Combining Lexicon-Driven Parsing and Phrase-Structure-Based Parsing

Masaru Tomita
Center for Machine Translation
and
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA15213, USA

Abstract

Lexicon-driven formalisms (e.g. Categorial Grammar, HPSG and GB Grammar), which do not have explicit phrase structure rules, are suitable for higher level syntax but not for low level language-specific constructions such as dates (July 15th, 15-JUL-1987, etc), which seem to require the power of phrase structure rules. This paper presents an implementation method to combine lexicon-driven parsing and phrase structure parsing, with a special means called *graph-structured stack*.

1. Introduction

Recent linguistic grammar formalisms can be classified into two families, *phrase-structure-based* or *lexicon-driven*. The phrase-structure-based formalisms include Definite Clause Grammar [8], Lexical Functional Grammar [3], Generalized Phrase Structure Grammar [5, 6] and Functional Unification Grammar [7]. They all are based on context-free phrase structure rules which are augmented in one way or another. On the other hand, lexicon-driven formalisms include Categorial Grammar [1], Head-driven Phrase Structure Grammar [9, 10] and GB-Grammar [4, 13]. All of these lexicon-driven formalisms do not have any explicit phrase structure rules, but information about how to combine constituents into a higher constituent is encoded in each lexicon or constituent which is to be combined. It is often argued that lexicon-driven formalisms can handle some linguistic phenomena such as free-word-order more elegantly, and also they can capture universality of multiple languages, as described, for example, by Wehrli [14].

In section 2, on the other hand, it is argued that, while lexicon-driven formalisms might be suitable to cope with linguistically "interesting" phenomena, they lack the power to express linguistically "uninteresting" phenomena which are often very specific to a particular language. Unfortunately, in order to build a practical parser for a "real" text, one must handle very many "uninteresting" low level phenomena which cannot be easily formalized in the lexicon-driven way. Hence, if we want to build a practical parser with a theory behind the lexicon-driven formalisms, then it is essential to combine lexicon-driven parsing (for higher level syntax) and phrase-structure-based parsing (for low level detail and other language-specific/exceptional constructions).

In section 3, we generalize the computational models of all the lexicon-driven and phrase-structure-based formalisms as *shift-reduce parsing*. Section 4 introduces the *graph-structured stack* to handle non-determinism in shift-reduce parsing. In sections 5 and 6, we describe the use of the graph-structured stack in Lexicon-driven and Phrase-structure-based parsing, respectively. We then discuss how to combine these two kinds of parsing with the graph-structured stack, in section 7.

2. Arguments against Lexicon-Driven Parsing

In practical applications, input sentences have many language-specific phenomena that are not linguistically "interesting". Consider the following sentences:

Take two tablets of aspirin *four times a day*.

Add *a paragraph or two* to *section 3.4*.

Schedule a meeting with *Doctor Jaime Carbonell* on *July 15th* from *3:30 PM*.

All of those sentences contain no interesting syntax phenomena, but have a lot of language-specific low-level constructions, such as "four times a day". It seems that there is no better way to specify this pattern than a phrase structure rule:

<frequency> <-- <integer> <times> <integer> <time-unit>

augmented in one way or another to check number agreement etc. Similar arguments can be made for rules:

<person-name> <-- <title> <first-name> <last-name>

<time> <-- <integer> ":" <integer> ("AM" | "PM")

These constructions are "uninteresting" because we all know that it is tedious to specify those phenomena, and it is just a matter of sitting down and writing rules. Therefore, for the purpose of linguistic study, they can be just ignored. In practical applications, however, we cannot ignore them by any means. In fact, input sentences are full of those "uninteresting" phenomena, unfortunately. More than half of English and Japanese grammar rules for CMU's machine translation project, for example, are language-specific "uninteresting" low-level rules. It is expected that, in any practical system, there must be a bunch of rules like those somewhere in the system; otherwise the system simply does not work.

Lexicon-driven formalisms are all motivated by linguistically "interesting" phenomena, and they are not suited for those language-specific low level constructions. It does not make any sense to try to write those rules in the lexicon-driven way, even if it is mathematically possible. It is clear that the power of phrase structure is necessary for practical applications. In the following sections, we describe how to combine lexicon-driven parsing and phrase-structure-based parsing.

3. Shift-Reduce Parsing

Let us consider both lexicon-driven and phrase-structure-based parsing to be shift-reduce parsing. In shift-reduce parsing, there is a *stack*, and all words in the input sentence are pushed (shifted) onto the stack from left to right. In doing so, the first n constituents (or words) from the top of the stack are occasionally combined (reduced), by popping the n constituents from the stack, computing a new constituent from the constituents, and pushing the new constituent onto the stack. A shift-reduce parser needs to be

controlled to determine when and how constituents should be reduced, and when a word should be shifted.

In phrase-structure-based parsing, it is controlled by a set of language-specific rules, typically a context-free phrase structure grammar, augmented in one way or another.

In lexicon-driven parsing, it is controlled by information encoded in each constituent to be reduced, and (if any) a set of language universal rules which reside inside the parser permanently, but with no language-specific outside rules. In the Categorial Grammar formalism, this information is encoded as complex categories with *functor* and *argument* features. Two elements can be reduced if and only if the category of one element is the same as the argument feature of the other element, and the category of the new element will be the functor of the latter element. In GB Parsing, categories are encoded in accordance with the X-bar theory, and some language-universal rules (often called *principles*) are fired to decide whether a particular pair of elements can be reduced.

In both phrase-structure-based and lexicon-driven parsing, there are cases where it cannot be uniquely determined whether the next action should be reduce or shift, or cases where more than one reduce action is possible at the same time. This means that the parser must handle some non-determinism, and naive techniques such as simple back tracking or breath first search would require exponential time. The parsing time, however, can be reduced to polynomial with a *graph-structured stack*, which is described in the following section.

4. The Graph-Structured Stack

The graph-structured stack was introduced in the *Generalized LR Parsing* algorithm [11, 12], to handle non-determinism in LR parsing in polynomial time. In this section, three key notions, splitting, combining and local ambiguity packing, are described.

4.1. Splitting

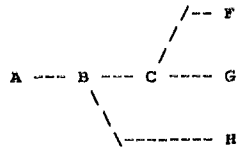
When a stack needs to be reduced (or popped) in more than one way, the top of the stack is split as in the following example.

Suppose that the stack is in the following state. The left most element, A, is the bottom of the stack, and the right most element, E, is the top of the stack. In a graph-structured stack, there can be more than one top, whereas there can be at most one bottom.

Suppose that the stack needs to be reduced in the following three different ways.

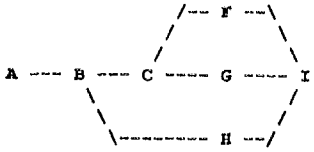
```
F <-- D E
G <-- D E
H <-- C D E
```

Then after the three reduce actions, the stack looks like:



4.2. Combining

When an element needs to be shifted (pushed) onto two or more tops of the stack, it is done only once by combining the tops of the stack. For example, if "I" is to be shifted to F, G and H in the above example, then the stack will look like:

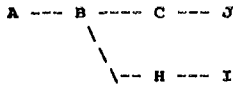


4.3. Local Ambiguity Packing

If two or more branches of the stack turned out to be identical, then they represent local ambiguity; the identical state of stack has been obtained in two or more different ways. They are merged and treated as a single branch. Suppose we have two rules:

```
J <-- F I
J <-- G I
```

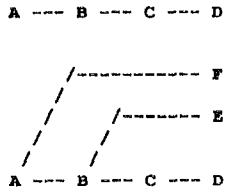
After applying these two rules to the example above, the stack will look like:



The branch of the stack, "A-B-C-J", has been obtained in two ways, but they are merged and only one is shown in the stack.

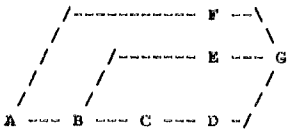
5. Lexicon-Driven Parsing with a Graph-Structured Stack

There is an obvious way to implement a lexicon-driven parser with the graph-structured stack. Basically, the parser parses a sentence strictly from left to right, shifting a word onto the stack one by one. The tops of the stack are inspected to see if there are any ways to reduce the stack (remember, there are no outside phrase structure rules in lexicon-driven parsing). The stack gets reduced *non-destructively*, wherever possible. A non-destructive reduce action simply adds a new branch to the stack, without removing the old branch of the stack that has been reduced. The following example shows the stack before and after the reductions of C and D into E, and B and E into F.



Repeat until no further reduce action is possible (note that one reduce action can trigger another reduce action). When no further reduce action is possible, the next word is shifted. When a word is shifted, all tops of the stack are inspected whether they can be shifted the word onto. Even when a word has been shifted onto more than one top of the stack, there is at most one element on the top of the stack, due to stack combining described in the previous

section. Thus, if a word G is shifted onto F, E and D, then the stack will look like:



When an how top elements can be reduced is determined by information encoded in each element itself, and the method varies from one formalism to another, as described in section 2.

6. Phrase-Structure-Based Parsing with a Graph-Structured Stack

Tomita [11, 12] introduced a generalized LR parsing algorithm, which is an LR parsing algorithm generalized to handle arbitrary context-free grammars with the graph-structured stack. The standard (not generalized) LR parsing algorithm is one of the most efficient parsing algorithms. It is totally deterministic, no backtracking or search is involved, and it runs in linear time. This standard LR parsing algorithm, however, can deal with only a small subset of context-free grammars called *LR grammars*, which is often sufficient for programming languages but clearly not for natural languages. If, for example, a grammar is ambiguous, then its LR table would have *multiple entries*, and hence deterministic parsing would be no longer possible. By introducing the graph-structured stack, however, multiple entries can be handled efficiently in polynomial time.

Figures 6-1 and 6-2 show an example of a non-LR grammar and its LR table. Grammar symbols starting with "*" represent pre-terminals. Entries "sh *n*" in the action table (the left part of the table) indicate that the action "shift one word from input buffer onto the stack, and go to state *n*". Entries "re *n*" indicate that the action "reduce constituents on the stack using rule *n*". The entry "acc" stands for the action "accept", and blank spaces represent "error". The goto table (the right part of the table) decides to what state the parser should go after a reduce action. The LR parsing algorithm pushes state numbers (as well as constituents) onto the stack; the state number on the top of the stack indicates the current state. The exact definition and operation of the LR parser can be found in Aho and Ullman [2].

We can see that there are two multiple entries in the action table; on the rows of state 11 and 12 at the column labeled "*prep". Roughly speaking, this is the situation where the parser encounters a preposition of a PP right after a NP. If this PP does not modify the NP, then the parser can go ahead to reduce the NP to a higher nonterminal such as PP or VP, using rule 6 or 7, respectively (re6 and re7 in the multiple entries). If, on the other hand, the PP does modify the NP, then the parser must wait (sh6) until the PP is completed so it can build a higher NP using rule 5.

With a graph-structured stack, we can handle these non-deterministic phenomena nicely. Figure 6-3 shows the graph-structured stack right after shifting the word "with" in the sentence "I

```
(1)   S --> NP VP
(2)   S --> S PP
(3)  NP --> *n
(4)  NP --> *det *n
(5)  NP --> NP PP
(6)  PP --> *prep NP
(7)  VP --> *v NP
```

Figure 6-1: An Example Ambiguous Grammar

State	*det	*n	*v	*prep	$	NP	PP	VP	S
0	sh3	sh4				2			1
1				sh6	acc		5		
2			sh7	sh6			9	8	
3		sh10							
4			re3	re3	re3				
5				re2	re2				
6	sh3	sh4				11			
7	sh3	sh4				12			
8				re1	re1				
9			re5	re5	re5				
10			re4	re4	re4				
11			re6	re6,sh6	re6		9		
12				re7,sh6	re7		9		

Figure 6-2: LR Parsing Table with Multiple Entries (derived from the grammar in fig 6-1)

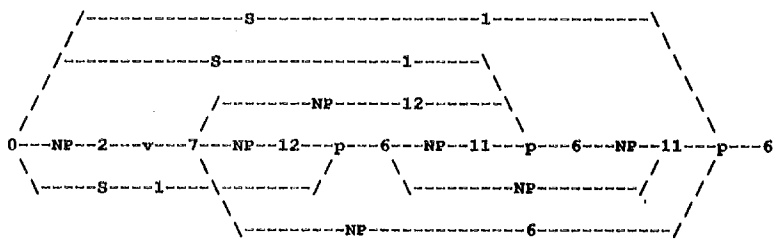


Figure 6-3: A Graph-Structured Stack

saw a man on the bed in the apartment with a telescope." More discussions on the generalized LR parsing algorithm can be found in Tomita [11, 12].

7. Combining Phrase-Structure-Based and Lexicon-Driven Parsing

In the previous sections, we have described the use of the graph-structured stack for lexicon-driven and phrase-structure-based parsing. It is now rather obvious that we can combine lexicon-driven and phrase-structure-based parsing; there are two ways.

We can enhance the generalized LR parsing algorithm by allowing lexicons to (optionally) have lexicon-driven rules. That is, the stack is periodically inspected to see if any part of the stack can be reduce by a lexicon-driven rule inside lexicons. This way is preferable if a system has many phrase structure rules, with a few lexicon-driven rules to handle certain linguistic phenomena more easily.

Alternatively, we can enhance lexicon-driven parsing described in section 5 by allowing the parser to reduce tops of the stack using outside phrase structure rules as well. That is, the stack is periodically inspected to see if any part of the stack can be reduced using one of the phrase-structure rules. This way is preferable if a system is heavily lexicon-driven, with a few phrase structure rules to handle low level and/or exceptional phenomena.

8. Summary

This paper first argued that it is necessary to enhance lexicon-driven parsing with phrase structure rules in practical applications. We then suggested a shift-reduce implementation of lexicon-driven parsing with a graph-structured stack, which is readily combinable with a shift-reduce implementation of phrase-structure-based parsing with a graph-structured stack (i.e., the generalized LR algorithm).

References

[1] Ades, A. E. and Steedman, M. J.
On the Order of Words.
Linguistics and Philosophy 4(4):517-558, 1982.

[2] Aho, A. V. and Ullman, J. D.
Principles of Compiler Design.
Addison Wesley, 1977.

[3] Bresnan, J. and Kaplan, R.
Lexical-Functional Grammar: A Formal System for Grammatical Representation.
The Mental Representation of Grammatical Relations.
MIT Press, Cambridge, Massachusetts, 1982, pages pp. 173-281.

[4] Chomsky, N.
Lectures on Government and Binding.
Foris Publications, 1981.

[5] Gazdar, G.
Phrase Structure Grammars and Natural Language.
Proceedings of IJCAI83 v.1, August, 1983.

[6] Dowty, D. R., Karttunen, L. and Zwicky, A. M. (editor).
Generalized Phrase Structure Grammar.
Harvard University Press, Cambridge, Mass., 1985.

[7] Kay, M.
Parsing in Functional Unification Grammar.
Natural Language Parsing: Psychological, Computational and Theoretical Perspectives.
Cambridge University Press, Cambridge, England, 1985, pages 251-278, Chapter 7.

[8] Pereira, F. and Warren, D.
Definite Clause Grammar for Language Analysis.
Artificial Intelligence 13:pp.231-278, May, 1980.

[9] Pollard, C.
Generalized Phrase Structure Grammars, Head Grammars, and Natural Languages.
PhD thesis, Stanford University, 1984.

[10] Pollard, C.
Lecture Notes on Head-driven Phrase-structure Grammar.
1985.

[11] Tomita, M.
Efficient Parsing for Natural Language.
Kluwer Academic Publishers, Boston, MA, 1985.

[12] Tomita, M.
An Efficient Augmented-Context-Free Parsing Algorithm.
Computational Linguistics 13(1-2):31-46, January-June, 1987.

[13] Wehrli, E.
A Government-Binding Parser for French.
Working Paper 48, Institut pour les Etudes Semantiques et Cognitives, Universite de Geneve, 1984.

[14] Wehrli, E.
Parsing with a GB-grammar.
1987.