ALGORITHMS FOR GENERATION IN LAMBEK THEOREM PROVING

Erik-Jan van der Linden *
Guido Minnen
Institute for Language Technology and Artificial Intelligence
Tilburg University
PO Box 90153, 5000 LE Tilburg, The Netherlands
E-mail: vdlinden@kub.nl

ABSTRACT

We discuss algorithms for generation within the Lambek Theorem Proving Framework. Efficient algorithms for generation in this framework take a *semantics-driven* strategy. This strategy can be modeled by means of rules in the calculus that are geared to generation, or by means of an algorithm for the Theorem Prover. The latter possibility enables processing of a *bidirectional* calculus. Therefore Lambek Theorem Proving is a natural candidate for a 'uniform' architecture for natural language parsing and generation.

Keywords: generation algorithm; natural language generation; theorem proving; bidirectionality; categorial grammar.

1 INTRODUCTION

Algorithms for tactical generation are becoming an increasingly important subject of research in computational linguistics (Shieber, 1988; Shieber et al., 1989; Calder et al., 1989). In this paper, we will discuss generation algorithms within the Lambek Theorem Proving (LTP) framework (Moortgat, 1988; Lambek, 1958; van Benthem, 1986). In section (2) we give an introduction to a categorial calculus that is extended towards bidirectionality. The naïve top-down control strategy in this section does not suit the needs of efficient generation. Next, we discuss two ways to implement a *semantics-driven* strategy. Firstly, we add inference rules and cut rules geared to generation to the *calculus* (3). Secondly, since these changes in the calculus do not support bidirectionality, we introduce a second implementation: a bottom-up algorithm for the theorem prover (4).

*We would like to thank Gosse Bouma, Wietske Sijtsma and Marianne Sanders for their comments on an earlier draft of the paper.

2 EXTENDING THE CALCULUS

Natural Language Processing as deduction The architectures in this paper resemble the uniform architecture in Shieber (1988) because language processing is viewed as *logical deduction*, in analysis and generation:

"The generation of strings matching some criteria can equally well be thought of as a deductive process, namely a process of constructive proof of the existence of a string that matches the criteria." (Shieber, 1988, p. 614).

In the LTP framework a categorial reduction system is viewed as a *logical calculus* where parsing a syntagm is an attempt to show that it follows from a set of axioms and inference rules. These inference rules describe *what* the processor does in assembling a semantic representation (representational non-autonomy: Crain and Steedman, 1982; Ades and Steedman, 1982). Derivation trees represent a particular parse process (Bouma, 1989). These rules thus *seem* to be nondeclarative, and this raises the question whether they can be used for generation. The answer to this question will emerge throughout this paper.

Lexical information As in any categorial grammar, linguistic information in LTP is for the larger part represented with the signs in the lexicon and not with the rules of the calculus (signs are denoted by **prosody:syntax:semantics**). A generator using a categorial grammar needs lexical information about the syntactic form of a functor that is connected to some semantic functor in order to syntactically correctly generate the semantic arguments of this functor. For a parser, the reverse is true. In order to fulfil both needs, lexical information is made available to the theorem prover in the form of *instances of axioms.*[1] Axioms then truely represent what should be axiomatic in a lexicalist description of a language: the lexical items, the connections between form and meaning.[2]

Rules Whenever inference rules are applied, an attempt is made to axiomatize the functor that participates in the inference by the first subsequent of the elimination rules. This way, lexical information is retrieved from the lexicon.

A prosodic operator connects prosodic elements. A prosodic identity element, *id*, is necessary because introduction rules are prosodically vacuous. In order to avoid unwanted matching between axioms and id-elements, one special axiom is added for id-elements. Meta-logical checks are included in the rules in order to avoid variables occuring in the final derivation. *nogenvar* recursively checks whether any part of an expression is a variable.

A *sequent* in the calculus is denoted with P => T, where P, called the antecedent, and T, the succedent, are finite sequences of signs. The calculus is presented in (1) . In what follows, X and Y are categories; T and Z, are signs; R, U and V are possibly empty sequences of signs; @ denotes functional application, a caret denotes λ-abstraction.[3]

(1)

```
/* axioms */
[Pros:X:Y] => [Pros:X:Y] <-
  [Pros:X:Y] =1> [Pros:X:Y] &
  true.

[Pros:X:Y] => [Pros:X:Y] <-
  (nogenvar(X), nonvar(Y)) &
  true.
```

```
/* elimination rules */
(U,[Pros_Fu:X/Y:Functor],[T|R],V)=>[Z] <-
  [Pros_Fu:X/Y:Functor] =>
                        [Pros_Fu:X/Y:Functor] &
  [T|R] => [Pros_Arg:Y:Arg] &
  (U,[(Pros_Fu*Pros_Arg):X:Functor@Arg],V) =>
                                            [Z].

(U,[T|R],[Pros_Fu:Y\X:Functor],V) => [Z] <-
  [Pros_Fu:Y\X:Functor] =>
                        [Pros_Fu:Y\X:Functor] &
  [T|R] => [Pros_Arg:Y:Arg] &
  (U,[(Pros_Arg*Pros_Fu):X:Functor@Arg],V) =>
                                            [Z].

/* introduction rules */
[T|R]=>[Pros:Y\X:Var_Y^Term_X] <-
  nogenvar(Y\X) &
  ([id:Y:Var_Y],[T|R]) =>
                        [(id*Pros):X:Term_X].

[T|R] => [Pros:X/Y:Var_Y^Term_X] <-
  nogenvar(X/Y) &
  ([T|R],[id:Y:Var_Y]) =>
                        [(Pros*id):X:Term_X].

/* axiom for prosodic id-element */
[id:X:Y] =1> [id:X:Y] <-
  isvar(Y).

/* lexicon, lexioms */
[john:np:john] =1> [john:np:john].
[mary:np:mary] =1> [mary:np:mary].
[loves:(np\s)/np:loves] =1>
                    [loves:(np\s)/np:loves].
```

In order to initiate analysis, the theorem prover is presented with sequents like (2). Inference rules are applied recursively to the antecedent of the sequent until axioms are found. This regime can be called *top-down* from the point of view of problem solving and *bottom-up* from a "parsing" point of view. For generation, a sequent like (3) is presented to the theorem prover. Both analysis and generation result in a derivation like (4). Note that generation not only results in a sequence of lexical signs, but also in a *prosodic phrasing* that could be helpful for speech generation.

(2)

```
[john:A:B,loves:C:D,mary:E:F] => [Pros:s:Sem]
```

(3)

```
U => [Pros:s:loves@mary@john]
```

Although both (2) and (3) result in (4), in the case of generation, (4) does not represent the

[1]Van der Linden and Minnen (submitted) contains a more elaborate comparison of the extended calculus with the original calculus as proposed in Moortgat (1988).

[2]A suggestion similar to this proposal was made by König (1989) who stated that lexical items are to be seen as axioms, but did not include them as such in her description of the L-calculus.

[3]Throughout this paper we will use a Prolog notation because the architectures presented here depend partly on the Prolog unification mechanism.

(4)

```
john:np:john loves:(np\s)/np:loves mary:np:mary => john*(loves*mary):s:loves@mary@john <-
   loves:(np\s)/np:loves => loves:(np\s)/np:loves <-
      loves:(np\s)/np:loves =l> loves:(np\s)/np:loves <- true
   mary:np:mary => mary:np:mary <-
      mary:np:mary =l> mary:np:mary <- true
   john:np:john loves*mary:np\s:loves@mary => john*(loves*mary):s:loves@mary@john <-
      loves*mary:np\s:loves@mary => loves*mary:np\s:loves@mary <- true
      john:np:john => john:np:john <-
         john:np:john =l> john:np:john <- true
      john*(loves*mary):s:loves@mary@john => john*(loves*mary):s:loves@mary@john:<- true
```

exact proceedings of the theorem prover. It starts applying rules, matching them with the antecedent, without making use of the original semantic information, and thus resulting in an inefficient and nondeterministic generation process: all possible derivations including all lexical items are generated until some derivation is found that results in the succedent.[4] We conclude that the algorithm normally used for parsing in LTP is inefficient with respect to generation.

3 CALCULI DESIGNED FOR GENERATION

A solution to the efficiency problem raised in the previous section is to start from the original semantics. In this section we discuss *calculi* that make explicit use of the original semantics. Firstly, we present Lambek-like rules especially designed for generation. Secondly, we introduce a Cut-rule for generation with sets of categorial reduction rules. Both entail a variant of the crucial starting-point of the *semantic-head*-driven algorithms described in Calder et al. (1989) and Shieber et al. (1989): if the functor of a semantic representation can be identified, and can be related to a lexical representation containing syntactic information, it is possible to generate the arguments syntactically. The efficiency of this strategy stems from the fact that it is guided by the known semantic and syntactic information, and lexical information is retrieved as soon as possible.

In contrast to the semantic-head-driven approach, our semantic representations do not allow for immediate recognition of semantic heads: these can only be identified after all arguments have been stripped of the functor recursively (loves@mary@john => loves@mary => loves).

Calder et al. conjecture that their algorithm

"(...) extends naturally to the rules of composition, division and permutation of Combinatory Categorial Grammar (Steedman, 1987) and the Lambek Calculus (1958)" (Calder et al., 1989, p. 237).

This conjecture should be handled with care. As we have stated before, inference rules in LTP describe *how* a processor operates. An important difference with the categorial reduction rules of Calder et al. is that inference-rules in LTP implicitly initiate the recursion of the parsing and generation process. Technically speaking, Lambek rules cannot be arguments of the rule-predicate of Calder et al. (1989, p. 237). The gist of our strategy is similar to theirs, but the algorithms differ.

Lambek-like generation Rules are presented in (5) that explicitly start from the known information during generation: the syntax and semantics of the succedent. Literally, the inference rule states that a sequent consisting of an antecedent that unifies with two sequences of signs U and V, and a succedent that unifies with a sign with semantics **Sem_Fu@Sem_Arg** is a theorem of the calculus if V reduces to a syntactic functor looking for an argument on its left side with the functor-meaning of the original semantics, and U reduces to its argument. This rule is an equivalent of the second elimination rule in (1).

[4]cf. Shieber et al. (1989) on top-down generation algorithms.

(5)

```
/* elimination rule */
[U,V] =>
     [(Pros_Arg*Pros_Fu):X:Sem_Fu@Sem_Arg] <-
  V =>[Pros_Fu:Y\X:Sem_Fu] &
  U =>[Pros_Arg:Y:Sem_Arg].

/* introduction-rule */
[T|R] => [Pros:Y\X:Var_Y^Term_X] <-
     nogenvar(Y\X) &
     ([[id:Y:Var_Y]],[T|R]) =>
                         [(id*Pros):X:Term_X].
```

A Cut-rule for generation A *Cut-rule* is a structural rule that can be used within the L-calculus to include partial proofs derived with categorial reduction rules into other proofs. In (6) a generation Cut-rule is presented together with the AB-system.

(6)

```
/* Cut-rule for generation */
[U,V] => [Pros_Z:Z:Sem_Z] <-
  [Pros_X:X:Sem_X, Pros_Y:Y:Sem_Y] =*>
                              [Pros_Z:Z:Sem_Z]
  U => [Pros_X:X:Sem_X] &
  V => [Pros_Y:Y:Sem_Y].

/* reduction rules, system AB */
[Pros_Fu:X/Y:Functor, Pros_Arg:Y:Arg] =*>
          (Pros_FU*Pros_Arg):X:Functor@Arg].

[Pros_Arg:Y:Arg, Pros_Fu:Y\X:Functor] =*>
          (Pros_Arg*Pros_Fu):X:Functor@Arg].
```

The generator regimes presented in this section are semantics-driven: they start from a semantic representation, assume that it is part of the uppermost sequent within a derivation, and work towards the lexical items, axioms, with the recursive application of inference rules. From the point of view of theorem proving, this process should be described as a top-down problem solving strategy. The rules in this section are, however, geared towards generation. Use of these rules for parsing would result in massive non-determinism. Efficient parsing and generation require different rules: the calculus is not *bidirectional.*

4 A COMBINED BOTTOM-UP/TOP-DOWN REGIME

In this section, we describe an algorithm for the theorem prover that proceeds in a combined bottom-up/top-down fashion from the problem solving point of view. It maintains the same semantics-driven strategy, and enables efficient generation with the bidirectional calculus in (1). The algorithm results in derivations like (4), in the same theorem prover architecture, be it along another path.

Bidirectionality There are two reasons to avoid duplication of grammars for generation and interpretation. Firstly, it is theoretically more elegant and simple to make use of one grammar. Secondly, for any language processing system, human or machine, it is more economic (Bunt, 1987, p. 333). Scholars in the area of language generation have therefore pleaded in favour of the *bidirectionality* of linguistic descriptions (Appelt, 1987).

Bidirectionality might in the first place be implemented by using one grammar and two separate algorithms for analysis and generation (Jacobs, 1985; Calder et al., 1989). However, apart from the desirability to make use of one and the same grammar for generation and analysis, it would be attractive to have one and the same *processing architecture* for both analysis and generation. Although attempts to find such architectures (Shieber, 1988) have been termed "looking for the fountain of youth",[5] it is a stimulating question to what extent it is possible to use the same architecture for both tasks.

Example An example will illustrate how our algorithm proceeds. In order to generate from a sign, the theorem prover assumes that it is the succedent of one of the *subsequents* of one of the inference rules (7-1/2). (In case of an introduction rule the sign is matched with the succedent of the *headsequent*; this implies a top-down step.) If unification with one of these subsequents can be established, the other subsequents and the headsequent can be partly instantiated. These sequents can then serve as starting points for further bottom-up processing. Firstly, the headsequent is subjected to bottom-up process-

[5]Ron Kaplan during discussion of the Shieber presentation at Coling 1988.

(7)

Generation of nounphrase *the table.* Start with sequent

```
P => [Pros:np:the@table]
```

1- Assume succedent is part of an axiom:

```
[Pros:np:the@table] => [Pros:np:the@table]
```

2- Match axiom with last subsequent of an inference rule:

```
(U,[Pros_Fu:X/Y:Functor],[T|R],V) => [Z] <-
     [Pros_Fu:X/Y:Functor] => [Pros_Fu:X/Y:Functor] &
     [T|R] => [Pros_Arg:Y:Arg] &
     (U,[(Pros_Fu*Pros_Arg):X:Functor@Arg],V) => [Z].
```

Z = Pros:np:the@table; Functor = the; Arg = table; X = np; U = [ ]; V = [ ].

3- Derive instantiated head sequent:

```
[Pros_Fu:np/Y:the],[T|R] => [Pros:np:the@table]
```

4- No more applications in head sequent: Prove (bottom-up) first instantiated subsequent:

```
[Pros_Fu:np/Y:the] => [Pros_Fu:np/Y:the]
```

Unifies with the axiom for "the": Pros_Fu = the; Y = n.

5- Prove (bottom-up) second instantiated subsequent:

```
[T|R]=>[Pros_Arg:n:table]
```

Unifies with axiom for "table": Pros_Arg = table; T = table:n:table; R = [ ]

6- Prove (buttum-up) last subsequent: is a nonlexical axiom.

```
[(the*table):np:the@table] => [(the*table):np:the@table].
```

7- Final derivation:

```
the:np/n:the table:n:table => the*table:np:the@table <-
   the:np/n:the => the:np/n:the <-
      the:np/n:the =1> the:np/n:the <- true
   table:n:table => table:n:table <-
      table:n:table =1> table:n:table <- true
   the*table:np:the@table => the*table:np:the@table <- true
```

ing (7-3), in order to axiomatise the head functor as soon as possible. Bottom-up processing stops when no more application operators can be eliminated from the head sequent (7-4). Secondly, working top-down, the other subsequents (7-4/5) are made subject to bottom-up processing, and at last the last subsequent (7-6). (7) presents generation of a nounphrase, *the table*.

5 CONCLUDING REMARKS

Conclusion Efficient, bidirectional use of categorial calculi is possible if extensions are made with respect to the calculus, and if a combined bottom-up/top-down algorithm is used for generation. Analysis and generation take place within the same processing architecture, with the same linguistics descriptions, be it with the use of different algorithms. LTP thus serves as a natural candidate for a uniform architecture of parsing and generation.

Semantic non-monotonicity A constraint on grammar formalisms that can be dealt with in current generation systems is semantic monotonicity (Shieber, 1988; but cf. Shieber et al., 1989). The algorithm in Calder et al. (1989) requires an even stricter constaint. Firstly, in van der Linden and Minnen (submitted) we describe how the addition of a unification-based semantics to the calculus described here enables processing of non-monotonic phenomena such as non-compositional verb particles and idioms. Identity semantics (cf. Calder et al. p. 235) should be no problem in this respect. Secondly, unary rules and type-raising (ibid.) are part of the L-calculus, and are neither fundamental problems.

Inverse β-reduction A problem that exists for all generation systems that include some form of λ-semantics is that generation necessitates the inverse operation of β-reduction. Although we have implemented algorithms for inverse β-reduction, these are not computationally tractable.[6] A way out could be the inclusion of a unification based semantics.[7]

[6]Bunt (1987) states that an expression with n constants results in $2^n - 1$ possible inverse β-reductions.

[7]As proposed in van der Linden and Minnen (submitted) for the calculus in (2).

Non-determinism A source for non-determinism in the semantics-driven strategy is the fact that the theorem prover forms hypotheses about the direction a functor seeks its arguments, and then checks these against the lexicon. A possibility here would be to use a calculus where dominance and precedence are taken apart. We will pursue this suggestion in future research.

Implementation The algorithms and calculi presented here have been implemented with the use of modified versions of the categorial calculi interpreter described in Moortgat (1988).

6 REFERENCES

Ades, A., and Steedman, M., 1982 On the order of words. *Linguistics and Philosophy*, 4, pp. 517-558.

Appelt, D.E., 1987 Bidirectional Grammars and the Design of Natural Language Systems. In Wilks, Y. (Ed.), Theoretical Issues in Natural Language Processing. Las Cruces, New Mexico: New Mexico State University, January 7-9, pp. 185-191.

Van Benthem, J., 1986 Categorial Grammar. Chapter 7 in Van Benthem, J., *Essays in Logical Semantics*. Reidel, Dordrecht.

Bouma, G., 1989 Efficient Processing of Flexible Categorial Grammar. In Proceedings of the EACL 1989, Manchester. pp. 19-26.

Bunt, H., 1987 Utterance generation from semantic representations augmented with pragmatic information. In Kempen 1987.

Calder, J., Reape M., and Zeevat, H., 1989 An algorithm for generation in Unification Categorial Grammar. In Proceedings of the EACL 1989, Manchester. pp. 233-240.

Crain, S., and Steedman, M., 1982 On not being led up the garden path. In Dowty, Karttunen and Zwicky (Eds.) *Natural language parsing*. Cambridge: Cambridge University Press.

Jacobs, P., 1985 PHRED, A generator for Natural Language Interfaces. Computational Linguistics 11, 4, pp. 219-242.

Kempen , G., (Ed.) 1987 *Natural language generation: new results in artificial intelligence, psychology and linguistics.* Dordrecht: Nijhoff.

König, E., 1989 Parsing as natural deduction. In Proceedings of the ACL 1989, Vancouver.

Lambek, J., 1958 The mathematics of sentence structure. *Am. Math Monthly*, 65, 154-169.

Linden, E. van der, and Minnen, G., (submitted) An account of Non-monotonous phenomena in bidirectional Lambek Theorem Proving.

Moortgat, M., 1988 *Categorial Investigations. Logical and linguistic aspects of the Lambek calculus.* Disseration, University of Amsterdam.

Shieber, S., 1988 A uniform architecture for Parsing and Generation. In Proceedings of Coling 1988, Budapest, pp. 614-619.

Shieber, S., van Noord, G., Moore, R., and Pereira, P., 1989 A semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms. In Proceedings of ACL 1989 Vancouver.

Steedman, M., 1987 Combinatory Grammars and Parasitic Gaps *Natural Language and Linguistic Theory*, 5, pp. 403-439.