Parsing Noisy Sentences

Hiroaki SAITO

Center for Machine Translation
Carnegie Mellon University
Pittsburgh, PA 15213, USA
and
ATR Interpreting Telephony Research Laboratories
Twin 21 MID Tower, 2-1-61 Shiromi
Higashi ku, Osaka 540, Japan

Masaru TOMITA

Center for Machine Translation
Carnegie Mellon University
Pittsburgh, PA 15213, USA

Abstract

This paper describes a method to parse and understand a "noisy" sentence that possibly includes errors caused by a speech recognition device. Our parser is connected to a speech recognition device which takes a continuously spoken sentence in Japanese and produces a sequence of phonemes. The output sequence of phonemes can quite possibly include errors: altered phonemes, extra phonemes and missing phonemes. The task is to parse the noisy phoneme sequence and understand the meaning of the original input sentence, given an augmented context-free grammar whose terminal symbols are phonemes. A very efficient parsing method is required, as the task's search space is much larger than that of parsing un-noisy sentences. We adopt the generalized LR parsing algorithm, and a certain scoring scheme to select the most likely sentence out of multiple sentence candidates. The use of a *confusion matrix*, which is created in advance by analyzing a large set of input/output pairs, is discussed to improve the scoring accuracy. The system has been integrated into CMU's knowledge-based machine translation system.

1. Introduction

There have been a few attempts to integrate a speech recognition device with a natural language understanding system. Hayes *et. al* /Hayes86/ adopted the technique of *caseframe instantiation* to parse a continuously spoken English sentence in the form of a *word lattice* (a set of word candidates hypothesized by a speech recognition module) and produce a frame representation of the utterance. Poesio and Rullent /Poesio 1987/ suggested a modified implementation of the caseframe parsing to parse a word lattice in Italian. Lee *et. al* /Lee 1987/ developed a prototype Chinese (Mandarin) dictation machine which takes a syllable lattice (a set of syllables, such as [guo-2] and [tieng-1], hypothesized by a speech recognition module) and produces a Chinese character sequence which is both syntactically and semantically sound.

In this paper, we try to parse a Japanese utterance in the form of a sequence of phonemes.[1] Our speech recognition device, which is a high-speed speaker-independent system developed by Matsushita Research Institute /Morii 1985/, /Hiraoka 1986/ takes a continuous speech utterance, for example "megaitai" ("I have a pain in my eye."), from a microphone and produces a *noisy* phoneme sequence such as "ebaitaai."[2]

The speech recognition device does not have any syntactic or semantic knowledge. More input/output examples of the speech device are presented in Figure 1-1.

<correct sequence>		<recognition output>
igamukamukasuru	--->	igangukamukusjuru
		igamukamonkasjuru
kubigakowabaqteiru	--->	kurigakoogateiru
		azubigakoabaqciiru
atamagaitai	--->	otomogaitai
		atamogeitain

Figure1-1: Input and Output of Recognition Device

Note that the speech recognition device produces a phoneme sequence, not a phoneme lattice; there are no other phoneme candidates available as alternates. We must make the best guess based solely on the phoneme sequence generated by the speech device. Errors caused by the speech device can be classified into three groups:

· **Altered Phonemes** -- Phonemes recognized incorrectly. The second phoneme /b/ in "ebaitaai" is an altered phoneme, for example.

· **Missing Phonemes** -- Phonemes which are actually spoken but not recognized by the device. The first phoneme /m/ in "megaitai", for example, is a missing phoneme.

· **Extra Phonemes** -- Phonemes recognized by the device which are not actually spoken. The penultimate phoneme /a/ in "ebaitaai", for example, is an extra phoneme.

To cope with these problems, we need:

· A very efficient parsing algorithm, as our task requires much more search than conventional typed sentence parsing. And

· A good scoring scheme, to select the most likely sentence out of multiple candidates.

In sections 2 and 3, we describe the parsing algorithm and the scoring scheme, respectively.

2. The Parsing Algorithm

The grammar we are using is an Augmented Context-Free Grammar whose terminal symbols are phonemes rather than words. That is, the grammar includes rules like

1. Phonemes (e.g. /g/, /a/, /s/, etc.) are even lower level units than syllables.
2. We distinguish *noisy* from *ill-formed*. The former is due to recognition device errors, while the latter is due to human users.

Noun --> w a t a s i

instead of

Noun --> 'watasi'

The grammar has been developed primarily for CMU's knowledge-based machine translation system /Tomita 1987/ and consists of more than 2000 rules including lexical rules like one above.[3]

2.1. Generalized LR Parsing

Tomita /Tomita 1985/, /Tomita 1987b/ introduced the *Generalized LR Parsing Algorithm* for Augmented Context-Free Grammars, which can ingeniously handle nondeterminism and ambiguity with a *graph-structured stack*. Tomita also showed that it can be used for a word lattice parsing /Tomita 1986/. Our algorithm here is based on Tomita's parsing algorithm.

A very simple example grammar is shown in Figure 2-1, and its LR parsing table, compiled automatically from the grammar, is shown in Figure 2-2.

```
------------------------------------------
(1)  S   -->  NP  PD
(2)  S   -->  N
(3)  S   -->  PD
(4)  NP  -->  N  P
(5)  N   -->  m  e
(6)  N   -->  i
(7)  P   -->  g  a
(8)  PD  -->  i  t  a  i
------------------------------------------
```

Figure 2-1: An Example Grammar

State	a	i	e	m	g	t	$	N	NP	P	PD	S
0		s4		s5				2	3		1	6
1							r3					
2					s7,r2					8		
3		s9									10	
4					r6	s11,r6						
5			s12									
6							acc					
7	s13											
8		r4										
9					s11							
10							r1					
11	s14											
12					r5		r5					
13		r7										
14		s15										
15							r8					

Figure 2-2: LR Parsing Table with Multiple Entries

Grammar symbols of lower case characters are terminals. The Generalized LR parsing algorithm is a table driven shift-reduce parsing algorithm that can handle arbitrary context-free grammars in polynomial time. Entries "s *n*" in the action table (the left part of the table) indicate the action "shift one word from input buffer onto the stack and go to state *n*". Entries "r *n*" indicate 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 which state the parser should go after a reduce action. While the encountered entry has only one action, parsing proceeds exactly the same way as LR parsers, which are often used in compilers of programming languages. When there are multiple actions in one entry called *conflicts*, all the actions are executed in parallel with the graph-structured stack. We do not describe the Generalized LR parsing algorithm in greater detail, referring the reader to /Tomita 1985/, /Tomita 1986/, /Tomita 1987b/.

2.2. Handling altered, extra, and missing phonemes

To cope with altered, extra and missing phonemes, the parser must consider these errors as it parses an input from left to right. While the algorithm described in the previous subsection cannot handle these noisy phenomena, it is well suited to consider many possibilities at the same time, and therefore, it can be relatively easily modified to handle such noisy phenomena as the following.

- **Altered phonemes** -- Each phoneme in a phoneme sequence may have been altered and thus may be incorrect. The parser has to consider all these possibilities. We can create a phoneme lattice dynamically by placing alternate phoneme candidates in the same location as the original phoneme. Each possibility is then explored by each branch of the parser. Not all phonemes can be altered to any other phoneme. For example, while /o/ can be mis-recognized as /u/, /i/ can never be mis-recognized as /o/. This kind of information can be obtained from a confusion matrix, which we shall discuss in the next section. With the confusion matrix, the parser does not have to exhaustively create alternate phoneme candidates.
- **Extra phonemes** -- Each phoneme in a phoneme sequence may be an extra, and the parser has to consider these possibilities. We have one branch of the parser consider an extra phoneme by simply ignoring the phoneme. The parser assumes at most one extra phoneme can exist between two real phonemes, and we have found the assumption quite reasonable and safe.
- **Missing phonemes** -- Missing phonemes can be handled by inserting possible missing phonemes between two real phonemes. The parser assumes that at most one phoneme can be missing between two real phonemes.

2.3. An Example

In this subsection, we present a sample trace of the parser. Here we use the grammar in Figure 2-1 and the LR table in Figure 2-2 to try to parse the phoneme sequence "ebaitaai" represented in Figure 2-3. (The right sequence is "megaitai" which means "I have a pain in my eye.")

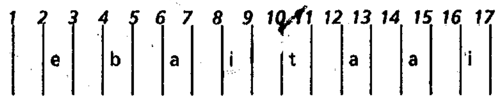


Figure 2-3: An input sequence of phonemes

3. The run-time grammar, which contains both syntax and semantics, is compiled automatically from more abstract formalisms: the Functional Grammar formalism for syntax and frame representation for semantics. For more discussions on this *Universal Parser Architecture*, see /Tomita 1987a/.

In this example we make the following asumptions for altered and missing phonemes.

- /i/ may possibly be mis-recognized as /e/.
- /e/ may possibly be mis-recognized as /a/.
- /g/ may possibly be mis-recognized as /b/.
- /m/ may be missed in the output sequence with a higher probability.

Now we begin parsing: first an initial state 0 is created. The action table indicates that the initial state is expecting "m" and "i" (Figure 2-4). Since the parsing proceeds strictly from left to right, the parser looks for the missing phoneme candidates between the first time frame 1 - 2. (We will use the term T1, T2, ... for representing the time 1, time 2, ... in Figure 2-3.) Only the missing phoneme "m" in this group is applicable to state 0. The new state number 5 is determined from the action table(Figure 2-5).

The next group of phonemes between T2 and T3 consists of the "e" phoneme in the phoneme sequence and the altered candidate phonemes of "e". In this group "e" is expected by state 5 and "i" is expected by state 0(Figure 2-6). After "e" is taken, the new state is 12, which is ready for the action "reduce 5". Thus, using the rule 5(N --> m e), we reduce the phonemes "m e" into N. From state 0 with the nonterminal N, state 2 is determined from the goto table. The action table, then, indicates that state 2 has a multiple entry, i.e., state 2 is expecting "g" and ready for the reduce action(Figure 2-7). Thus, we reduce the nonterminal N into S by rule 2(S --> N), and the new state number 6 is determined from the goto table(Figure 2-8). The action table indicates that state 6 is an accept state, which means that "m e" is a successful parse. But only the first phoneme "e" of the input sequence "ebaitaai" is consumed at this point. Thus we discard this parse by the following constraint.

[Constraint 1] The successful parse should consume the phonemes at least until the phoneme just before the end of the input sequence.

Note that only the parse S in Figure 2-8 is ignored and that the nonterminal N in Figure 2-7 is alive.

Now we return to the Figure 2-6 and continue the shift action of "i". After "i" is taken, the new state 4 is determined from the action table. This state has a multiple entry, i.e. state 4 is expecting "t" and ready for the reduce action. Thus we reduce "i" into N by rule 6. Here we use the *local ambiguity packing* technique, because the reduced nonterminal is the same, the starting state is 0 for both, and the new state is 2 for both. Thus we do not create the new nonterminal N.

Now we go on to the next group of phonemes between T3 and T4. Only "m" is applied to the initial state(Figure 2-9).

The next group of phonemes between T4 and T5 has one applicable phoneme, i.e. an altered phoneme candidate "g" to state 2. After "g" is taken, the new state 7 is determined from the action table (Figure 2-10).

The next group of phonemes between T5 and T6 has only one applicable phoneme; a missing phoneme candidate "m" to state 0. Here we can introduce another constraint which discards this partial-parse.

[Constraint 2] After consuming two phonemes of the input sequence, no phonemes can be applied to the initial state 0.

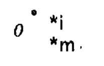


Figure 2-4

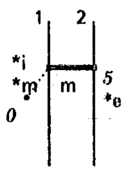


Figure 2-5

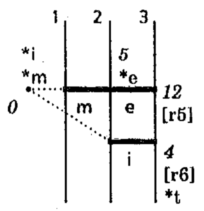


Figure 2-6

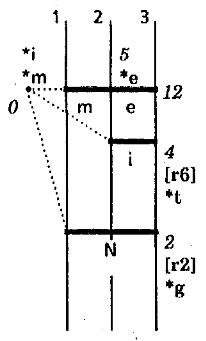


Figure 2-7

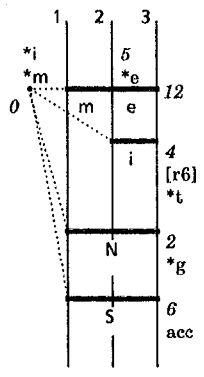


Figure 2-8

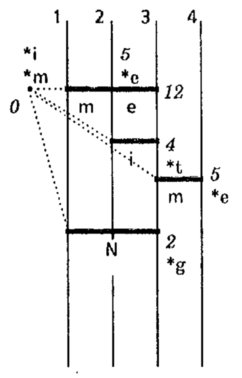


Figure 2-9

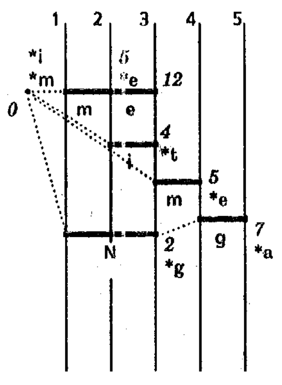


Figure 2-10

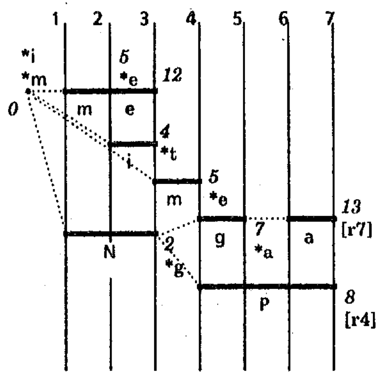


Figure 2-11

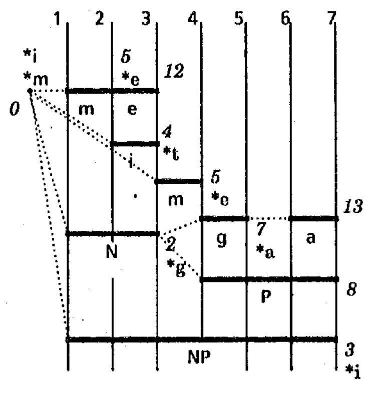


Figure 2-12

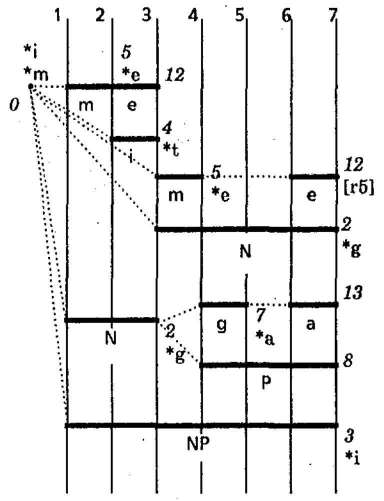


Figure 2-13

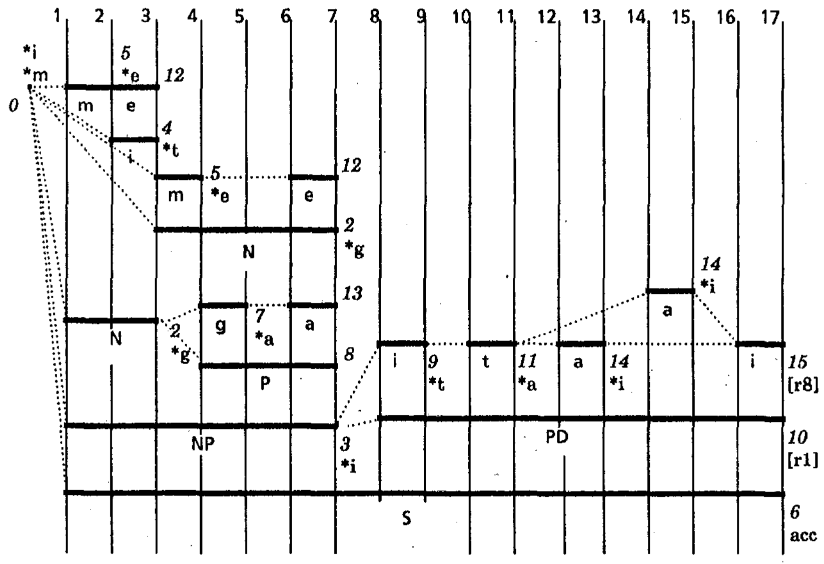


Figure 2-14

This constraint is natural because it is unlikely that more than two phonemes are recorded before the actual beginning phoneme for our speech recognition device.

The next group of phonemes between T6 and T7 has two applicable phonemes, i.e. the output phoneme "a" to state 7 and the altered phoneme candidate "e" to state 5. After "a" is taken, the new state 7 is ready for the reduce action. Thus, we reduce "g a" into P by rule 7 (Figure 2-11). The new state 8 is determined by the goto table, and is also ready for the reduce action. Thus we reduce "N P" into NP by rule 4 (Figure 2-12). The new state is 3. In applying "e", there are two "state 2"s: one is "m" between T1 and T2; the other one is "m" between T3 and T4. Here we can introduce a third constraint which discards the former partial-parse.

[Constraint 3] A shift action is not applied when the distance between the phoneme and the applied (non)terminal is more than 4. (This distance contains at least one real phoneme.)

Figure 2-13 shows the situation after "e" is applied.

The parsing continues in this way, and the final situation is shown in Figure 2-14. As a result, the parser finds two successful parses; "megaitai" and "igaitai"(which means "I have a stomachache.")

3. Scoring and the Confusion Matrix

There are two main reasons why we want to score each parse: first, to prune the search space by discarding branches of the parse whose score is hopelessly low; second, to select the best sentence out of multiple candidates by comparing their scores. Branches of the parse which consider fewer altered/extra/missing phonemes should be given higher scores. Whenever a branch of the parse handles an altered/extra/missing phoneme, a specific penalty is given to the branch. Scoring accuracy can improve with the confusion matrix.

Figure 3-1 shows a part of the confusion matrix made by the manufacturer of the recognition device from the large word data. This matrix tells us, for example, that if the phoneme /a/ is inputed, then the device recognizes it

I\O	/a/	/o/	/u/	/i/	/e/	/j/	/w/	...	(I)	(II)
/a/	93.8	1.1	1.3	0	2.7	0	0	...	0.9	5477
/o/	2.4	84.3	5.8	0	0.3	0	0.6	...	6.5	7529
/u/	0.3	1.8	79.7	2.4	4.6	0.1	0	...	9.7	5722
/i/	0.2	0	0.9	91.2	3.5	0.7	0	...	2.9	6158
/e/	1.9	0	4.5	3.3	89.1	0.1	0	...	1.1	3248
/j/	0	0	1.1	2.3	2.2	80.1	0.3	...	11.4	2660
/w/	0.2	5.1	5.8	0.5	0	2.6	56.1	...	11.2	428
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
(III)	327	176	564	512	290	864	212	...		

(I) rate of missing phonemes
(II) total number of samples
(III) number of extra phonemes

Figure 3-1: A Confusion Matrix (portion)

correctly 93.8% of the time; mis-recognizes it as /o/ 1.1% of the time, as /u/ 1.3% of the time, and so on. The column (I) says that the input is missed 0.9% of the time.

Conversely, if the phoneme /o/ is generated from the device, there is a slight chance that the original input was /a/, /u/ and /w/, respectively, but no chance that the original input was /i/, /e/ or /j/. The probability of the original input being /a/ is much higher than being /w/. Thus, an altered phoneme /w/ should be given a more severe penalty than /a/. A score for altered phonemes can be obtained in this way, missing phonemes should be given a negative score, and extra phonemes should be given a zero or a negative score. With this scoring a score of a partial parse is calculated by summing up the score of each constituent. Therefore, the more likely parse has a higher score.

Two methods have been adopted to prune partial parses by a score:

- Discarding the low-score shift-waiting branches when a phoneme is applied.
- Discarding the low-score branches in a local ambiguity packing.

The former method is very effective when strictly applied.

The confusion matrix only shows us the phoneme-to-phoneme transition, therefore a broader unit transition should also be considered, such as a tendency for the /w/ phoneme in 'owa' or 'owo' to be missed or for the very first /h/ sound of an input to be missed, and the frequent transformation to 'h@' of the 'su' sound in 'desuka.'

4. Conclusions

The parser has been implemented in Common Lisp on a Symbolics Lisp Machine and is being integrated into CMU's knowledge-based machine translation system to accept a spoken Japanese sentence in the domain of doctor-patient conversation and generate sentences in English, German and Japanese.

The parser has been tested against five persons. Each person pronounced 27 sentences in which long sentences are not included due to the limits of the speech recognition device. 84 % of the inputs are parsed correctly and the right sentence appears as the best-score candidate in 88 % out of the correctly parsed inputs. The average parsing time for one sentence is 82 seconds.

Acknowledgements

The authors would like to thank Shuji Morii for giving us the opportunity to use the speech recognition device and to thank other members of the Center for Machine Translation for useful comments and advices. We are also indebted to ATR Interpreting Telephony Research Laboratories for providing the computational environment.

Appendix. Sample Runs

Two actual outputs of the parser are shown on the next page. The first input phoneme sequence is "ebaitaai" and the correct sequence is "megaitai"(which is the same sentence as in the example in Section 2.), which is produced as the top-score sentence of all parses. The second input sequence is "kurigakoogateiru=" and the correct sequence is "kubigakowabaqteiru" which means "I have a stiff neck." The frame-structure output after each parse is the meaning of the sentence. This meaning is extracted in the same way the CMU's machine translation system does. Namely, each rule of the context free grammar has a function which is executed each time the rule is applied (i.e. when the reduce action occurs.) If the function returns nil, this partial parse is discarded because the parse is not correct semantically. If the function returns a non-nil value, the value becomes the semantic of the right-hand-side of the rule and is forwarded to the left-hand-side nonterminal symbol. The details are described in /Tomita 1987a/.

References

/Hayes 1986/ Hayes, P. J., Hauptmann, A. G., Carbonell, J. G., and Tomita, M.
Parsing Spoken Language: A Semantic Caseframe Approach. 11th International Conference on Computational Linguistics (COLING86). Bonn, August, 1986.

/Hiraoka 1986/ Hiraoka, S., Morii, S., Hoshimi, M., and Niyada, K.
Compact Isolated Word Recognition System for Large Vocabulary. IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing (ICASSP86). Tokyo, April, 1986.

/Lee 1987/ Lin-shan Lee, Chiu-yu Tseng, K.J. Chen, and James Huang.
The Preliminary Results of A Mandarin Dictation Machine Based Upon Chinese Natural Language Analysis. Proceedings of the Tenth International Joint Conference on Artificial Intelligence. Milan, August, 1987.

/Morii 1985/ Morii, S., Niyada, K., Fujii, S., and Hoshimi, M.
Large Vocabulary Speaker-independent Japanese Speech Recognition System. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP85). Tampa, March, 1985.

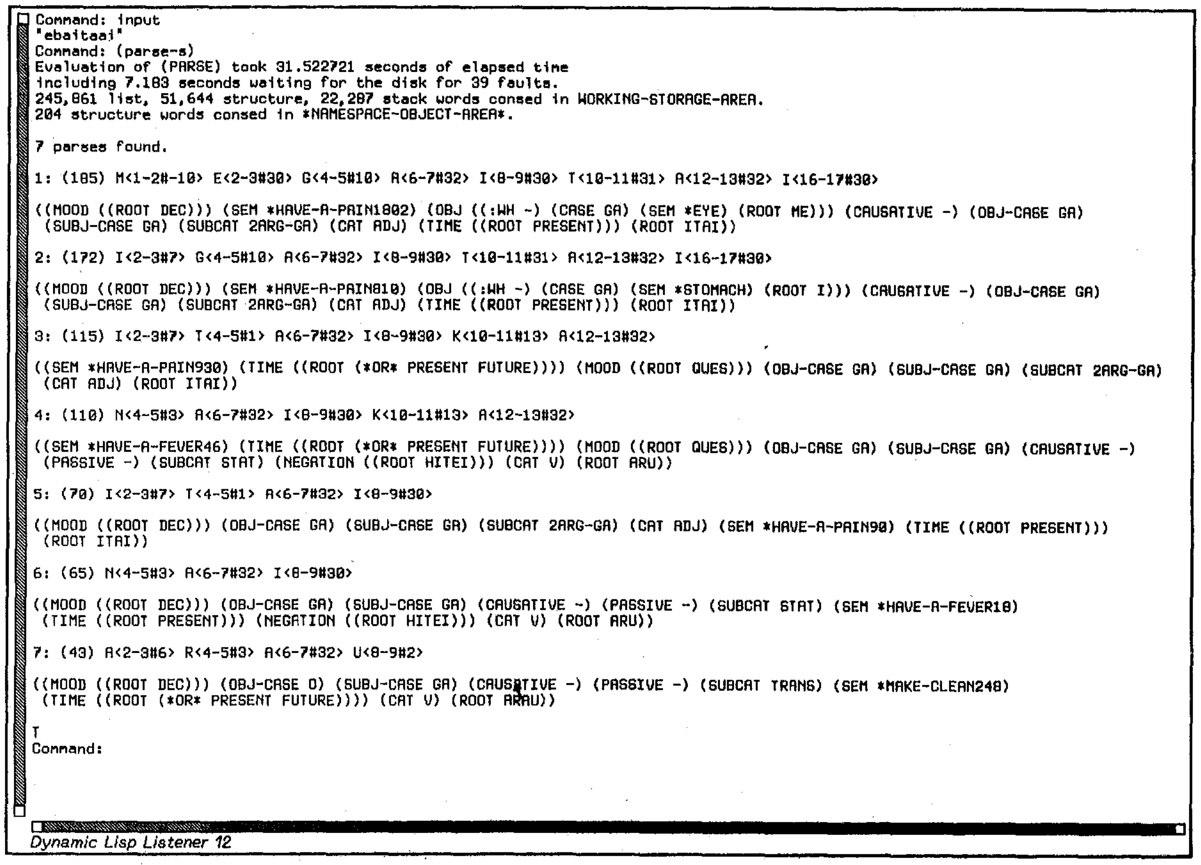
/Poesio 1987/ Poesio, M. and Rullent, C.
Modified Caseframe Parsing for Speech Understanding Systems Based Upon Chinese Natural Language Analysis. Proceedings of the Tenth International Joint Conference on Artificial Intelligence. Milan, August, 1987.

/Tomita 1985/ Tomita, M.
Efficient Parsing for Natural Language: A Fast algorithm for Practical Systems. Kluwer Academic Publishers. Boston, MA, 1985.

/Tomita 1986/ Tomita, M.
An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition. IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing (ICASSP86). Tokyo, April, 1986.

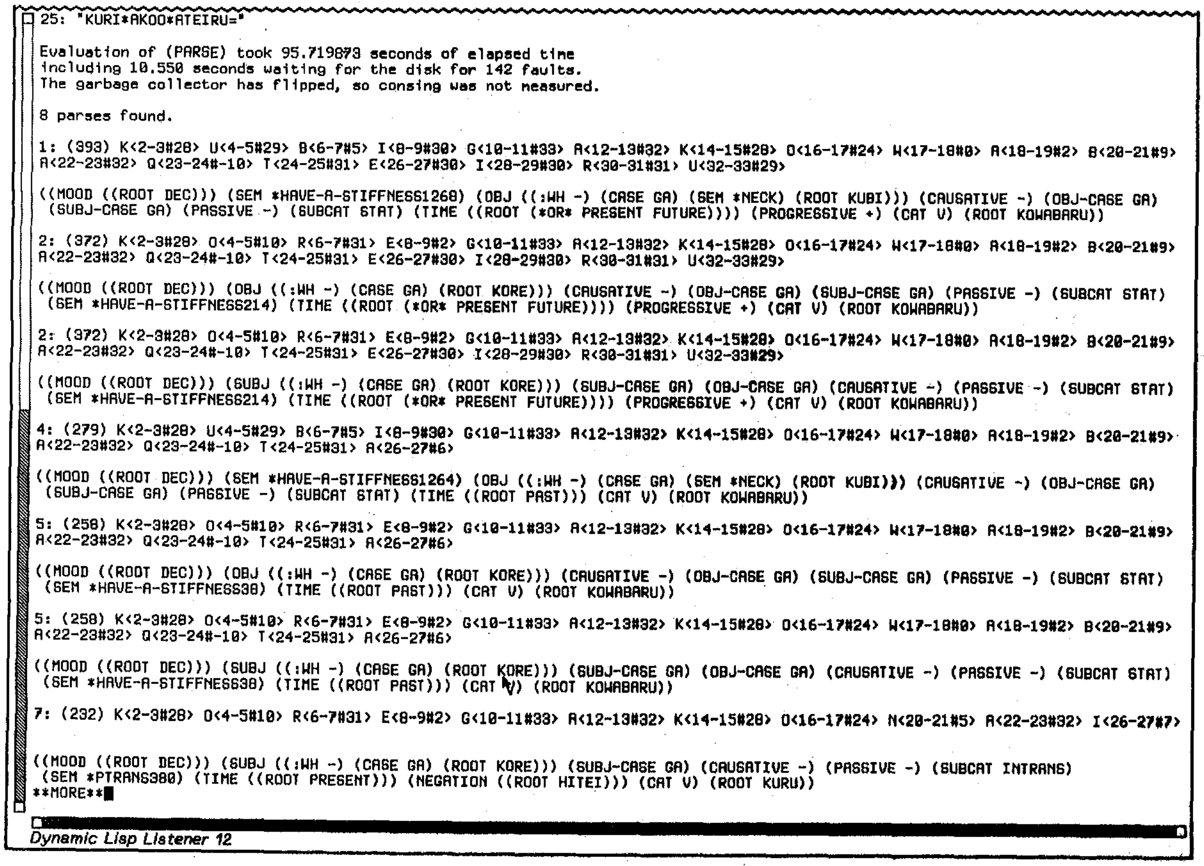
/Tomita 1987a/ Tomita, M. and Carbonell, J. G.
The Universal Parser Architecture for Knowledge-Based Machine Translation. Proceedings of the Tenth International Joint Conference on Artificial Intelligence. Milan, August, 1987.

/Tomita 1987b/ Tomita, M.
An Efficient Augmented-Context-Free Parsing Algorithm. Computational Linguistics. 1987.



```
Command: input
"ebaitaai"
Command: (parse-s)
Evaluation of (PARSE) took 31.522721 seconds of elapsed time
including 7.183 seconds waiting for the disk for 39 faults.
245,861 list, 51,644 structure, 22,287 stack words consed in WORKING-STORAGE-AREA.
204 structure words consed in *NAMESPACE-OBJECT-AREA*.

7 parses found.

1: (185) M<1-2#-10> E<2-3#30> G<4-5#10> A<6-7#32> I<8-9#30> T<10-11#31> A<12-13#32> I<16-17#30>

((MOOD ((ROOT DEC))) (SEM *HAVE-A-PAIN1802) (OBJ ((:WH -) (CASE GA) (SEM *EYE) (ROOT ME))) (CAUSATIVE -) (OBJ-CASE GA)
 (SUBJ-CASE GA) (SUBCAT 2ARG-GA) (CAT ADJ) (TIME ((ROOT PRESENT))) (ROOT ITAI))

2: (172) I<2-3#7> G<4-5#10> A<6-7#32> I<8-9#30> T<10-11#31> A<12-13#32> I<16-17#30>

((MOOD ((ROOT DEC))) (SEM *HAVE-A-PAIN810) (OBJ ((:WH -) (CASE GA) (SEM *STOMACH) (ROOT I))) (CAUSATIVE -) (OBJ-CASE GA)
 (SUBJ-CASE GA) (SUBCAT 2ARG-GA) (CAT ADJ) (TIME ((ROOT PRESENT))) (ROOT ITAI))

3: (115) I<2-3#7> T<4-5#1> A<6-7#32> I<8-9#30> K<10-11#13> A<12-13#32>

((SEM *HAVE-A-PAIN930) (TIME ((ROOT (*OR* PRESENT FUTURE)))) (MOOD ((ROOT QUES))) (OBJ-CASE GA) (SUBJ-CASE GA) (SUBCAT 2ARG-GA)
 (CAT ADJ) (ROOT ITAI))

4: (110) N<4-5#3> A<6-7#32> I<8-9#30> K<10-11#13> A<12-13#32>

((SEM *HAVE-A-FEVER46) (TIME ((ROOT (*OR* PRESENT FUTURE)))) (MOOD ((ROOT QUES))) (OBJ-CASE GA) (SUBJ-CASE GA) (CAUSATIVE -)
 (PASSIVE -) (SUBCAT STAT) (NEGATION ((ROOT HITEI))) (CAT V) (ROOT ARU))

5: (70) I<2-3#7> T<4-5#1> A<6-7#32> I<8-9#30>

((MOOD ((ROOT DEC))) (OBJ-CASE GA) (SUBJ-CASE GA) (SUBCAT 2ARG-GA) (CAT ADJ) (SEM *HAVE-A-PAIN90) (TIME ((ROOT PRESENT)))
 (ROOT ITAI))

6: (65) N<4-5#3> A<6-7#32> I<8-9#30>

((MOOD ((ROOT DEC))) (OBJ-CASE GA) (SUBJ-CASE GA) (CAUSATIVE -) (PASSIVE -) (SUBCAT STAT) (SEM *HAVE-A-FEVER18)
 (TIME ((ROOT PRESENT))) (NEGATION ((ROOT HITEI))) (CAT V) (ROOT ARU))

7: (43) A<2-3#6> R<4-5#3> A<6-7#32> U<8-9#2>

((MOOD ((ROOT DEC))) (OBJ-CASE O) (SUBJ-CASE GA) (CAUSATIVE -) (PASSIVE -) (SUBCAT TRANS) (SEM *MAKE-CLEAN248)
 (TIME ((ROOT (*OR* PRESENT FUTURE)))) (CAT V) (ROOT ARAU))

T
Command:
```

Sample Run 1



```
25: "KURI*AKOO*ATEIRU="

Evaluation of (PARSE) took 95.719873 seconds of elapsed time
including 10.550 seconds waiting for the disk for 142 faults.
The garbage collector has flipped, so consing was not measured.

8 parses found.

1: (393) K<2-3#28> U<4-5#29> B<6-7#5> I<8-9#30> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> W<17-18#0> A<18-19#2> B<20-21#9>
A<22-23#32> Q<23-24#-10> T<24-25#31> E<26-27#30> I<28-29#30> R<30-31#31> U<32-33#29>

((MOOD ((ROOT DEC))) (SEM *HAVE-A-STIFFNESS1268) (OBJ ((:WH -) (CASE GA) (SEM *NECK) (ROOT KUBI))) (CAUSATIVE -) (OBJ-CASE GA)
 (SUBJ-CASE GA) (PASSIVE -) (SUBCAT STAT) (TIME ((ROOT (*OR* PRESENT FUTURE)))) (PROGRESSIVE +) (CAT V) (ROOT KOWABARU))

2: (372) K<2-3#28> O<4-5#10> R<6-7#31> E<8-9#2> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> W<17-18#0> A<18-19#2> B<20-21#9>
A<22-23#32> Q<23-24#-10> T<24-25#31> E<26-27#30> I<28-29#30> R<30-31#31> U<32-33#29>

((MOOD ((ROOT DEC))) (OBJ ((:WH -) (CASE GA) (ROOT KORE))) (CAUSATIVE -) (OBJ-CASE GA) (SUBJ-CASE GA) (PASSIVE -) (SUBCAT STAT)
 (SEM *HAVE-A-STIFFNESS214) (TIME ((ROOT (*OR* PRESENT FUTURE)))) (PROGRESSIVE +) (CAT V) (ROOT KOWABARU))

2: (372) K<2-3#28> O<4-5#10> R<6-7#31> E<8-9#2> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> W<17-18#0> A<18-19#2> B<20-21#9>
A<22-23#32> Q<23-24#-10> T<24-25#31> E<26-27#30> I<28-29#30> R<30-31#31> U<32-33#29>

((MOOD ((ROOT DEC))) (SUBJ ((:WH -) (CASE GA) (ROOT KORE))) (SUBJ-CASE GA) (OBJ-CASE GA) (CAUSATIVE -) (PASSIVE -) (SUBCAT STAT)
 (SEM *HAVE-A-STIFFNESS214) (TIME ((ROOT (*OR* PRESENT FUTURE)))) (PROGRESSIVE +) (CAT V) (ROOT KOWABARU))

4: (279) K<2-3#28> U<4-5#29> B<6-7#5> I<8-9#30> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> W<17-18#0> A<18-19#2> B<20-21#9>
A<22-23#32> Q<23-24#-10> T<24-25#31> A<26-27#6>

((MOOD ((ROOT DEC))) (SEM *HAVE-A-STIFFNESS1264) (OBJ ((:WH -) (CASE GA) (SEM *NECK) (ROOT KUBI))) (CAUSATIVE -) (OBJ-CASE GA)
 (SUBJ-CASE GA) (PASSIVE -) (SUBCAT STAT) (TIME ((ROOT PAST))) (CAT V) (ROOT KOWABARU))

5: (258) K<2-3#28> O<4-5#10> R<6-7#31> E<8-9#2> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> W<17-18#0> A<18-19#2> B<20-21#9>
A<22-23#32> Q<23-24#-10> T<24-25#31> A<26-27#6>

((MOOD ((ROOT DEC))) (OBJ ((:WH -) (CASE GA) (ROOT KORE))) (CAUSATIVE -) (OBJ-CASE GA) (SUBJ-CASE GA) (PASSIVE -) (SUBCAT STAT)
 (SEM *HAVE-A-STIFFNESS38) (TIME ((ROOT PAST))) (CAT V) (ROOT KOWABARU))

5: (258) K<2-3#28> O<4-5#10> R<6-7#31> E<8-9#2> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> W<17-18#0> A<18-19#2> B<20-21#9>
A<22-23#32> Q<23-24#-10> T<24-25#31> A<26-27#6>

((MOOD ((ROOT DEC))) (SUBJ ((:WH -) (CASE GA) (ROOT KORE))) (SUBJ-CASE GA) (OBJ-CASE GA) (CAUSATIVE -) (PASSIVE -) (SUBCAT STAT)
 (SEM *HAVE-A-STIFFNESS38) (TIME ((ROOT PAST))) (CAT V) (ROOT KOWABARU))

7: (232) K<2-3#28> O<4-5#10> R<6-7#31> E<8-9#2> G<10-11#33> A<12-13#32> K<14-15#28> O<16-17#24> N<20-21#5> A<22-23#32> I<26-27#7>


((MOOD ((ROOT DEC))) (SUBJ ((:WH -) (CASE GA) (ROOT KORE))) (SUBJ-CASE GA) (CAUSATIVE -) (PASSIVE -) (SUBCAT INTRANS)
 (SEM *PTRANS380) (TIME ((ROOT PRESENT))) (NEGATION ((ROOT HITEI))) (CAT V) (ROOT KURU))
**MORE**
```

Sample Run 2