DIRECT BOTTOM-UP CHART PARSING USING ABBREVIATED CONTEXT-FREE GRAMMARS

Guido Minnen, William Thompson, Harry Bliss, Dale Russell
Motorola Labs, Schaumburg, IL 60196, USA
{minnen,thompson,bliss,russell}@labs.mot.com

Abstract

In practical applications, the speed with which grammars are loaded into a parser can be of prime importance, particularly when they have to be loaded dynamically. In this paper we address this issue by describing an extension to a bottom-up chart parser that enables it to directly process input utterances using context-free grammars represented in abbreviated form. As a result of this extension, the expensive process of eliminating abbreviatory notation can be avoided, leading to faster grammar loading and parsing times. Our parser is evaluated with respect to a number of realistic grammars of varying sizes, by comparing it with a parser that first expands these grammars during grammar loading.

1 Introduction

Some practical natural language applications require that grammars be loaded dynamically, because the particular set of grammars to be used is not known in advance. For example, a voice-based web browser for VoiceXML content must be able to load grammars when a web page is loaded. The viability of such applications depends on the ability to quickly and efficiently load grammars into internal memory from the textual format in which they are specified.[1] This problem is especially severe for grammars that make extensive use of abbreviatory notation, due to the fact that these grammars are typically expanded before they can be used to parse an input utterance. The expansion process eliminates the abbreviatory notation by producing an equivalent set of context-free grammar rules.

We describe a chart parser that directly processes input utterances using context-free grammars represented in abbreviated form. As a result of the ability to directly parse using abbreviated context-free grammars, it is no longer necessary to eliminate the abbreviatory notation used in the textual representation of the grammar. Consequently, significant improvements in grammar loading and parsing times are achieved. In addition, eliminating the expansion step produces a significant decrease in the footprint of the parser, which is beneficial if the parser is to be employed on devices with limited memory capacity, such as a cell phone or personal digital assistant.

Our parser supports all standard abbreviatory devices within grammar rules, including those for repeated, alternative, and optional elements. Our parser also supports a variety of context-free grammar notations. Importantly, it supports the use of semantic construction rules in conjunction with abbreviatory notation. Rather than expanding the abbreviated rules of a grammar, the described parser compiles the right-hand sides of these rules into finite state automata, with each transition labeled by an identifier referring to a semantic construction rule and a terminal or category label. While parsing an input utterance, the chart parser identifies paths through these automata on the basis of the words in the input utterance and previously computed intermediate results stored in the chart. A parse is complete once a result spanning the complete input utterance has been found, and the semantic representation of the input utterance is constructed by executing the relevant semantic construction rules determined during the parsing process via the semantic identifiers.

2 Related Work

Our parser is closely related to the SOUP parser as described in Gavalda (2000) which is inspired by Ward's Phoenix Parser (Ward, 1990). Gavalda proposes a chart parsing algorithm that processes input utterances in a top-down fashion, using recursive transition networks that are automatically derived from grammars in the Java Speech Grammar Format (JSGF, 1998). The parser described here differs from the SOUP parser in that it

[1] In such a setup, speeding up grammar loading times using a binary save and restore mechanism, i.e., a byte-code representation, is not an option as the grammars that need to be loaded are not known in advance.

processes input utterances in a bottom-up fashion. We conjecture that a bottom-up approach improves upon Gavalda's algorithm with respect to efficiency in the case of processing fragmentary input, resulting from speech recognition errors and/or ungrammatical utterances.

Our parser is also related to that of Earley (1970), which shows that a chart parser can be extended to deal with grammar rules that express repetition using Kleene star. However, Earley does not discuss the further extension of his algorithm to deal with optionality and alternatives within a single grammar rule. In addition, Earley's algorithm differs from the described parsing algorithm in that it uses a top-down prediction step.

Shieber (1984) discusses the extension of a chart parser for direct processing of Immediate Dominance/Linear Precedence (ID/LP) grammars as proposed in the context of Generalized Phrase Structure Grammar (Gazdar et al., 1985). Although Shieber's chart parser is related to the parser described here, the abbreviatory notation that is used in ID/LP grammars is designed especially for abbreviating grammars of natural languages exhibiting relatively free word order and is quite different from the abbreviatory notation supported by our parser.

3 Abbreviated Context-free Grammars

Most context-free grammar notations provide grammar writers with the possibility to use abbreviatory notation to control the structure of grammars and to minimize their size by making the textual grammar representation more compact. Consider the example grammar shown in Figure 1.[2] The grammar notation used is Augmented

```
1. $address = ( $number $street [ (Apt $number) ] $city $state $zip_code [ $country ] ) .
2. $number = ( $digit $digit* ).
3. $street = ( Main Street | Broadway Road | Lincoln Avenue ).
4. $city = ( Chicago | New York | Los Angeles ).
5. $state = ( Illinois | New York | California ).
6. $zip_code = ( $digit $digit $digit $digit $digit ).
7. $country = ( USA | Canada ).
8. $digit = ( 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ).
```

Figure 1. Example ABNF Grammar

Backus Naur Form (ABNF), which has recently been adopted as the standard grammar notation by the World Wide Web Consortium (W3C). In ABNF vertical bars are used to express alternatives, optionality is expressed using squared brackets, repetition is expressed using Kleene star, and parentheses are used to group sequences of right-hand side grammar rule elements. ABNF is closely related to the Extended Backus Naur Form (EBNF; see, for example, Pagan 1981) for context–free grammars. However, in contrast to EBNF, ABNF also allows the grammar writer to specify semantic rules that specify how to construct a semantic representation of an input utterance. The grammar in figure 1 generates strings corresponding to street addresses such as, for example, "1 1 6 9 Lincoln Avenue Apt 3 7 Chicago Illinois 6 0 2 0 1 USA".

Abbreviatory notation is in general very useful to develop and represent grammars. However, many parsing algorithms cannot make direct use of grammars that use such abbreviatory notation, and are therefore forced to expand rules expressed using it. Depending on the size and form of the original abbreviated grammar, expanding its rules by removing abbreviatory notation can be a costly operation where a handful of abbreviated grammar rules can expand to thousands of individual grammar rules. In general, the potential to embed arbitrary numbers of alternative and optional elements makes expansion an exponential process.

4 Direct Parsing Abbreviated Context-free Grammars

The parser we are describing is a modified version of a bi-directional bottom-up chart parser that we developed in the Voice Dialog Systems Lab within Motorola Labs. This previous parser is currently being used as part of the Motorola Mobile Applications Development Kit (MADK), as well as by various research groups inside of Motorola[3]. It supports over a dozen different context-free grammar notations, including JSGF and Nuance's

[2] This example grammar has been simplified for expository reasons, for example, by ignoring the semantics of natural language expressions. Realistic grammars for generating street addresses would be much larger and would contain semantic annotation.

[3] The MADK is a toolkit enabling developers to rapidly create wireless applications that integrate voice and data to run on the Mobile Internet Exchange (MIX) platform (URL: mix.motorola.com/audiences/developers/dev_main.asp).

Grammar Specification Language. [4] Though the original version of the parser was found to be a useful tool for processing natural language input, it suffers from the fact that in order to load grammars into internal memeory dynamically, they need to be expanded. As indicated in the previous section, depending on the size, complexity and amount of abbreviatory notation used in the textual representation of a grammar, the time required for grammar loading can completely overshadow parsing times. This observation led us to develop a parser that can process input utterances directly using abbreviated grammars. Compared to the original version of the Motorola parser and standard bottom-up chart parsers in general (for example, Allen 1994), direct chart parsing using abbreviated context-free grammars involves three types of changes: changes related to the internal representation of the grammar, changes related to the internal representation of edges in the chart, and changes related to how edges are created during parsing.

Grammar Representation The internal representation of the grammar needs to be modified in order to allow for the more complex abbreviated grammar rule format, which consists of a non-terminal associated with a finite state automaton consisting of transitions, each labeled with an identifier referring to a semantic construction rule and a category label. An abbreviated grammar rule is represented as a pair <LHS, RHS>, where LHS specifies the category of the non-terminal node on the left-hand side of the rule and RHS specifies the right-hand side of the rule.

Edge Representation Using the extended parser operations to combine edges (as discussed below) the parser finds paths through these automata representing the right-hand sides of the abbreviated grammar rules. Every time a path through a finite state automaton can be extended, a new edge is stored in the agenda/chart in much the same fashion as with a normal chart parser. However, the form of the edges stored is different: [LHS, StartVtx, EndVtx, RHS, LeftDot, RightDot, BackPtrs]. LHS specifies the category of the non-terminal node on the left-hand side of the rule with respect to which the edge was obtained. StartVtx and EndVtx specify which portion of the input utterance this edge spans. RHS points to the finite state automaton representing the right-hand side of the grammar rule with respect to which the edge was obtained. LeftDot and RightDot specify the states in the RHS corresponding to the fringes of the path through the finite state automaton that has been recognized so far.[5] Finally, in order to construct the parse tree representing the analysis of the input expression found, the edge stores the pointers to its daughter edges in the form of a list of back-pointers (BackPtrs).

Notice that, as a result of the fact that the edges in the chart use the abbreviated grammar representation, the number of edges in the chart can often be reduced. Intuitively understood, each sub-path through the finite state automaton constituting the right-hand side of a grammar rule has to be represented by a separate edge in a standard chart parsing algorithm. If a grammar (rule) does not use abbreviatory notation the number of edges created during parsing cannot be reduced compared to standard chart parsing. However, the number of edges in the chart will never increase as a result of using abbreviated grammar rules.

Parser Operation The main difference between the completion operation used in standard bottom-up chart parsing and our modified completion operation is that completion is no longer a deterministic operation. This is due to the fact that, in general, there can more than one way to combine two edges/an edge and a grammar rule. In addition, due to the fact that the parser is bi-directional there is a left and right variant of the operation an active and a complete edge. Once an edge spanning the complete input utterance has been found, the semantic representation of the input utterance is constructed by executing the relevant semantic construction rules as determined during the parsing process using the semantic identifiers.

5 Evaluation

The parser described in the previous section has been implemented in C++. We have performed an evaluation of this implementation using the original version of the parser as the baseline. The original version is in all relevant respects identical to the current version, except that it does not have the ability to load and parse directly with abbreviated grammars, i.e., it has to expand grammars during the loading process.

The evaluation was performed with respect to three test grammars. All three grammars have been written for specific applications by speech technologists (non-linguists) outside our group. These grammars exhibit little ambiguity. Given the power of abbreviatory notation, the number of rules in a grammar is no longer a good

[4] The semantic construction rules allowed in notations such as, for example, JSGF and ABNF, can exceed context-free generative power. However they all have a syntactic backbone that is strictly context-free.

[5] It is necessary to keep track of both the left and the right fringe of the path through the finite state automaton, because the parser processes the right-hand side of a grammar rule bi-directionally.

indication of the size and complexity of the grammar. A better metric is the [abbreviated rule number]/[expanded rule number] ratio. With respect to this metric the test grammars score as follows: 14/260, 23/889 and 3/5404.

We have tested the performance of the parser with respect to each of these grammars for a set of ten random test utterances of differing length. The average input utterance is 6.4 words.[6] The results of our evaluation are displayed in Table 1. We have averaged the numbers over all test grammars and test utterances involved in the evaluation.

	Direct Loading Parser	*Original Parser*
Grammar Loading Time (sec.)	1.642	6.142
Parse Time (msec.)	6.028	31.625
Size Chart (edges)	139	164
Static Memory (kb)	9,259	46,826
Dynamic Memory (kb)	106	98
Total Footprint (kb)	9,365	46,924

Table 1. Evaluation results

The upper part of the evaluation results in Table 1 shows that the described parser improves on the original parser with respect to grammar loading and parsing times and chart size. Average grammar loading time has decreased by more than 73 percent and average parsing time has decreased by more than 81 percent. We conjecture that the decrease in average parsing time is a consequence of the smaller number of edges that are created during the parsing process, as discussed in the previous section.

The direct loading parser also improves upon the original parser with respect to memory allocation (i.e., footprint size), as a result of the ability to directly process abbreviated grammars. In Table 1, we sub-divide memory allocation into static and dynamic components. The direct loading parser achieves a 79 percent reduction in static memory allocation, while at the same time being only slightly worse (8 percent) in dynamic memory allocation.

6 Conclusion

We have described a bottom-up chart parser for direct processing of input utterances using abbreviated context-free grammars. The presented parser achieves significantly faster grammar loading and parsing times as a result of eliminating the expansion of context-free grammars that are represented using abbreviated notation. As such it is particularly suitable in any application in which grammars must be loaded dynamically. In addition, a significant reduction in the parser's footprint makes it a better candidate to be used on handheld devices with limited memory capacity.

7 References

Allen, J. 1994. *Natural Language Understanding*. Benjamin/Cummings Publishing.

Earley, J. 1970. "An Efficient Context-free Parsing Algorithm". *Communications of the ACM*, 6(8), 451-455.

Gavalda, M. 2000. "SOUP: A Parser for Real-World Spontaneous Speech". *International Workshop on Parsing Technology.*

Gazdar, G., E. Klein, G. Pullum & I. Sag 1985. "Generalized Phrase Structure Grammar" Blackwell.

Pagan, F. 1981. *Formal Specification of Programming Languages: A Panoramic Primer.* Prentice Hall

Shieber, S. 1984. "Direct Parsing of ID/LP Grammars". *Linguistics and Philosophy*, 7:135-154.

Ward, W. 1990. The CMU Air Travel Information Service: Understanding Spontaneous Speech. DARPA Speech and Language Workshop.

[6] This average corresponds closely to what we have observed for actual usages of the applications for which the grammars were written.