AN EMPIRICAL COMPARISON OF GENERALIZED LR TABLES

Marc Lankhorst
Computer Science Department, University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands

ABSTRACT

In this study, table sizes and parsing efficiency of Tomita's algorithm with LR(0), SLR(1), LALR(1) and LR(1) parsing tables are compared on the basis of empirical data. From this comparison, it can be concluded that LALR(1) tables are the best choice regarding parsing time. These tables are about the same size as SLR(1) and noticeably smaller than LR(1) tables. If the size of the tables or the ease of construction is a determining factor, it can be advisable to use LR(0) tables.

As is known from the theory of 'standard' LR parsing, the use of LR(1) tables is expected to result in the best parsing efficiency compared to the other three types. However, this is not the case in Generalized LR parsing. LR(1) turns out to be significantly slower than the other types presented. We give an elegant explanation for this unexpected phenomenon.

1 INTRODUCTION

A well-known method for parsing a certain class of context-free languages is the LR algorithm, introduced by Knuth [1965]. One of the limitations this algorithm imposes, is that the grammars which are used must be unambiguous. In Tomita [1985] an extension to this algorithm is described which can parse almost all non-cyclic context-free languages. This variant, which Tomita calls generalized LR parsing, makes it possible to parse ambiguous natural language on the basis of the efficient LR algorithm.

Parsers based on the LR algorithm use LR parsing tables, consisting of an action and a goto part, which can be constructed directly from the grammar. There are many different types of these LR tables, which vary in space and parsing efficiency. In the following, we will be speaking of LR tables although the generalized LR tables used by Tomita's algorithm differ from standard LR tables in one aspect: Conflicts (multiple entries) in the action part of the table are no longer forbidden.

Since the LR tables are the only part of (generalized) LR-like parsers that are grammar-dependent, the decision which type to use is quite important in the construction of such a parser.

To be able to make a well-founded choice between the different types of generalized LR tables, a comparison on both theoretical and practical aspects is necessary. In this paper, table sizes and parsing efficiency of four of the most widely used types are compared on the basis of empirical data. From this comparison we can draw conclusions considering which table type is the most suitable for use by Tomita's algorithm.

2 LR TABLE TYPES

In this paragraph we discuss the main differences between the four different types of LR tables that we have looked at. These four types are: LR(0), SLR(1), LR(1) and LALR(1). We will not go into the construction methods of these types. These are extensively treated in e.g. Aho, Sethi and Ullman [1986].

LR(0) tables

LR(0), which stands for LR with 0 look-aheads, is the simplest type of LR table. As is stated in its name, this type employs no look-ahead symbol, which means that all reductions are carried out regardless of the next input symbol. In many situations such a reduction will not lead to a correct parse of the input, hence it was invalid and unnecessary work has been done. To prevent this, other methods, which use one or more look-ahead symbols, are invented. Three types which use one look-ahead are discussed below.

The main advantage of LR(0) tables is their relatively small size compared to the other types. This results from the fact that all reductions can be stored in one column instead of across the entire action table.

SLR(1) tables

The first of these types is SLR(1), which means Simple LR with 1 look-ahead. A reduction to a symbol X is carried out only if the next input symbol is an element of FOLLOW(X), the set of terminal symbols that can follow X in any derivation of the start symbol. If the next input symbol is not in this set, it is not possible to derive the start symbol after carrying out the reduction to X, so this reduction is invalid.
SLR(1) tables have the same number of states as LR(0) tables, but they are considerably larger because the reductions have to be stored separately for each valid look-ahead symbol.

**LR(1) tables**

The third type is LR(1), the canonical LR method with 1 look-ahead. In SLR(1) it is still possible that a superfluous reduction to X is carried out when the input symbol can, in general, follow X in a derivation of the start symbol, but not in this particular parse. LR(1) circumvents this problem by carrying more information in a state and splitting states when necessary. In this way some of these invalid reductions can be ruled out. It can be proved that LR(1) makes the best possible use of the look-ahead information, therefore it is called *canonical LR(1)*.

A problem with LR(1) tables is their enormous size. The number of states can be tens of times larger than that of LR(0) and SLR(1) tables for the same grammars.

LR(1) tables have the best parsing efficiency of all single look-ahead types when standard LR parsing is regarded. We shall see that this is not the case in generalized LR parsing. Billot and Lang have found a similar result. They conclude that "sophistication in look-ahead parsing schema (e.g. use of look-ahead) may reduce time and space efficiency instead of improving it" [1989, p. 143]. However, this is not widely known. Nozohoor-Farshi e.g. wrongly states that "using non-optimal LR(1) tables will decrease the number of superfluous reductions in general." [1989, p. 187].

**LALR(1) tables**

Because the large size of the LR(1) tables poses a problem, LALR(1), which stands for Look-Ahead LR(1), was conceived. It forms a compromise between the parsing efficiency of LR(1) and the size of SLR(1) tables. The construction of LALR(1) tables can be regarded as the merging of LR(1) states into a table with the number of states of LR(0) and SLR(1). The parsing efficiency of LALR(1) is better than that of SLR(1) and the size of the tables is comparable to SLR(1).

3 THE EXPERIMENT

The objective of the experiment was to gain insight in the aptness for Tomita's algorithm of the four different types of LR tables described above.

**Grammars**

To conduct the experiment, we used three grammars, which are all given by Tomita [1985]. Grammar 3, formed by 225 productions, can be used on both the first and the second set of sentences (see below). Grammars 1 and 2, consisting of 8 and 44 productions respectively, can only be applied on the second sentence set.

**Sentences**

The input sentences are also derived from Tomita [1985]. The first sentence set consists of 40 'real life' sentences, most of which are taken from actual publications.

The second set is made more systematically. The n-th sentence (1 <= n <= 8) is obtained by the following schema:

```
noun verb det noun (prep det noun)^n-1
```

Tomita [1986, p. 81-82, 153] assumes that these sentences have the same ambiguity when parsed with all three grammars. This is not the case! Because of the absence of a recursive rule of the form 'NP -> NP PP' in the second grammar, this grammar allows less deep ambiguities than the first and third.

**Method**

To obtain the test results we have used an implementation of Tomita's algorithm which was executed on a Sun Microsystems SPARCstation1. Every input sentence was parsed five times in a row and this process was repeated four times. The average parsing time was computed from these 20 results to minimize potential inaccuracies.

Apart from the parsing time, the sizes of the graph-structured stack Γ and the packed forest T of Tomita's algorithm were also determined. These sizes are a measure for the density of the packing that was applied. Without packing, Γ is exactly twice as large as T. This can be deduced from the formal description of Tomita's algorithm.

In LR(0) tables all reductions to be carried out can be found in one column of the action table. To determine whether this is an advantage, we have also regarded LR(0) tables in which the reductions were filled in across the entire action table.

The sizes of the different types of tables for the three grammars can be found in figure 1. From this figure, which has a logarithmic scale, one can observe that the "normal" LR(0) tables are the smallest and the LR(1)
Figure 1. Table size against grammar size for grammars 1, 2 and 3

Figure 2. Parsing time against ambiguity for grammar 1 and sentence set 2
tables are by far the largest. The LR(1) tables for grammar 3 are several megalobytes in size and could not be computed due to a lack of memory. Therefore this size is estimated.

4 RESULTS

In figure 2 the parsing times of the sentence set 2 combined with grammar 1 are given for the different table types. This figure shows a small but suspicious detail: The LR(1) parsing times are somewhat slower than the SLR(1) and LALR(1) times, whereas one would expect LR(1) to be faster (see 2.3). But the differences are so small that they could be attributed to measuring inaccuracies.

However, figure 3 shows the same difference on a much larger scale. This cannot be some inaccuracy but requires a better explanation.

Explanation

The unexpected phenomenon described above can be explained as follows.

LR(1) tables contain in general (many) more different states than LR(0), SLR(1) and LALR(1) tables. This means that in a non-deterministic LR(1) parse of an ambiguous sentence, different states occur which would have coincided if LR(0), SLR(1) or LALR(1) tables were used.

Figure 3. Parsing time against ambiguity for grammar 2 and sentence set 2

Figure 4. Graph-structured stack during LALR(1) parsing

Figure 5. Graph-structured stack during LR(1) parsing
Tomita's algorithm merges top vertices with the same state in the graph-structured stack. In the case of LR(1) less merging is possible due to the larger number of different states. This results in more top vertices than in the other three cases. Because reductions and shift actions are carried out on these top vertices, more work has to be done and longer parsing times are obtained.

An example of this can be seen in figures 4 and 5. Figure 4 shows the graph-structured stack during the LALR(1) parsing of a sentence from set 2 with grammar 2. Figure 5 shows the stack during the LR(1) parsing of the same sentence. Both figures show the situation immediately after shifting a "prop symbol onto the stack. In figure 4, the top vertices have just been merged. In the LR(1) case of figure 5, this is not possible because the top states are different. The next actions have to be carried out thrice in the case of LR(1), but only once in the LALR(1) case. This leads to a major difference in parsing time.

Packing is less efficient in the case of LR(1) than in the other cases. The stack sizes of the different types relative to LR(0) are shown in figure 6. From this figure, it can be seen that the LR(1) packing grows less and less dense compared to LR(0) when the ambiguity of the input is raised.

One would expect LR(1) to be the fastest method when no packing is used. As can be seen from figure 7, this is indeed the case. This fact forms a strong support for our explanation.

More results

Some more results are shown in figures 8 and 9. From both these figures, it is apparent that SLR(1) is significantly faster than LR(0) and LALR(1) is the fastest method. Also one can see that the advantage of 'normal' LR(0) over the 'filled in' LR(0) is not very large.

5 CONCLUSIONS

The first conclusion that can be drawn from the above is that LR(1) tables are unsuitable for the generalized LR parsing algorithm of Tomita. These tables are much larger than the other three types presented and result in significantly longer parsing times. This conclusion is not in accordance with most literature on standard LR parsing, which states that LR(1) is the most efficient single look-ahead LR type. This is definitely not the case in generalized LR parsing.

Secondly, we can conclude that from the four types presented, LALR(1) is the best choice if maximum parsing efficiency is required. The use of these tables can result in a 5-10% improvement in parsing time compared to LR(0). Also a gain in space efficiency of the same order of magnitude can be achieved.

If the size of the parsing tables is a constraint, LR(0) tables are advisable. These are several times smaller than the other types. Another advantage of LR(0) tables is their ease of construction.

Acknowledgement

This research was done during my stay with the Language and Speech Group of Océ Nederland BV, Venlo, The Netherlands. I would like to thank Rob Heemels, Eddy van Vliembergen, Lou Cremers, Erik Hommersom and all my other colleagues at Océ for their help and support.

REFERENCES


Figure 6. Size of the graph-structured stack relative to LR(0)

Figure 7. Parsing time against ambiguity for grammar 2 and sentence set 2 without packing
Figure 8. Parsing time against ambiguity for grammar 3 and sentence set 1

Figure 9. Parsing time against ambiguity for grammar 3 and sentence set 2