ that song$_{acc}$ sing
'The people sing this song.'

b. geu norae-ga saramdeul-e uihae bur-*yeo jinda*.
that sing$_{nom}$ people$_{pp}$ is.sung
'This song is sung by the people.'

Two tree schemata are proposed for the Korean passive construction in Fig.(3). The tree schema (Fig.3(a)) represents the structure for the morphological passive. This elementary tree is anchored by a morphological passive verb. The tree schema (Fig.3(b)) represents the structure for the syntactical passive which contains two verbal anchor nodes : the one is for a main verb, and the other is for a passive auxiliary verb. The subject in the active becomes the agent marked with a postposition in the passive. An additional `agent` function is used for agent. The initial `nom1` function for subject changes into an additional `agent` function. Concerning the subject in the passive, in Korean, not only the accusative argument, but the argument marked with an other postposition can be promoted to a subject position, e.g., the *Minho-egeseo*, the NP marked by a postposition is promoted to the subject position in the passive (5b). So an additional `patient` function waits for `acc1` or `obli` functions, whose arguments take the PATIENT feature in the thematique relation with respect to its predicate.

(5) a. keu namja-ga Minho-egeseo jigab-eul
that man$_{nom}$ Minho$_{pp}$ purse$_{acc}$
ppaesassada.
stole
'That man stole a purse from Minho.'

b. Minho-ga keu namja-hante jigab-eul
Miho$_{nom}$ that man$_{pp}$ purse$_{acc}$
ppaesass*gi*eossda.
be.stolen
'Minho had his purse stolen by that man'

In KMG, in order to represent the morphological passive sentence as in (5b), in redistribution dimension, the *Morph.nom1-agent-patient-nom1* terminal class is used. This class inherits information from the `nom1` → `agent` class for the demotion of the subject, from the `patient` → `nom1` class for the promotion to subject, and from the *Morph.* class for morphological passive type. In the surface realization dimension, the sentence (5b) is generated with the tree schema (Fig. 3(a)).

Causative: Likewise in the passive construction, Korean has two different causative forms[4]. -i) One is a morphological causative: a predicate is inflected by one of morphemes *i, hi, ri, gi, u, gu, chu* as in (7), -ii) The other type is a syntactical causative: the main verb is followed by a causative auxiliary verb *-ge(-dorok) hada*,

[4]Examples ((7) and (8)) are causative sentences of (6).

(a) Morphological Passive (b) Analytical Passive

Figure 3: Tree Schemata for Passive

manteulda, sikida as in (8). In causative construction, the causer is a new argument. The `nom1` function is assigned to this new argument for the subject realization in causatives. Concerning the causee, it can be marked by various markers, e.g., causees with dative in (7(a) and 8(a)), the causee with nominative in (8b). There are constraints for causee realizations in Korean. For example, the nominative causee never appears in the morphological causative such as in (7b), while it can be accepted in the syntactical causative such as in (8b). Because the analytic causative construction contains two verbs (the matrix verb and the auxiliary verb), except for the subject of the causative auxiliary verb, we can also expect that another subject in the sentence will appear in addition to the matrix verb. That is why the nominative marker can be accepted.

(6) Sumi-ga yak-eul meokneunda.
Sumi$_{nom}$ medicine$_{acc}$ takes
'Sumi takes medicine.'

(7) a. Minho-ga Sumi-ege yak-eul mog*i*nda.
Minho$_{nom}$ Sumi$_{dat}$ medicine$_{acc}$ take$_{Mcau}$
'Minho makes Sumi take medicine.'

*b. Minho-ga Sumi-ga(Sumi$_{nom}$) yak-eul mog*i*nda.

(8) a. Minho-ga Sumi-ege yak-eul mok-ge handa.
Minho$_{nom}$ Sumi$_{dat}$ medicine$_{acc}$ take$_{Aux.cau}$

b. Minho-ga Sumi-ga(Sumi$_{nom}$) yak-eul mok-ge handa.

In causatives, the sentence meaning is changed according to causee markers[5] We can consider that the function for causee depends on the transitivity of the embedded verb, and on the causative form. We propose the following constraints for the causee :

	Intransitivity	**Transitivity**
Suffix	Acc1	Acc1, Dat [animate]
Aux	Nom1, Acc1, Dat [animate]	Nom1, Acc1, Dat [animate],Obli

Table 1: Functions of causee in Korean Causative

According to the redistribution of the initial subject function (`nom1`), we have various terminal classes for the causative. Tree schemata are proposed for the Korean causative construction in Fig.(4) : The monoclausal structure is recommended for morphological causatives, in which the nominative causee is not accepted, and in which the rest except the causer forms one constituent. Bi-clausal structures are recommended for complex causatives, in which the causee can be represented with the nominative or with other markers, and in which the sentence meaning is changed according to the markers of the causee.

3.3 Surface Realization in KMG

For syntactic realizations, three general classes are used : `non-realization` is used for empty constituents, and the `pre-verbal` is used for canonical position realizations. It can also cover the questioned element realization because Korean does not have a *wh-* movement, and the `post-verbal` is used for constituents of cleft, relative construction and extraposition.

`non-realization`: Korean freely allows empty arguments as in (10). In order to represent empty arguments, elementary trees whose argument NPs are associated with ϵ are used.

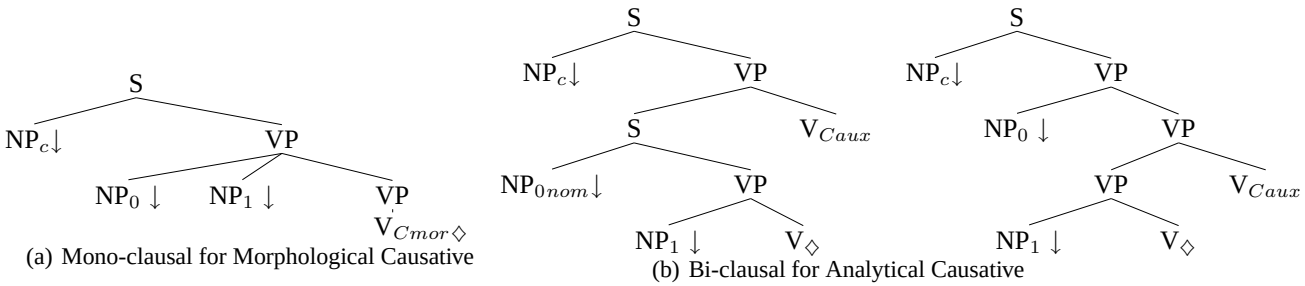
(10) ϵ ϵ ilkeossda.
ϵ ϵ read
'(I/you/he/we/they) read (it/them)

`pre-verbal` : This class is used for argument appearances before their predicates that have a syntactical dependance in a clause. For example, when a normal subject argument is realized, a nominal argument appears in

[5]For a more detailed explanation about the relationship between the sentence meaning and causee markers, see (Yoon S.-W., 2003). For instance, the causation with the nominative causee such as (9a) is permissive, whereas the causation with the accusative causee such as (9b) is coercive.

(9) a. Minho-ga Sumi-ga ga-dorok haessda.
Minho$_{nom}$ Sumi$_{nom}$ go$_{Aux.cau}$
Minho made Sumi go

b. Minho-ga Sumi-reul(Sumi$_{acc}$) ga-dorok haessda.



(a) Mono-clausal for Morphological Causative

(b) Bi-clausal for Analytical Causative

Figure 4: Tree Schemata for Causative

front of the verb with a nominative marker *-ga/ -i* as in (11a), and it permutes with other constituents in a clause, i.e., local scrambling as in (11b). By under-specifying realization positions with `pre-verbal`, elementary trees for permutations of arguments before a predicate are automatically generated. The local scrambling can be handled with the KMG.

(11) a. *Minho-ga* chaek-reul ilkeossda.
Minho$_{nom}$ chaek$_{acc}$ read
'Minho read a book.'

b. chaek-reul *Minho-ga* ilkeossda.

It will be the same for pronoun subjects*na (I), neo (you), geunyeo (she), geu (he), uri (we), neohui (you)*, and the interrogative pronouns subjects *(nu(gu): who, mueos: what)*. Therefore, we use the same tree schemata for a verb with a simple subject, for a verb with an interrogative subject, and for a verb with a pronominal subject.

Concerning a direct object realization and other oblique object realizations, whether they are nominal, pronoun, interrogative pronoun or sentential, they are identical to the realization of the subject.

`post-verbal`: This class is used for argument appearances after their predicates that have a syntactical dependance in a clause, i.e., *Extraposition* or *Korean inversion*. Not only nominal elements as in (12a), but sentential elements can also be extraposed as in (12b). Likewise for local scrambling, by under-specifying realization positions with the `post-verbal` class, we obtain elementary trees for extraposed elements - even elementary trees for the permutation among extraposed elements.

(12) a. Minho-ga *t* johahanda, [Sumi-reul]$_t$
Minho$_{nom}$ likes Sumi$_{acc}$
'Minho likes Sumi.'

b. John-i *t* malhaessda,
John$_{nom}$ *t* said
Minho-ga Sumi-reul johahan-dago.
[Minho$_{nom}$ Sumi$_{acc}$ like$_{comp}$]$_t$
'John said that Minho liked Sumi.'

We also use this `post-verbal` class for realizations of relative constructions. Relative clauses are NP modifiers in which an argument position is empty. For instance, a subject argument position is empty in the relative clause of (14). We can consider that there is an empty element after the main verb, which corresponds to this empty-subject argument, and which is syntactically related to the relative clause, e.g., *saram(person)* in (14). We call this a relativized-subject. By adjoining an auxiliary tree representing the relativized-subject to a NP, the NP is modified by a relative with an empty-subject[6]. In KMG, the relative modification comes about through the `post-verbal` class.

(14) [$_s$ Sumi-hanteseo satang-eul eot-eun] (saram)
Sumi$_{pp}$ candy$_{acc}$ get$_{rel}$ (person)
'(person) [who gets a candy from Sumi.]'

In the same way, the `post-verbal` class is used for clefted-arguments in cleft sentence. In order to represent the relativized-subject in (14), and the clefted-subject in (15), tree schemata are proposed in Fig.(5). The tree family for the relativized-subject is represented is by the auxiliary tree with the foot node NP. The tree family for the clefted-subject is selected by the copular *i* as the main verb and a sentential noun phrase (SNP) as the subject.

(15) [$_s$ satang-eul meok-eun] saram-eun (saram) ida.
candy$_{acc}$ eat$_{adn}$ person$_{top}$ (person) be
'It is (person) who ate the candy.'

we can see in (Yoon S.-W., 2003) more detailed explanations about linguistic organizations and about the relationship among the classes in KMG.

[6]The modified NP is now semantically associated with the empty-subject in the relative. The same NP can be modified by an auxiliary tree representing a relativized-object :

(13) [$_s$ Sumi-ga johaha-neun] (saram)
Sumi$_{nom}$ like$_{rel}$ (person)
'(person) [whom Sumi likes]'

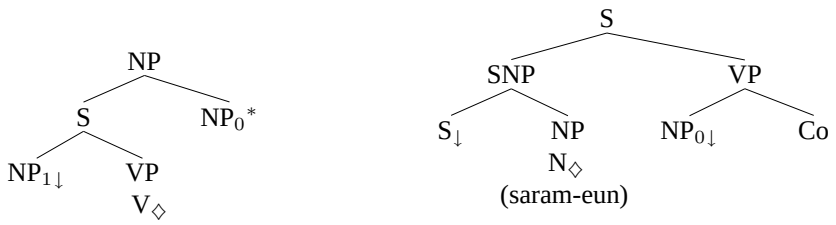


Figure 5: Tree Schemata for relativized and clefted-subject

4 Implementations

The implementation for Korean Meta Grammar is working with the Meta Grammar compiler maintained by B. Gaiffe (Gaiffe et al., 2002). A more specific status will be reported. Korean is a language with a very productive morphological system. In order to handle the morphology with a Korean Meta-Grammar, a MG generates tree templates, the morphology is encoded in the form of a dictionary of inflected terms like the French Meta-Grammar (Abeillé and Candito, 2000). The handling for unbounded dependency phenomena,e.g., non-local scrambling, is one of non-resolved problems with a MG compiler. We propose to use a compiler adapted by the new TAG variant, Tree-local MC TAG with shared nodes(RSN MC TAG), that is used to handle the unbounded dependency phenomena in free-word order variation (see (Kallmeyer and Yoon, 2004)).

5 Conclusion

We offer to develop and implement a wide-coverage LTAG Korean Grammar using a meta-grammar. The 26 tree families for verbs and 9 tree families for adjectives are proposed for Korean LTAG. With the hierarchical grammar, various syntactic phenomena can be covered in a Korean MG. For example, the auxiliary verb constructions, the nominal and/or sentential complements, the raising verb and/or the control verb constructions, the passive and/or causative constructions, and the relative and/or cleft constructions are handled. Furthermore, by suggesting *pre-verbal* and *post-verbal* classes in the syntactic realization dimension, we can also deal with the Korean local scrambling and the (simple) extraposition. The first evaluation for KMG is promising, but more has to improve the lexical coverage by increasing the lexical database, and the grammar coverage by refining the constraints on agrammatical syntactic constructions.

Acknowledgments

For helpful comments and numerous valuable discussions of the subject of this paper, I would like to thank Anne Abeillé, Alexandra Kinyon, Benoit Crabbé, Shi-Jong Ryu and Na-Rae Han. Furthermore, I'm grateful to three anonymous reviewers for their valuable suggestions for improving the paper.

References

Abeillé, A. 2002. *Une Grammaire électronique du FRANCAIS.* CNRS (eds).

Abeillé, A., and Candito, M. -H. 2000. FTAG : A Lexicalized Tree Adjoining Grammar for French. In *Tree Adjoining Grammars.* CSLI (eds).

Aravind. K. Joshi. 1985. Tree Adjoining Grammars : How much context Sensitivity is required to provide a reasonable struictural description. In D. Dowty, I Karttumen, and A. Zwicky, eds., *Natural Language Parsing*, Cambridge University Press, Cambridge, U.K., 206-250J.

Aravind. K. Joshi. 1987. An Introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Languages*, John Benjamins, Amsterdam.

Barrier, S. and Barrier, N. 2003. Une métagrammaire pour les noms prédicatifs du FRANCAIS. In *Proceedings of TALN 03'*.

Barrier, S. and Barrier, N. 2004. Metagrammars: a new implementation for FTAG. In *Proceedings of TAG+7*, Vancouver.

Candito, M. -H. 1996. A Principle-based hierarchical representation of LTAGs. In *Proceedings of COLING 96'*, Copenhagen.

Candito, M. -H. 1999. *Organisation Modulaire et paramétrable de grammaire électroniques lexicalisées*. Ph.D. thesis, University of Paris 7.

Clement L. and Kinyon A. 2003. Generating parallel multilingual LFG-TAG grammars using a MetaGrammar. In *Proceedings of ACL'03.*, Sapporo.

Gaiffe, B. Crabbé, B. Roussanaly, A. 2002. A New Metagrammar Compiler. In *Proceedings of TAG+6* , Venice.

Gerdes, K. 2002. *Topologie et grammaires formelles de l'allemand* Ph.D. thesis, University of Paris 7.

Han C.-H, Yoon J. -T, Kim N. -R, and Palmer Martha. 2000. *A Feature-Based Lexicalized Tree Adjoining Grammar for Korean*. Technical Report IRCS 00-04, University of Pennsylvania.

Tree Families for adjectives	Examples
n0A	haneul-i pureuda. The sky is blue
n0dat1A	Minho-ga modeun saramdeul-ege chinjeolhada. Minho is attentitive to everyone.
n0pn1A	A seon-i B seon-kwa pyeonghaenghada. Line A is paralell to line B.
n0pn1pn2A	Minho-neun Sumi-wa seongkyeok-eseo dareuda. Minho's character is different from Sumi's.
n0nNOM1A	Minho-ga deum-i pilyohada. Minho needs help.
sn0A	Minho-ga onil sukje-reul cechulha-ki-ka eolyeopda. It will be hard for Minho to hand in his homework today.
sn0pn1A	jeongchi anjeong-eul doechassneun-geos-i gyeongjebaljeon-e iopda. It is profitable in the economic development to restore the political stability .
sn0dat1A	Sumi-ga neul honja issda-neun-sasil-i Minho-ege buranseureowossda. It is anxious to Minho that Sumi is always alone.
n0s1A	Minho-ga Sumi-reul manna-go sipeohanda. Minho hopes to see Sumi.

Table 2: Tree Families for adjectival anchors in KMG

Suh J.-S. 1996. *Korean Grammar*. HanYang (eds)., Seoul.

Kallmeyer, L., and Yoon, S.-W. 2004. Tree-local MCTAG with shared Nodes : Word Order variation in German and Korean. In *Proceedings of TAG+7* , Vancouver.

Kinyon A. 2003. *MetaGrammars for efficient development, extraction and generation of parallel grammars*. Ph.D. thesis proposal, University of Pennsylvania.

Kinyon A. and Prolo C. 2002. A classification of Grammar Development Strategies. In *Proceedings of COLING-GEE 02*, Carroll, Oostdijk, Sutcliffe (eds)., Taipei.

Kinyon A. and Rambow O. 2003. The MetaGrammar: across-framework and cross-language test-suite generation tool. In *LINC-EACL-2003*, Budapest.

Nam J.-S. 1994. *Classification Syntaxique des Constructions Adjectivales en Coreen*. Ph.D. thesis, University of Paris 7.

Prolo, C. 2002. Generating the Xtag english grammar using metarules. In *Proceedings of COLING 02'*, Taipei.

Rogers, J. and Vijay-Shanker, K. 1994. Obtaining trees from their description : an application to TAGs. *Computational Intelligence 10:4*.

Lee S.-O. 1999. *A Study of Causative and Passive Constructions in Korean*. JipMunDang (eds). Seoul.

Lee Y. -S. 1993. *Scrambling as Case-driven Obligatory Movement*. Technical Report IRCS 01-03, University of Pennsylvania.

XTAG Research Group. 2001. *A Lexicalized Tree Adjoining Grammar*. Technical Report IRCS 01-03, University of Pennsylvania.

Yoon J.-T., Han C.-H., Kim N.-R. 2000. Customizing the XTAG system for efficient grammar development for Korean. In *Proceedings of TAG+5*, Paris.

Yoon S. -W. 2003. *Une Meta-Grammaire pour Le Coreen.* Manuscrit

Tree Families for verbs	Examples
n0V	Minho-ga janda. Minho is sleeping.
n0ad1V	Sonyo-ga yeppeuge saengkyeossda. The girl looks beautiful.
n0pn1ad2V	Minho-ga Sumi-hante mulyehage kunda. Minho is being rude to Sumi.
n0acc1V	Minho-ga sakwa-reul meokneunda. Minho is eating an apple.
n0acc1adv2V	Minho-neun geu gangaji-reul josimseuleopge darueossda. Minho is stroking his puppy.
n0acc1ACC2V	Minho-ga Sumi-reul pal-eul japassda. Minho is taking Sumi by the arm.
n0dat1V	Minho-ga Sumi-ege malhanda. Minho is speaking to Sumi
n0acc1dat2V	Minho-ga sakwa-reul Sumi-ege jueossda. Minho gave an apple to Sumi.
n0pn1V	ceongbo-ga yeoreo chwulcheo-robuteo giinhada. The information originates from various sources.
n0acc1pn2V	Minho-neun Sumi-reul geunyeo innaesim-edaehae chinchanhassda. Minho admired Sumi for her patience.
n0dat1pn2V	Minho-neun Sumi-ege jigeum sanghwang-edaehae seolmyeonghassda. Minho explained this situation to Sumi.
n0n1CO	Minho-ga uisa ida. Minho is a doctor.
n0NOM1V	Minho-ga kyosu-ga doeossda. Minho became a professor.
sn0V	mulgeon-e haja-ga issda-neun sasil-i deureonassda. A default revealed itself in the product.
sn0acc1V	Sumi-ga tteonassda-n geos-i Minho-reul goerophinda. The fact that Sumi went out upsets Minho.
sn0acc1adv2V	gangaji-ga jugossda-n sasil-i Minho-reul seulpeuge mandeunda. The fact that the puppy died makes Minho sad.
sn0pn1V	Minho-ga eolida-neun-geos-i baesimwon pankyeo-e yeongyangjueossda. The fact that Minho was young influenced on the jury's decision.
n0sacc1V	Minho-neun Sumi-ga jigap-eul humchi-n sasil-eul arassda. Minho noticed that Sumi had stolen his wallet.
n0s1V	Minho-neun chakhage cheosinhaess-dago saenggakhanda. Minho thinks that he behaved very wisely.
n0dat1s2V	Minho-ga Sumi-ege yeonghwakwan-e gasseoss-dago malhaessda. Minho told to Sumi to go to the movies.
n0pn1s2V	Minho-neun sijang-euro saengseon-eul sa-ro hyanghassda. Minho moved to buy fish to the market.
n0acc1s2pn3V	Minho-ga Sumi-reul saengseon-eul sa-ro sijang-euro ponaessda. Minho sent Sumi to buy fish (at the market)
n0acc1s2V	Minho-neun i nonjeungdeul-eul deol seoldeukjeoi-rago saenggakhanda. Minho doen't find these arguments very convincing.
n0s1Vc	Minho-neun honja jip-e namgess-dago gyeolsimhaessda. Minhonom decides to leave alone at home.
n0dat1s2Vc	Minho-ga Sumi-ege Inho-wa tteona-rago kangyohaessda. Minho forced Sumi to leave with Inho.
n0pn1s2Vc	Minho-ga Sumi-wa jadongcha-reul guipha-gilo hapuipoassda. Minho confers with Sumi to buy a car.

Table 3: Tree Families for verbal anchors in KMG