A Quantitative Comparison Between an LR Parser and an ATN Interpreter

Chao-Lin Liu and Keh-Yih Su

Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.

Abstract

We have known that a bottom-up parser will parse an English sentence faster than a top-down parser does. Nevertheless, there is still no report on how much faster it is. To quantitatively compare these two parsers, an LR parser and an ATN interpreter are built and are used as bottom-up and top-down parsers respectively. These two parsers are currently widely used in the computational linguistics community. From the tests we have proceeded, we find that the average parsing speed of the LR parser is tens of times faster than that of the ATN interpreter.

Introduction

LR and ATN parsers are currently two of the most widely used parsers for natural languages processing. In this paper, we will compare their parsing speed quantitatively. Since the speed is the main topic of concern, not the grammar formalism, the grammars used in both parsers are kept the same through the whole comparison.

In the design of translators of programming languages, bottom-up translation has been one of the well-known strategies. Among the parsers based on bottom-up translation techniques, LR parser [AHO 86] is the most popular one. A lot of translators, like C compiler and YACC in UNIX, take advantage of an LR parser to translate the input files.

Although the LR parser has been a good parser for the translation of programming languages. It can not be used to parse natural languages, like English, directly. The point is that the traditional LR parser does not accept grammars that are ambiguous, while the grammars for natural languages are usually ambiguous. But, due to its success in parsing programming languages, researchers [SU 85, TOMI 85, HSU 86] augmented it to accept ambiguous grammars and give it the power to handle linguistic problems, so that similar techniques can be used to parse natural languages.

On the other hand, another famous formalism called Augmented Transition Networks (ATN) [WOOD 70] based on top-down translation techniques were designed to parse natural languages. ATN formalism is derived from the Recursive Transition Networks (RTN) which is in return derived from the Basic Transition Networks (BTN). Both RTN and BTN are not adequate to parse the natural languages due to some shortcomings of them [BATE 78]. In addition to the intrinsic features of RTN and BTN, the arcs of an ATN are associated with actions and conditional checks. This augmentation gives an ATN the computational power of a Turing machine [FINI 83].

The major difference between ATN and augmented LR parsers is the way by which they form a larger node from their constituents. An ATN parser is essentially a top-down parser which adopts hypothesis-driven paradigm [MARC 80]. On the other hand, the LR parsers are bottom-up parsers, and thus are data-driven. Due to this, ATN is expected to parse English slower than LR parser does. However, we are curious about how much slower the ATN is when compared with the LR parser. In this paper, we will describe the tests we have conducted on our LR parser and ATN interpreter and compare their parsing speed quantitatively.

The Environment

Both parsers under testing are currently implemented on the SUN 3/160C workstation. The LR parser is written in C language while the ATN interpreter is written in SUN Common Lisp Version 2.0. Before the comparison, the LR parser is compiled into the executable object codes of the SUN 3/160C, and the ATN interpreter is compiled into binary codes that are executable by the Lisp interpreter of the SUN 3/160C. A simple Benchmark shows that the compiled Lisp codes have almost identical execution speed to that of a compiled C program, therefore, we shall ignore the possible speed-up effects introduced by the differences between these two languages. The tests are proceeded on the SUN 3/160C workstation which, declared by the SUN microsystems, INC., is a 2 MIPS machine.

The reason that we implemented an ATN interpreter instead of an ATN compiler is that the former is far easier to implement than the latter is. According to Finin's experience [FINI 83], the parsing speed of the former is about 5 times slower than the latter's. Knowing about this, we do not bother to implement an ATN compiler to compare an ATN and LR parser. On the contrary, we only have to build an ATN interpreter to compare them. Besides, the reason to choose Common Lisp instead of C as the implementation language of ATN interpreter is that ATN interpreters are typically implemented in Lisp. As described in the last paragraph, the compiled Lisp has almost identical execution time to that of a C program, hence, the differences in these two languages should not have significant effects on the comparison. Finally, the arcs and actions of the ATN interpreter are implemented as suggested in [BATE 78, CHRI 83, FINI 83].

Under this environment, we will compare the parsing speeds of a typical LR parser and a typical ATN interpreter.

The Parameters Controlled and Compared

To compare these two parsers, we must use the same grammar, the same lexicon, and the same set of sentences as test data. The grammar we used is the second test grammar in appendix F of [TOMI 85], as shown below:

```
   S  :  NP VP PP PP ;
      :  NP VP PP ;
      :  PP NP VP ;
      :  NP VP ;
      :  S conj S .
  NP  :  NP conj NP ;
      :  NP1 that S ;
      :  NP1 S ;
      :  NP1 .
  PP  :  PP conj PP ;
      :  prep NP .
 NP1  :  ADJM NP0 PP PP ;
      :  ADJM NP0 PP ;
      :  ADJM NP0 ;
      :  NP0 PP ;
      :  NP0 ;
      :  NP0 PP PP .
ADJM  :  adj ;
      :  adj ADJM ;
      :  ADVM adj ;
      :  ADJM conj ADJM .
  NP0  :  NM ;
       :  ADJM NM ;
       :  art NM ;
       :  art ADJM NM .
   NM  :  n ;
       :  n NM .
 ADVM  :  adv ADVM ;
       :  adv ;
       :  ADVM conj ADVM .
   VP  :  VC NP ;
       :  VP conj VP ;
       :  VC .
   VC  :  aux v ;
          v .
```

In the grammar listed above, the capital symbols represent the non-terminal symbols while the other symbols represent terminal symbols. Written in the form of ATN grammar, this grammar contains 45 states and 67 arcs. And, the lexicon we used is a small one which currently contains only 61 words.

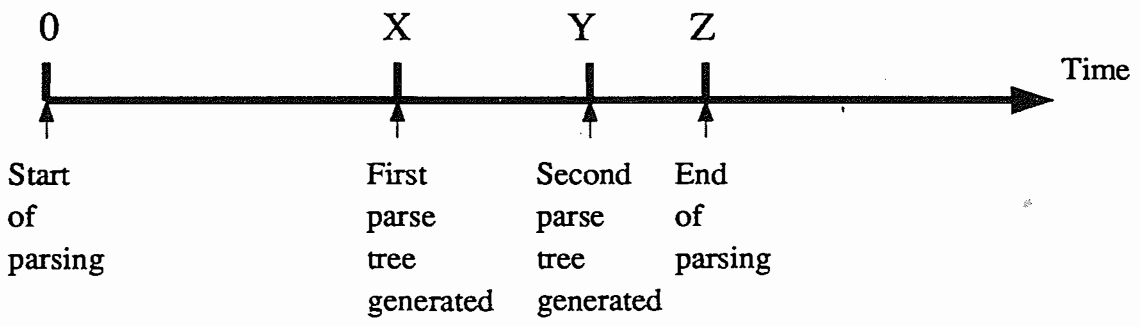
39 sentences are used in the tests. They can be divided into several groups according to their word counts and number of ambiguities. These two parameters will affect the parsing time needed to parse the sentences. In the next section, tests will be taken to see how they affect the parsing speed of the parsers.

Two parsing times will be compared in the tests, they are :

(1) First Parse tree Time (FPT) : the period of time from the beginning of parsing to the time the first legal parse tree is generated.

(2) Average Parse tree Time (APT) : the period of time from the beginning of parsing to the time the last parse tree is generated divided by the number of parse trees generated.

For example: Suppose that sentence A has two ambiguities, and when A is parsed the timings are recorded as shown below:



then FPT = X and APT = Y/2.

Tests and Discussions

In this section , we will call the ATN parser 'ATNP' and the LR parser 'LRP' for short. In each of the tests, the test sentences and the results are discussed.

Test 1: Test for sentences with the same number of ambiguities but different number of words.

In this test, the number of ambiguities of the sentences is the same but the number of words of the sentences is different. Five sentences are used in this test, as shown below. The sentences listed below have 5, 10, 15, 20, and 25 words respectively, and all of them have 2 syntactic ambiguities with respect to the test grammar. Because the vocabulary for the test version of the parsers is limited, these sentences may be semantically nonsense.

(1) Cruelly cruelly cruel computer loves.
(2) I require the beautiful beautiful beautiful computer in the apple.
(3) This good maintenance will show where shows the tree at the station where in plane.
(4) The angry angry angry woman that the good good good man loves kills the green green green apple in plane.
(5) The angrily angrily angrily angry woman that the man loves kills the angrily angrily angrily angry woman that the man loves in the green apple.

The results are shown in Figure 1, Figure 2, and Figure 3 on the next pages. Figure 1 and Figure 2 are the timings of the ATNP and LRP. The X-coordinates of both figures represent the number of the parse tree generated. The '0', on the X-coordinate, represents the beginning of parsing. The 'END', on the X-coordinate, represents the end of parsing. The Y-coordinates of both figures represent the CPU time[1] taken by the parsers to parse the sentences. Each line in the figures represents the timings for one sentence. For examples, as shown in Figure 1, it takes ATNP about 8.5 seconds CPU time to generate the first parse tree for sentence (4) and 9.7 seconds CPU time to generate both parse trees for sentence (4). The curves for ATNP have larger initial slopes, but the slopes become flatten after the first parse tree is acquired. The curves for the LRP, on the other hand, does not show this feature. That is, most of the parsing time taken by the ATNP to parse a sentence is used to find the first parse tree while the LRP is not.

[1] CPU time = User CPU time + System CPU time of process, and henceforth.

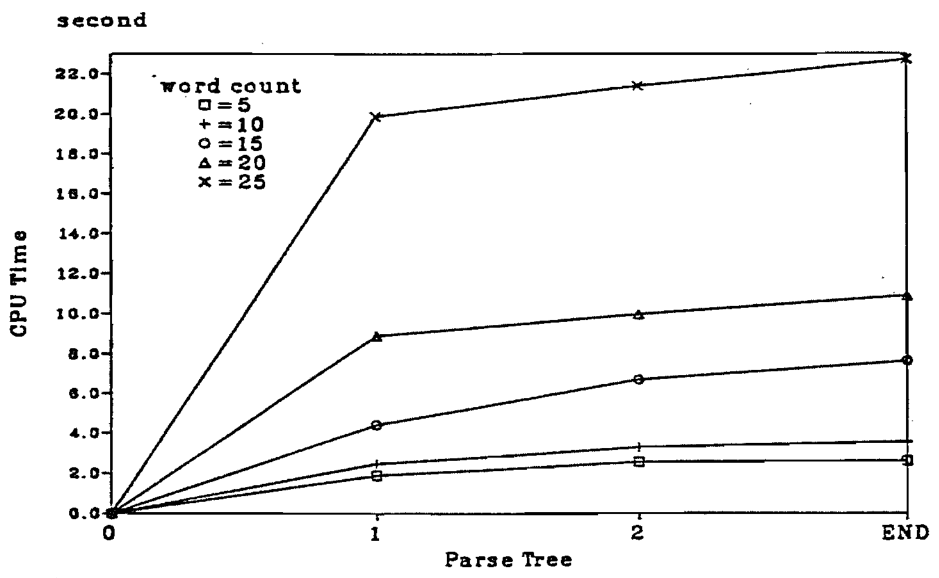


Figure 1 : ATNP timings for sentences with the same number of ambiguities

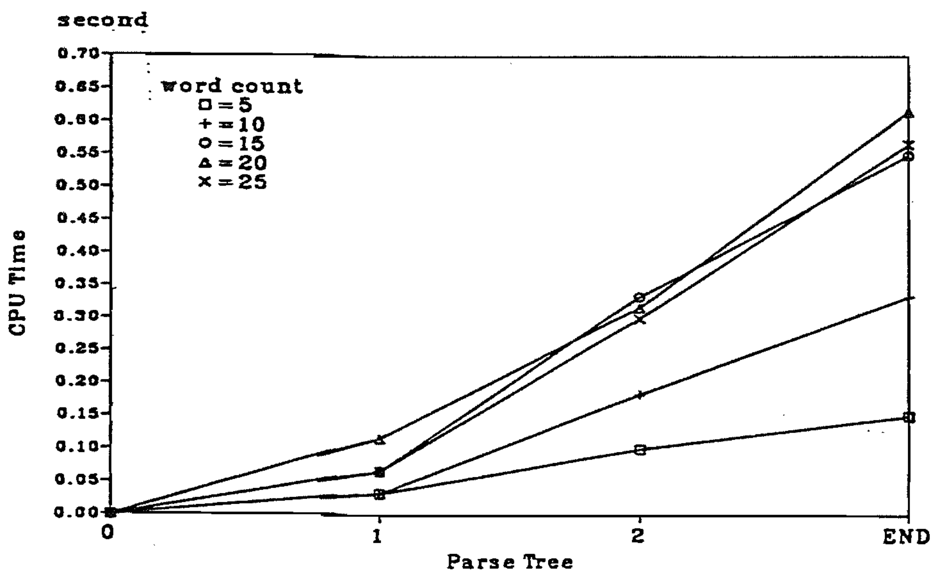


Figure 2 : LRP timings for sentences with the same number of ambiguities

In Figure 3 on the next page, the X-coordinate represents the word count of the sentences and the Y-coordinate represents the time ratio of the ATNP and LRP. The line with square markers is the time ratio of the times taken by the ATNP and LRP to generate the *first* parse tree of the sentences, that is, the ratio of FPT of ATNP and LRP. Similarly, the line with plus markers is the time ratio of the times taken by the ATNP and LRP to generate *both* parse trees of the sentences. Finally, the line with circle markers is the time ratio of the times taken by the whole parsing process. From this figure, we find a number of interesting facts:

(1) For FPT's, the ATNP is slower than the LRP for a factor of at least 55.

(2) To generate both parse trees, the ATNP is slower than the LRP for a factor of at least 20.

(3) From point (1) and (2) above, the speed-up or reduction in time becomes less drastic after the first parse tree is acquired. This is because most information acquired in the analysis of the first parse tree is retained and is available to successive analyses during backtracking.

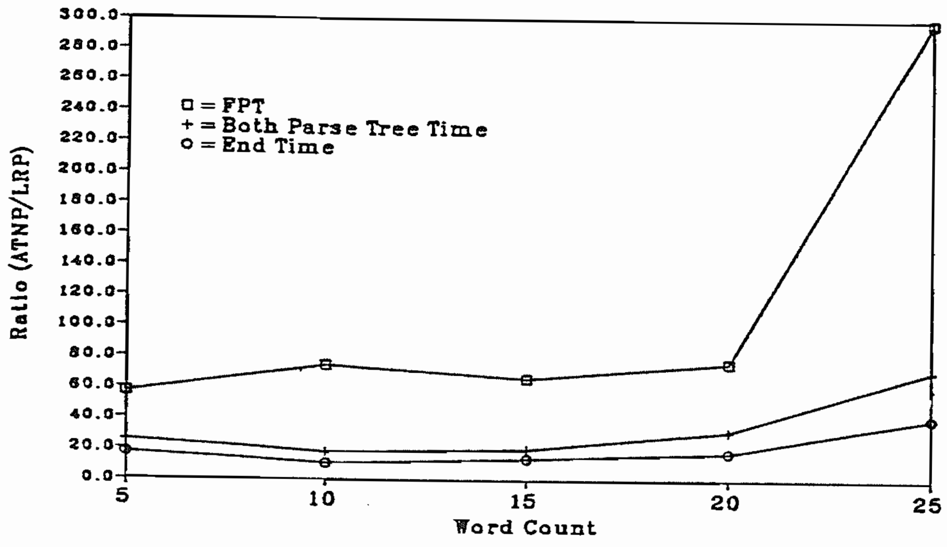


Figure 3 : Ratios of timings (ATNP/LRP)

The first observation suggests that one can benefit from an LR parser significantly if the first parse tree is the one to be used as output. This is usually the case for a system with well-defined scoring mechanism.

Test 2: Test for sentences with different number of ambiguities.

In this test, 34 sentences are used. All of them have 10 or 11 words. And the number of sentences having 1, 2, 4, 5, 6, 8, 10, and 12 ambiguities are 9, 5, 7, 3, 2, 5, 2, and 1 respectively. To save space, we will not list the sentences here.

The results are shown in Figure 4 and table 1 on the next page. In Figure 4, the Y-coordinate represents time ratio of the ATNP and the LRP, and the X-coordinate represents the number of ambiguities for the test sentences. The line with square markers is for FPT while the line with plus markers is for APT.

All timings used to calculate the ratios in Figure 4 are the average ones. In other words, the FPT's or APT's for a given number of ambiguities, say 2 ambiguities, are averaged before their ratio are computed. For example, from Figure 4, we know that the ratio of the average FPT is about 60 for sentences having 2 ambiguities. From Figure 4, we also find that :

(1) The more ambiguities the sentences have, the larger the FPT ratio is. And the ratio is at least 35.

(2) The more ambiguities the sentences have, the lesser the APT ratio is. And the ratio is at most 35.

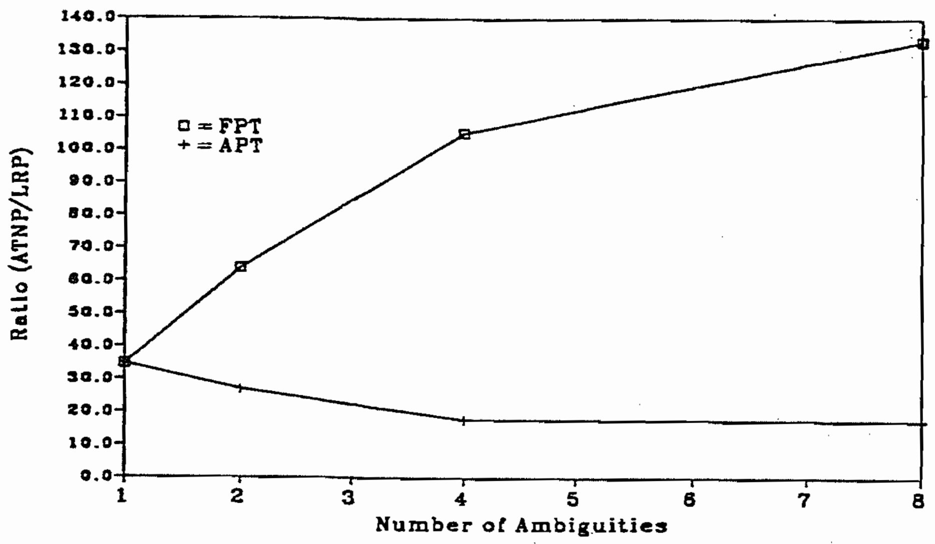


Figure 4 : Ratios of FPT and APT (ATNP/LRP)

#AMBIGUITY	PARSER / STATISTICS	ATNP FPT	ATNP APT	LRP FPT	LRP APT
1	STD	0.51	0.51	0.020	0.020
	AVG	1.99	1.99	0.057	0.057
	STD/AVG	0.26	0.26	0.34	0.34
2	STD	3.62	2.45	0.065	0.093
	AVG	6.18	4.04	0.097	0.150
	STD/AVG	0.59	0.61	0.67	0.62
4	STD	1.71	0.51	0.015	0.007
	AVG	4.51	2.15	0.043	0.121
	STD/AVG	0.38	0.24	0.35	0.06
8	STD	2.05	0.85	0.032	0.020
	AVG	8.45	2.59	0.063	0.149
	STD/AVG	0.24	0.33	0.51	0.14
TOTAL	STD	3.77	1.31	0.037	0.056
	AVG	5.67	2.60	0.063	0.124
	STD/AVG	0.67	0.50	0.59	0.45

Table 1 : Statistics of the second test. (Unit : second CPU time)

In Table 1, we list some statistics about the FPT's and APT's in this test. The number in the leftmost column represents the number of ambiguities of the test sentences. The row labeled 'TOTAL' represents all of the 34 test sentences. In each row, 'STD' and 'AVG' stand for the *standard deviation* and *average* value, and 'STD/AVG' stands for the ratio of standard deviation to the average. Using statistics in this table, we can derive some other statistics.

For example, to find the difference of the average FPT's of these 34 sentences. From this table, we find that the average FPT of the ATNP is 5.67 second CPU time and LRP is 0.063 second CPU time. From this table, we find that :

(1) For the 34 sentences, the FPT ratio of ATNP and LRP is 90.43, that is, in average, the ATNP is 90 times slower than the LRP in generating the first parse tree of the sentences.
(2) For the 34 sentences, the APT ratio of ATNP and LRP is 20.99, that is, in average, the ATNP is 21 times slower than the LRP in generating all the parse trees of sentences.
(3) For the 34 sentences, the 'STD/AVG' of FPT of both ATNP and LRP are about 0.60. This means that the variance, in the sense of percentage changes, of the time needed to generate the first parse tree by both parsers is about the same.

From the tests discussed above, we see that ATNP is slower than the LRP. We think that the major reasons for this phenomenon are :

(1) Top-down parsing is intrinsically inferior to bottom-up for natural languages like English. In other words, the characteristics of English makes it more desirable to use a data-driven parser instead of a hypothesis-driven one.
(2) The state transition of LRP is explicitly coded in the parsing table while the ATNP is not. So, in the course of parsing, the LRP does not have to recompute the next possible state transition while the ATNP does have to.
(3) The Lisp in itself is slower than the C language. Although it is easier to implement an ATN interpreter in Lisp. This factor, however, is consider less significant when both parser/interpreter are compiled, because our preliminary Benchmark shows that they have almost identical execution time for the Benchmark.

Furthermore, using simple toy grammar such as the one we used in the tests, the ATN interpreter has been so slow in the parsing speed compared with the LR parser . We expect that their difference in parsing speed will be even greater with more complicated grammar, say a grammar for a machine translation system.

Conclusion

In this paper, we have shown that the average parsing speed of a typical ATN interpreter is tens of times slower than that of the LR parser. And we have discussed briefly why the ATN interpreter is slower than the LR parser. Although an ATN parser can be easily constructed, it may not be practical, as far as parsing efficiency is considered, in constructing a large system which has a complicated grammar.

Acknowledgement

We are grateful to Professor I-Pen Lin at the Department of Computer Science and Information Engineering of National Taiwan University for his helpful recommendation and suggestions on the implementation of the ATN interpreter.

References

[AHO 86] Aho, A. V., R. Sethi and J. D. Ullman, *Compilers : Principles, Techniques and Tools*, Addison-Wesley Publishing Company, Reading, MA, 1986.

[BATE 78] Bates, M., "The Theory and Practice of Augmented Transition Network Grammars," in *Natural Language Communication with Computers*, pp. 191–259, Leonard Bolc (ed.), Springer Verlag, 1978.

[CHRI 83] Christaller, T., "An ATN Programming Environment," in Bolc, L. (ed.) *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*, pp. 71–148, Springer Verlag, 1983.

[FINI 83] Finin, T. W., "The Planes Interpreter and Compiler for Augmented Transition Network Grammars," in Bolc, L. (ed.) *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*, pp. 1–69, Springer Verlag, 1983.

[HSU 86] Hsu, H.-H. and K.-Y. Su, "A Bottom-up Parser in the Machine Translation System with the Essence of ATN," *Proc. Int. Computer Symposium (ICS) 1986*, Vol. 1, pp. 166–173, Tainan, Taiwan, R.O.C., 1986.

[MARC 80] Marcus, M.P., *A Theory of Syntactic Recognition for Natural Language*, MIT Press, London, U.K., 1980.

[SU 85] Su, K.-Y., H.-H. Hsu, and M.-L. Hsiao, "Development of an English-to-Chinese Machine Translation System," *Proc. Natl. Computer Symposium (NCS) 1985*, Vol. 2, pp. 997–1001, Kao-Hsiung, Tainwan, R.O.C., 1985.

[TOMI 85] Tomita, M., *An Efficient Context-Free Parsing Algorithm for Natural Languages and its Applications*, Ph. D. Dissertation, Carnegie-Mellon University, 1985.

[WOOD 70] Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Communication of ACM*, pp. 591–606, October 1970.