The Computational Complexity of Tomita's Algorithm

Mark Johnson

April 26, 1989

1 Introduction

The Tomita parsing algorithm adapts Knuth's (1967) well-known parsing algorithm for LR(k) grammars to non-LR grammars, including ambiguous grammars. Knuth's algorithm is provably efficient: it requires at most $O(n|G|)$ units of time, where $|G|$ is the size of (i.e. the number of symbols in) G and n is the length of the string to be parsed. This is often significantly better than the $O(n^3|G|^2)$ worst case time required by standard parsing algorithms such as the Earley algorithm. Since the Tomita algorithm is closely related to Knuth's algorithm, one might expect that it too is provably more efficient than the Earley algorithm, especially as actual computational implementations of Tomita's algorithm outperform implementations of the Earley algorithm (Tomita 1986, 1987).

This paper shows that this is not the case. Two main results are presented in this paper. First, for any m there is a grammar L_m such that Tomita's algorithm performs $\Omega(n^m)$ operations to parse a string of length n. Second, there is a sequence of grammars G_m such that Tomita's algorithm performs $\Omega(nc^{|G_m|})$ operations to parse a string of length n. Thus it is not the case that the Tomita algorithm is always more efficient than Earley's algorithm; rather there are grammars for which it is exponentially slower. This result is forshadowed in Tomita (1986, p. 72), where the author remarks that Tomita's algorithm can require time proportional to more than the cube of the input length. The result showing that the Tomita parser can require time proportional to an exponential function of the grammar size is new, as far as I can tell.

2 The Tomita Parsing Algorithm

This section briefly describes the relevant aspects of the Tomita parsing algorithm: for further details see Tomita (1986). Familiarity with Knuth's LR parsing algorithm is presumed: see the original article by Knuth (1967), Aho and Ullman (1972), or Aho, Sethi and Ullman (1986) for details.

The Tomita algorithm and Knuth's LR parsing algorithm on which it is based are both shift-reduce parsing algorithms, and both use the same LR automaton to determine the parsing actions to be performed. The LR automaton is not always deterministic: for example, if the grammar is ambiguous then at some point in the analysis of an ambiguous string two different parsing actions must be possible that lead to the two distinct analyses of that string. Knuth's algorithm is only defined for grammars for which the parsing automaton is deterministic: these are called the LR(k) grammars, where k is the length of the lookahead strings. Tomita's algorithm extends Knuth's to deal with non-deterministic LR automata.

Tomita's algorithm in effect simulates non-determinism by computing *all* of the LR stacks that result from each of the actions of a non-deterministic LR automaton state. Tomita's algorithm mitigates the cost of this non-determinism by representing the set of all the LR stacks possible at a given point of the parse as a multiply-rooted directed acyclic graph called a *graph-structured stack*, which is very similiar to a parsing chart (Tomita 1988). Each node of this graph represents an LR state of one or more of the LR stacks, with the root nodes representing the top states of LR parse stacks. The graph contains exactly one leaf node (i.e. a node with no successors). This leaf node represents the start state of the LR automata (since this is the bottom element of all LR parse stacks), and each maximal path through the graph (i.e. from a root to the leaf) represents an LR parse stack.

As each item in the input string is read all of the parsing actions called for by the top state of each LR stack are performed, resulting in a new set of LR stacks. Because of the way in which the set of LR stacks are represented, Tomita's algorithm avoids the need to copy the each LR stack in its entirity at non-deterministic LR automaton states; rather the top elements of the two (or more) new stacks are represented nodes whose successors are the nodes that represent the LR stack elements they have in common. Similiarly, if the same LR state appears as the top element of two or more new stacks then these elements are represented by a single node whose immediate successors are the set of nodes that represent the other elements of these LR stacks. This "merging" of identical top elements of distinct LR stacks allows Tomita's algorithm to avoid duplicating the same computation in different contexts.

Finally, Tomita employs a *packed forest* representation of the parse trees in order to avoid enumerating these trees, the number of which can grow exponentially as a function of input length. In this representation there is at most one node of a given category at any string location (i.e. a pair of beginning and ending string positions), so the number of nodes in such a packed forest is at most proportional to the square of the input length. Each node is associated with a set of sequences of daughter nodes where each sequence represents one possible expansion of the node; thus the trees represented can easily be "read off" the packed forest representation.

3 Complexity as a Function of Input Length

The rest of this paper shows the complexity results claimed above. This section describes a sequence of grammars L_m such that on sufficiently long inputs the Tomita algorithm performs more than $\Omega(n^m)$ operations to parse an input of length n. This result follows from properties of the packed forest representation alone, so it applies to *any* algorithm that constructs packed forest representations of parse trees.

Consider the sequence of grammars L_m for $m > 0$ defined in (1), where S^{m+2} abbreviates a sequence of S's of length $m + 2$.

$$\begin{array}{l} S \rightarrow a \\ S \rightarrow SS \\ S \rightarrow S^{m+2} \end{array} \qquad (1)$$

All of these grammars generate the same language, namely the set of strings a^+. Consider the input string a^{n+2} for $n > m$. By virtue of the first two rules in (1) any non-empty string location can be analyzed as an S. Thus the number of different sequences of daughter nodes of the matrix or top-most S node licensed by the third rule in (1) is $\binom{n}{m+1}$ the number of ways of choosing different right string positions of the top-most S node's first $m + 1$ daughters. Since $\binom{n}{m+1}$ is a polynomial in n of order $m + 1$, it is bounded below by cn^m for some $c > 0$ and sufficiently large n, i.e. $\binom{n}{m+1} = \Omega(n^m)$. Since any algorithm which uses the packed forest representation, such as Tomita's algorithm, requires the construction of these sequences of daughter nodes, any such algorithm must perform $\Omega(n^m)$ operations.

Finally, it should be noted that this result assumes that the sequences of daughter nodes are completely enumerated. It might be possible these sequences could themselves be "packed" in such a fashion that avoids their enumeration, possibly allowing the packed forest representations to be constructed in polynomial time.

4 Complexity as a Function of Grammar Size

This section shows that there are some grammars such that the total number of operations performed by the Tomita algorithm is an exponential function of the size of the grammar.

The amount of work involved in processing a single input item is proportional to the number of distinct top states of the set of LR stacks corresponding to the different non-deterministic analyses of the portion of the input string shown so far. By exhibiting a sequence of grammars in which the number of such states is an exponential function of the size of the grammar we show that the total number of operations performed by the Tomita algorithm can be at least exponentially related to the size of the grammar.

Consider the sequence of grammars G_m for $m > 0$ defined in (2).

$$\begin{array}{ll} S \rightarrow A_i & 0 \leq i \leq m \\ A_i \rightarrow B_j A_i & 0 \leq i,j \leq m, i \neq j \\ A_i \rightarrow B_j & 0 \leq i,j \leq m, i \neq j \\ B_j \rightarrow a & 0 \leq j \leq m \end{array} \qquad (2)$$

All of the grammars G_m generate the same language, namely the set of strings a^+. Since these grammars are ambiguous they are not LR(k) for any k.

Consider the behaviour of a non-deterministic LR parser for the grammar G_m on an input string a^n where $n > m$. The items of the start state are shown in (3).

$$\begin{bmatrix} S \rightarrow \cdot A_i \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{bmatrix} \quad 0 \leq i,j \leq m, i \neq j \qquad (3)$$

The parser shifts over the first input symbol a to the state shown in (4).

$$[B_j \rightarrow a \cdot] \quad 0 \leq j \leq m \qquad (4)$$

This is a non-deterministic state, since all of the m reductions $B_j \rightarrow a$ are possible parsing actions from this state. Suppose that the reduction to B_{k_1} is chosen. The state that results from the reduction to B_{k_1} is shown in (5). There are m such states.

$$\begin{bmatrix} A_i \rightarrow B_{k_1} \cdot A_i \\ A_i \rightarrow B_{k_1} \cdot \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{bmatrix} \quad 0 \leq i,j \leq m, i \neq j, k_1 \qquad (5)$$

After shifting over the next input symbol the parser again reaches the same ambiguous state as before, namely the state shown in (4). Suppose the reduction to B_{k_2} is chosen. If $B_{k_1} = B_{k_2}$ then the resulting state is the one shown in (5). On the other hand, if $B_{k_1} \neq B_{k_2}$ then the resulting state is as shown in (6). There are $m(m-1)/2$ distinct states of the form shown in (6), so after reducing B_{k_2} there will be $m(m+1)/2$ distinct LR states in all.

$$\begin{bmatrix} A_i \rightarrow B_{k_1} \cdot A_i \\ A_i \rightarrow B_{k_1} \cdot \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{bmatrix} \quad 0 \leq i,j \leq m, i \neq j, k_1, k_2 \qquad (6)$$

It is not hard to see that after $n \geq m$ input symbols have been read and reduced to $B_{k_1} \ldots B_{k_n}$ respectively the resulting state will be as shown in (7).

$$\begin{bmatrix} A_i \rightarrow B_{k_n} \cdot A_i \\ A_i \rightarrow B_{k_n} \cdot \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{bmatrix} \qquad 0 \leq i, j \leq m, i \neq j, k_1 \ldots k_n \tag{7}$$

Since there are $2^m - 1$ distinct such states, the Tomita parser must perform at least $2^m - 1$ computations per input item after the first m items have been read. Since $|G_m| = 5m^2 - m = O(m^2)$, the ratio of the average number of computations per input item for a sufficiently long string to grammar size is $\Omega(2^m/m^2) = \Omega(c^m)$ for some $c > 1$. Thus the total number of operations performed by the parser is $\Omega(c^{|G_m|})$, exponential function of grammar size.

5 Conclusion

The results just demonstrated do not show that Tomita's algorithm is always slower than polynomially bounded algorithms such as Earley's, in fact in practice it is significantly faster than Earley's algorithm (Tomita 1986). On the other hand, the results presented here show that this superior performance is not just a property of the algorithm alone, but also depend on properties of the grammars (and possibly the inputs) used. It would be interesting to identify the properties that are required for efficient functioning of Tomita's algorithm.

Second, it might possible to modify Tomita's algorithm so that it provably requires at most polynomial time. For example, requiring all grammars used by the algorithm to be in Chomsky Normal Form would prohibit the grammars used to show that Tomita's algorithm does not always run in polynomial time. Whether this restriction would ensure polynomial time behaviour with respect to input length is an open question (note that the grammars used to show the exponential complexity with respect to grammar size are already in Chomsky Normal Form).

Finally, the non-polynomial behaviour of Tomita's algorithm with respect to input length followed from the properties of the packed forest representation of parse trees, so it follows that any algorithm which uses packed forest representations will also exhibit non-polynomial behaviour.

6 Bibliography

Aho and Ullman (1972) *The Theory of Parsing, Translation and Compiling*, vol. 1, Prentice Hall, New Jersey.

Aho, Sethi and Ullman (1986) *Compilers: Principles, Techniques and Tools*, Addison-Wesley, Reading, Mass.

Tomita (1986) *Efficient Parsing for Natural Language*, Kluwer, Boston, Mass.

Tomita (1987) "An Efficient Augmented-Context-Free Parsing Algorithm", *Computational Linguistics*, vol. 13, 31-46.

Tomita (1988) "Graph-Structured Stack and Natural Language Parsing", in *The Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, SUNY Buffalo, New York.