INTERFACE. Journal of Applied Linguistics 3.1 (1988), 57-77

TOWARDS A SPEECH-TO-SPEECH TRANSLATION SYSTEM (1)

Masaru TOMITA, Marion KEE, Hiroaki SAITO,
Teruko MITAMURA & Hideto TOMABECHI
Carnegie-Mellon University, Pittsburgh

ABSTRACT

We describe our Universal Parser Architecture and its use in a speech translation system, based on the Machine Translation system under development at the Center for Machine Translation at Carnegie Mellon. To "understand" natural language, a system must have syntactic knowledge of the language and semantic knowledge of the domain. The Universal Parser Architecture allows grammar writers to develop these kinds of knowledge separately in a declarative manner, and the compiler/interpreter integrates these two knowledge bases dynamically in order to parse input sentences. We recently integrated our system with a speech recognition system to accept spoken sentences (rather than typed sentences) by extending our runtime parser component to handle "noisy" phoneme sequences (of spoken utterances) that possibly include recognition errors. The nature of this modification is explained and examples are presented. We find our architecture very suitable for speech translation, as it combines the use of domain semantic knowledge with a highly efficient runtime parsing algorithm, thus accommodating the increased search space necessary for parsing speech input.

1. INTRODUCTION

To "understand" natural language, a system must have syntactic knowledge of the language and semantic knowledge of the domain. We want to separate these kinds of knowledge so that semantics for any given domain can be used for many languages and the syntactic grammar for any given language can be used for many different domains. The Universal Parser Architecture allows grammar writers to develop these kinds of knowledge separately in a modular manner, and the compiler/interpreter integrates these two knowledge bases dynamically in order to parse input sentences. We believe that, for future systems, this architecture is essential in a software engineering sense, as grammars are getting larger and more complex in practical applications such as knowledge-based machine translation systems. In essence, we are building a "Parser Factory" whose raw materials are syntactic grammars and domain knowledge bases, whose machinery is the Universal Parser Compiler, and whose output is integrated semantic/ syntactic runtime parsing systems. Such systems are very efficient, but not necessarily human-readable in their compiled form.

The Universal Parser Architecture has been employed in CMU's multilingual machine translation system. We recently integrated our system with a speech recognition system to accept spoken sentences (rather than typed sentences) by extending our runtime parser component to handle "noisy" phoneme sequences (of spoken utterances) that possibly include recognition errors. There are four major reasons why we feel that a knowledge-based system such as the Universal Parser is appropriate for speech translation. First, speech translation is always final (human post-editing is not applicable to speech); thus it must be accurate, which requires the comprehension available from a knowledge-based system. Second, speech utterances often contain elliptical, anaphoric or syntactically ill-formed expressions, which require knowledge-based analysis in order to be resolved. Third, a speech system must respond in real time, and requires an efficient algorithm such as ours. Fourth, a speech parsing system must be able to handle recognition errors made by its speech recognition device; this greatly increases the necessary search space, which can be constrained by the use of knowledge, but still requires a highly efficient algorithm.

In this paper, we first describe the Universal Parser Compiler and its knowledge sources, and then explain our Generalized LR Parsing algorithm, and its extension to handle speech recognition, with examples.

Section 2 describes the components of the Universal Parser Compiler. We use the *Pseudo Unification Grammar formalism,* which is in a similar notation to that of the Unification Grammar formalism, for representing linguistic (syntactic) knowledge. For the representation of domain semantic knowledge, we adopt *FrameKit,* which is a frame representation system developed at Carnegie Mellon University. The Universal Parser Compiler compiles these knowledge bases and produces the Augmented Context-Free Grammar (ACFG) for runtime parsing. An ACFG is essentially a list of context-free phrase structure rules, to each of which is attached a LISP program (augmentation) (2). These LISP programs are further compiled into machine code using the standard LISP compiler. The phrase structure part of the ACFG is also compiled further into a Generalized LR parsing table for the Generalized LR parsing algorithm described in Section 3.2.

Section 3 outlines the Generalized LR parsing algorithm, the runtime parsing algorithm for the compiled grammar, and goes on to show how it is modified for use in speech recognition. It is an LR parsing algorithm extended to handle arbitrary context-free grammars. A *graph-structured stack,* the key to the way the Generalized LR algorithm handles nondeterminism in stack operations, is described. We then show how this algorithm is extended to handle spoken input, i.e., noisy phoneme sequences of Japanese utterances produced by an experimental continuous speech recognition system developed by Matsushita Research Institute. An extended example is used to illustrate the workings of the modified algorithm.

Some concluding remarks are made in section 4. Appendix I contains actual sample outputs from our speech translation system.

2. THE UNIVERSAL PARSER ARCHITECTURE

Multi-lingual systems require parsing multiple source languages, and thus a universal parser, which can take a language grammar as input (rather than building the grammar into the interpreter proper) is much preferred for reasons of extensibility and generality. When dealing with multiple languages, the linguistic structure is no longer a universal invariant that transfers across all applications (as it was for pure English language parsers), but rather is another dimension of parameterization and extensibility. However, semantic information can remain invariant across languages (though, of course, not across domains). Therefore, it is crucial to keep semantic knowledge sources separate from syntactic ones, so that if new linguistic information is added, it will apply across all semantic domains, and if new semantic information is added it will apply to all relevant languages. The question, of course, is how to accomplish this factoring, and how to accomplish it without making major concessions to either run-time efficiency or semantic accuracy.

The idea of the Universal Parser is depicted in figure 2-1. There are three kinds of knowledge sources: one containing syntactic grammars for different languages, one containing semantic knowledge bases for different domains, and one containing sets of rules which map syntactic forms (words and phrases) into the semantic knowledge structure. Each of the syntactic grammars is totally independent of any specific domain, and likewise, each of the semantic knowledge bases is totally independent of any specific language. The mapping rules are both language- and domain-dependent, and a different set of mapping rules is created for each language/domain combination. Syntactic grammars, domain knowledge bases, and mapping rules are written in a highly abstract, human-readable manner. This organization makes them easy to extend or modify but possibly machine-inefficient for a run-time parser. The grammar compiler takes one of the syntactic grammars (say Language L_i) and one of the domain knowledge bases (say Domain D_j, along with the appropriate set of mapping rules (in this case, Mapping L_i D_j), and produces one large grammar which contains both syntactic and semantic information. Such compilation proceeds off-line, producing a compiled grammar that need not be human-readable, but must be machine-efficient in terms of on-line run-time parsing speed. The pre-compiled grammar, in essence, consists of the legal subset of the cross product of L_i and D_j, cross-indexed and optimized for efficient machine access and computation. When the user inputs sentences in Language L_i (and Domain D_j), the run-time parser parses the sentences very efficiently, referencing only the compiled grammar, and producing semantic representations of the sentences.

We adopt *semantic case frames* for domain knowledge representation and the *Pseudo Unification Grammar formalism* for syntactic grammar representation. The run-time grammar produced by the multi-phase Universal Parser Compiler is an *augmented context-free grammar (ACFG)* which is further compiled into a *generalized LR table to* be used by a run-time parser based on the Generalized LR parsing algorithm [8], which is described briefly in section 3.2. The various types of knowledge sources

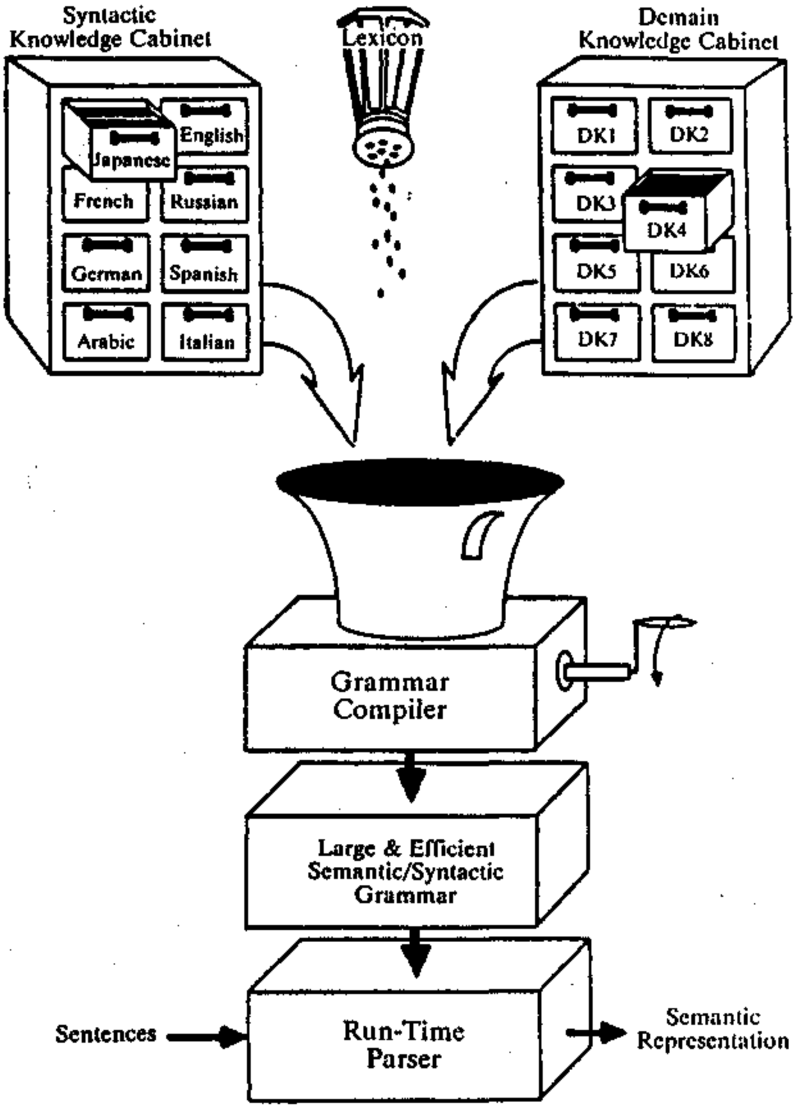


Figure 2-1.: Universal Parser Concept

used by the Universal Parser Compiler are described in detail in the following subsections.

2.1. Representation of Domain Semantic Knowledge

The semantic frame representation embodies domain knowledge in a language-neutral knowledge representation. We build an inheritance hierarchy of frames representing concepts. Each frame represents a different concept such as an object, action, state, process, etc. In addition to typing information, the frames encode semantic restriction information that prevents non-productive combinations from being computed. (For instance, only visible objects may have a color attribute.) We use the Framekit [5] knowledge representation language to encode domain semantic knowledge. Framekit is a general-purpose frame representation language developed at CMU.

To date, we have developed two different domains for use with the Universal Parser: doctor-patient conversations and personal computer manuals. The personal computer manual domain contains frames for concepts such as the personal computer, its various parts, actions such as turning switches on and off, etc. Frames in the doctor-patient domain include *HAVE-A-SYMPTOM (an event frame), and object frames for symptoms, body parts, pain, etc. Higher-level frames such as the overall concept definitions for actions and time expressions are similar across both domains.

Here are some sample frames from the domain of doctor-patient conversations. This domain is targeted primarily at the patient's initial discussion with the doctor about some physical complaint.

```
(*INGEST-MEDICINE
    (is-a (value *ACTION))
    (:object (sem *MEDICINE))
    (:ingest-with (sem *FOOD *MEDICINE))
    )

(TREATMENT-SUBSTANCE
    (is-a (value *NOMINAL))
    (:quant (sem *QUANTITY))
    )

(*MEDICINE
    (is-a (value *TREATMENT-SUBSTANCE))
    (:quant (sem *MEDICINE-QUANTITY))
    (:name (sem *NOMINAL))
    )
```

Here is an example of the format of actual parser output for the sentence "Did you take aspirin three hours ago" using the frames above.

```
(((:CFNAME *INGEST-MEDICINE) (:MOOD QUES)
  (:AGENT
   ((:CFNAME *PATIENT) (:HUMAN +) (:PRO +)
    (:NUMBER SG PL) (:PERSON 2)))
  (:TIME PAST)
  (:WHEN
    ((:CFNAME *TIME)
    (:TIME-RELATION
     ((:CFNAME * TIME-RELATION) (DIRECTION -)
       (:INTERVAL ((:CFNAME *DURATION) (:HOUR 3)))))))
  (:OBJECT
   ((:CFNAME * MEDICINE)
    (:QUANT ((:CFNAME *MEDICINE-QUANTITY)))
    (:NAME *ASPIRIN)))))
```

2.2. Representation of Linguistic Knowledge

We express the grammar of each particular language in a grammar formalism called Pseudo Unification Grammar. Unlike Unification Grammar formalisms, which are based on the operation called *unification,* the Pseudo Unification Grammar formalism is based on the operation called *pseudo unification.* The pseudo unification does not exactly do the unification, but it does something very close (3). On the other hand, implementation of pseudo unification is much simpler than that of standard unification, especially with disjunctions, structure sharing, and other complicated phenomena.

Here is an example of a Pseudo Unification Grammar rule. This rule is designed to capture a simple declarative sentence in English.

```
(<DEC> ↔ (<NP> <VP>)
    (((xl case) = nom)
     ((x2 form) =c finite)
     (*OR*
     (((x2 :time) = present)
      ((xl agr) = (x2 agr)))
     (((x2 :time) = past)))
     (x0 = x2)
      ((x0 passive) = -)
      ((x0 subj) = xl)))
```

A grammar rule consists of a context-free phrase structure rule followed by a list of pseudo equations. Pseudo equations are similar to the PATR-II notation [6]. "x0" refers to the functional structure corresponding to the left hand side of the rule. "xl" and "x2" refer to the first and second element of the right hand side of the rule, respectively. The equations in the above example roughly state:

The case of the noun phrase must be nominative; the form of the verb phrase must be finite; if the tense of the verb phrase is present, then agreements of the noun phrase and verb phrase must agree, or the tense must be past; "x0"(<S>) inherits all features from the verb phrase; and make it (passive -); and create a subject slot and put all the noun phrase features into that slot.

Besides ordinary equations, the following practical features are available in the Pseudo Unification Grammar formalism.

— Arbitrary LISP function calls. This feature is particularly useful when we want to do some kind of semantic processing or inference in parallel to the syntactic parsing.

— A *MULTIPLE* operator for assigning multiple values. This feature is particularly useful for adjuncts and conjunctions.

— A *NOT* operator for negation.

— *DEFINED* and *UNDEFINED* operators to check if a particular attribute is defined (undefined).

— A Wild Card symbol to match with any word. This is particularly useful for parsing proper nouns.

A grammar can be interpreted on a *character-by-character basis,* rather than a *word-by-word* basis; that is, terminal symbols of a grammar are characters, not words. It is so designed with a view to the use of this system for *unsegmented languages* such as Japanese, in which there are no boundary spaces between words, as well as for continuously spoken sentences, which are sequences of phonemes with no clear word boundary as described in section 3. One benefit of this character-based feature is that the lexical dictionary and the morphological rules can be written in the same formalism as the syntactic rules.

2.3. The Lexicon/Concept Mapping Rules

The mapping rules are that portion of the knowledge sources which is both domain- and language-dependent. Their function is to unite the concept frame definitions for a particular domain with the syntactic rules for a given language. Each mapping rule consists of a frame name and the syntactic elements associated with that concept in a given language.

In any language, one can use different syntactic structures to express a given set of concepts. For example, the sentences "I have a pain in the stomach" and "I have a stomach ache" refer to the same concepts. The mapping rules allow us to associate differing syntactic f-structures with the same semantic representation. On the other hand, a word can have different meanings depending upon the context in which it is used. For example, the English verb *take* is used in various ways in the doctor-patient domain:

I take the medicine every day.
I took a hot bath.
I took my temperature.

In order to differentiate among these three kinds of actions, we need to associate the first *take* with the *INGEST-MEDICINE frame, the second with the *BATHE frame, and the third the *MEASURE frame. The mapping rules allow us to associate a single word with different semantic representations depending upon the syntactic structure within which the word is used.

Each of the frames in the concept lexicon needs to connect to a certain level of syntactic structure. In order to do this, we first have to specify which frames take which root in the f-structures. Sentential frames, such as *BATHE and *INGEST-MEDICINE, require verb roots, whereas nominal frames, such as *PAIN and *BODY-PART, take noun roots. We associate frame names with lexical roots as follows:

```
Sentential frames:
    (*INGEST-MEDICINE  ↔  (*OR* take swallow))
    (*BATHE            ↔   bathe)

Nominal frames:
    (*PAIN             ↔  (*OR* pain ache))
    (*BODY-PART        ↔  stomach)
```

Notice that *take* and *swallow,* or *pain* and *ache* are treated as synonymous in the rules because both words can occur in the same syntactic structure without changing the meaning of the sentence:

I took two tablets of aspirin.
I swallowed two tablets of aspirin.

I have a pain in my stomach.
I have an ache in my stomach.

After putting the verb roots in each rule, we need to specify syntactic restrictions for matching the frame. In this example, we compare the frame definition of *INGEST-MEDICINE with the sentences which we want to map:

Take Tylenol.
I took three tablets of aspirin.
I swallowed two aspirin with a glass of milk.

```
(*INGEST-MEDICINE
    (is-a (value *PATIENT-ACTION))
    (:object (sem *MEDICINE))
    (:ingest-with (sem *FOOD *MEDICINE)))
```

'Tylenol', 'three tablets of aspirin', and 'two aspirin' are objects taken or swallowed, and are associated with the :object slot of the *MEDICINE frame. They also are the syntactic objects of the sentences shown, called obj in the syntactic f-structure. 'A glass of milk' is something that we take the medicine with, so it fills the :*ingest-with* slot in this frame; and syntactically it is a *ppadjunct.* Thus, we need to write the following mapping rule for the *INGEST-MEDICINE frame:

```
(*INGEST-MEDICINE  ↔ (*OR* take swallow)
    ((:object  ↔ (obj))
    ((:ingest-with) ↔ (ppadjunct (prep = with)))))
```

Notice that we need to specify the preposition of *ppadjunct* as a constraint. Otherwise, any *ppadjunct* can be mapped onto this slot if it contains food or medicine in its NP.

The following examples contain three frames, and each frame is followed by Japanese, English and French mapping rules. The verbs *itadaku* and *have* can be used for both solid foods and liquid foods (but note that there is no such verb in French). Therefore, these verbs are mapped onto the more general frame, *INGEST-FOOD. On the other hand, the verbs *taberu*, *eat* and *manger* are used with solid foods, since we cannot say ,*he eats milk'. Thus, these verbs are mapped onto the *INGEST-SOLID-FOOD frame.

```
(f *INGEST-FOOD
    (is-a (value *PATIENT-ACTION))
    (:object (sem *FOOD))
    (:quant (sem *FOOD-QUANTITY))
    )
        (jpn *INGEST-FOOD ↔ itadaku
                (:quant ↔ (advadjunct))
                (:object ↔ (obj)))
        (eng *INGEST-FOOD ↔ have
                (:object ↔ (obj)))
(f *INGEST-SOLID-FOOD
    (is-a (value *INGEST-FOOD))
    :object (sem *SOLID-FOOD))
    )
        (jpn *INGEST-SOLID-FOOD ↔ taberu
                (:object ↔ (obj)))
        (eng *INGEST-SOLID-FOOD ↔ eat
                (:object ↔ (obj)))
        (fre *INGEST-SOLID-FOOD ↔ manger
                (:object ↔ (obj)))
(f *NGEST-LIQUID-FOOD
    (is-a (value *INGEST-FOOD))
    (:object (sem *LIQUID-FOOD))
    )
        (jpn *INGEST-LIQUID-FOOD ↔ nomu
                (:object ↔ (obj)))
        (eng *INGEST-LIQUID-FOOD ↔ drink
                (:object ↔ (obj)))
        (fre *INGEST-LIQUID-FOOD ↔ boire
                (:object ↔ (obj)))
```

2.4. Augmented Context-Free Grammar for Runtime

The Universal Parser Compiler takes the three knowledge sources described in the previous subsections, and produces a runtime grammar in the formalism called Augmented Context-Free Grammar (ACFG). An ACFG is essentially a list of context-free phrase structure rules, except that we attach a LISP function to each phrase structure rule. Whenever constituents are reduced into a higherlevel nonterminal using a phrase structure rule, the LISP program associated with the rule is evaluated. The LISP program handles such aspects as constructing a syntactic/semantic representation of the input sentence, passing attribute values among constituents at different levels, and checking syntactic/semantic constraints.

If the LISP function returns NIL, the rule will not be used. If the LISP function returns a non-NIL value, then this value is given to the newly created non-terminal. The value includes attributes of the nonterminal and of the partial syntactic/semantic representation constructed thus far. It is important to emphasize that, at runtime, all the parser has to do in order to perform complex syntactic/semantic operations is to simply evaluate (i.e., run) the LISP programs. Notice that those LISP functions can be precompiled into machine code by the standard LISP compiler.

3. PARSING SPOKEN SENTENCES

This section describes the Generalized LR parsing algorithm, and shows how it can be extended to handle "noisy" sentences that possibly include errors due to 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 Japanese grammar in the Pseudo Unification Grammar formalism is written in such a way that its terminal symbols are morphemes; thus the morphological rules and the phonological rules are written in the same formalism. This means that the runtime grammar compiled by the Universal Parser Compiler includes all of the lexical, phonological, morphological, syntactic and domain semantic knowledge. The Generalized LR parsing algorithm has been adopted and modified to handle recognition errors made by the speech recognition device.

The have been a few attempts to integrate a speech recognition device with a natural language understanding system. Hayes *et. al* [7] adopted the technique of *caseframe instantiation* to parse a continously 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 [1] suggested a modified implementation of the caseframe parsing to parse a word lattice in Italian. Lee *et. al* [2] 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.

We try to parse a Japanese utterance in the form of a sequence of phonemes. (4) Our speech recognition device, which was developed by Matsushita Research institute [3, 4], 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" (5). The speech recognition device does not have any syntactic or semantic knowledge. More input/output examples of the speech device are presented in figure 3-1.

correct sequence of phonemes	output of the recognition device	
ingamukamukasuru	→	ingangukamukusjuru
		ingamukamonkasjuru
kubingakowabaqteiru	→	kubingakooboqteiiru
		kubingakooboqtuingju
atamangaitai	→	otomongaitai
		atamongeitain

Figure 3-1 : Input and Output of the Speech Recognition Device

3.1. Coping with Recognition Errors

Note that the speech recognition device produces a phoneme sequence, not a phoneme lattice ; that is, there are no other phoneme candidates available to alternate with any phoneme actually presented by the recognition device. We must make the best guess about possible alternate phonemes, 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 not recognized by the device which are actually spoken. The first phoneme /m/ in "megaitai" is a missing phoneme, for example.
— Extra Phonemes -- Phonemes recognized by the device which are not actually spoken. The penultimate phoneme, /a/, in "ebaitaai" is an extra phoneme, for example.

To cope with these problems, we need:

1, A very efficient parsing algorithm, as our task requires much more search than conventional typed sentence parsing.
2. A good scoring scheme to select the most likely sentence out of multiple candidates.

To cope with altered, extra and missing phonemes, the parser must consider these errors as it parses an input from left to right. The Generalized LR Parsing Algorithm is well suited to consider many possibilities at the same time, and therefore, it can be relatively easily modified to handle noisy phenomena:

— 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.

— 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.

— 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 that at most one extra phoneme can exist between two real phonemes, and we have found the assumption quite reasonable and safe.

3.2. Generalized LR Parsing

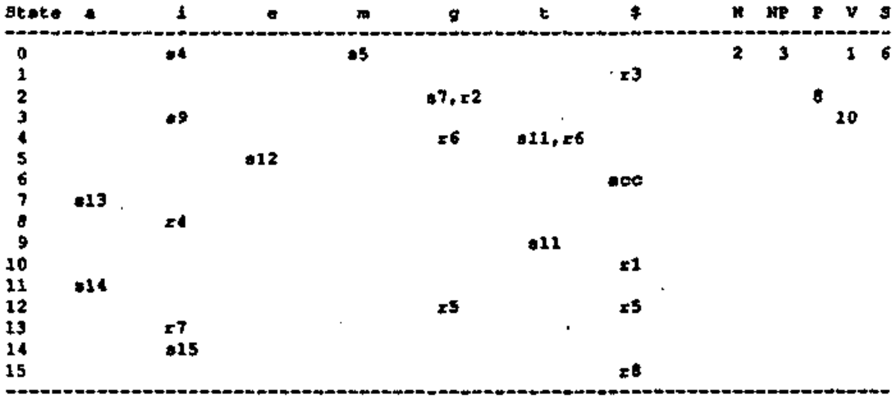
Tomita [8, 10] introduced the *Generalized LR Parsing Algorithm* for Augmented Context-Free Grammars, which can ingeniously handle non-determinism and ambiguity with a *graph-structured stack.* Tomita also showed that it can be used for a word lattice parsing [9]. Our algorithm here is based on Tomita's parsing algorithm.

A very simple example grammar is shown in Figure 3-2, and its LR parsing table, compiled automatically from the grammar, is shown in Figure 3-3.

(1) S → NP V
(2) S → N
(3) S → V
(4) NS → N P
(5) N → m e
(6) N → i
(7) P → g a
(8) V → i t a i

Figure 3-2 : An Example Grammar

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. In case there are multiple actions in one entry, it executes all the actions with the graph-structured stack. We do not describe the Generalized LR parsing algorithm in any more detail, referring the reader to [8, 10, 9].



State	a	i	e	m	g	t	$	N	NP	P	V	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 3-3 : LR Parsing Table with Multiple Entries

3.3. A Sample Parse

Using the grammar in Figure 3-2 and its LR table in Figure 3-3, let us try to parse the phoneme sequence "ebaitaai." (The correct sequence is "megaitai" which means "I have a pain in my eye.")

```
1   2   3   4   5   6   7   8  9   10 11   12  13   14  15   16  17
|   |e  |   |b  |   |a  |   |i  |   |t  |   |a  |   |a  |   |i  |
```

Figure 3-4: An input sequence of phonemes

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

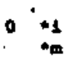


Figure 3-5

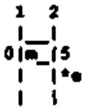


Figure 3-6

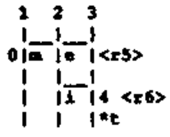


Figure 3-7

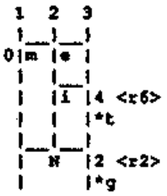


Figure 3-8

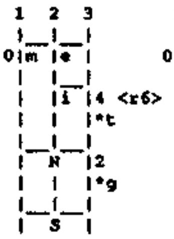


Figure 3-9

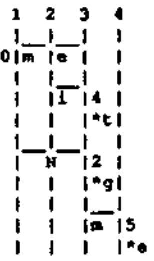


Figure 3-10

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 3-7). 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 3-8). 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 3-9). 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 3-9 is ignored and that the nonterminal N in Figure 3-8 is alive.

Now we return to the Figure 3-7 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 3-10).

The next group of phonemes between T4 and T5 has two applicable phonemes, i.e. "m" to state 0 and "g" to state 2. After "g" is taken, the new state 7 is determined from the action table (Figure 3-11).

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

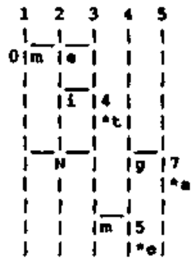


Figure 3-11

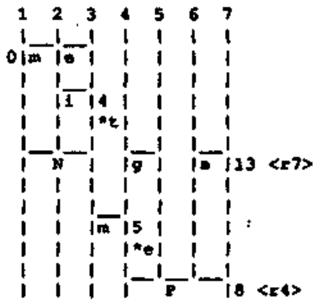


Figure 3-12

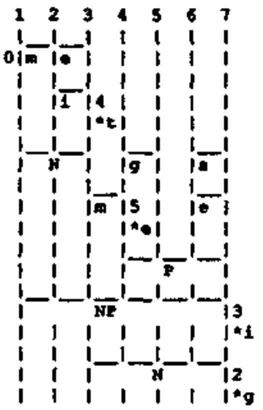


Figure 3-13

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

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. "a" to state 7 and "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 3-12). 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. 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 3-13 shows the situation after "e" is applied.

The parsing continues in this way, and the final situation is shown in Figure 3-14 (a little simplified).

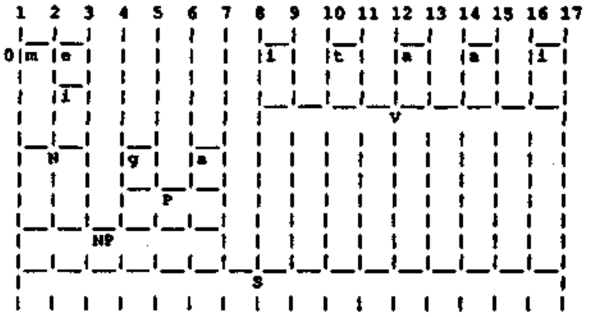


Figure 3-14

3.4. 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. In the next two subsections, a simple scoring method without the confusion matrix and a more sophisticated scoring method with the confusion matrix are respectively described.

3.4.1. *Penalty*

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. The following table indicates example scoring points.

real phoneme	10
altered phoneme	8
extra phonemes	0
missing phonemes	-6

Real phonemes are given 10 points while other "fake" phonemes are given less. Since the higher score indicates better accuracy, we can just simply add scores of each phoneme as the parsing proceeds.

3.4.2. *The confusion Matrix*

Scoring accuracy can improve with the confusion matrix:

INPUT	/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 3-15 : A Confusion Matrix (portion)

Figure 3-15 shows a part of the confusion matrix. This matrix tells us, for example, that if the phoneme /a/ is inputed, then the device recognizes it 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/.

See Appendix I for examples of actual parser output.

4. DISCUSSION AND CONCLUDING REMARKS

In this paper, we have described the Universal Parser Architecture for multi-lingual knowledge-based parsing systems, and its use in a speech translation system. The Universal Parser Architecture is very suitable for speech translation for two reasons :

1. Domain semantic knowledge seems absolutely necessary to the task, and
2. A very efficient algorithm is required at runtime, as the search space for parsing speech input is much larger than that for typed input, due to recognition errors.

Preliminary results of integrating the Universal Parser Architecture and a speech recognition system have been presented. The results so far have been very promising in terms of both accuracy and speed (see Appendix I.).

Future improvements are planned for the Universal Parser, to make it easier to write and alter grammars. It takes about 15-20 minutes to compile our largest grammar (with over 2000 rules) all the way into an augmented LR parsing table. For smaller grammars with 300 rules, it takes only a couple of minutes. We are developing an *incremental compilation mode,* in which the grammar writer does not have to compile everything each time he makes a small change to the grammar. We are also developing an *interpreting mode,* in which the grammar developer can test and debug his grammar without compilation, by sacrificing runtime efficiency.

In the speech translation system, we plan to make more use of contextual knowledge, and to expand the coverage of the system within our test domain. Eventually we hope to add other domains, perhaps including portions of those under active development for our text translation system.

I. Appendix I: Some Sample Runs of the Speech Parser

The actual output of our parser is shown in this appendix. The first subsection shows a run from an earlier version of the system, and the second subsection shows a run from the latest version. The earlier samples show that speech parsing can result in ambiguous parses with totally different meanings. (This implies a strong need for contextual analysis in future systems.) Note that the later version is more accurate, and automatically generates the English translation as (typed) output. In the near future, the system will be connected to DECTALK in order to produce synthesized speech output.

1.1. Earlier Version

The input phoneme sequence is "atomo ba itai", which was intended to be "atama ga itai" (which means "I have a headache."). The parser actually produced two parses with the highest scores, 104 and 94. The parser prints out, for each parse, its score, its supposed phoneme sequence and its f-structure.

```
1: (104)  A<2-3> T<4-5> A<6-7#8> M<8-9> A<10-11#8>
          G<12-13#8> A<14-15> I<16-17> T<18-19> A<20-21>
          I<22-23>

 ((:MOOD DEC) (SEM *HAVE-A-PAIN441)
  (OBJ((:WH -) (CASE GA) (SEM *HEAD) (ROOT ATAMA)))
  (CAUSATIVE -) (OBJ-CASE GA) (SUBJ-CASE GA)
  (SUBCAT 2ARG-GA) (CAT ADJ) (:TIME PRESENT) (ROOT ITAI))

2 : (94)   A<2-3> S<4-5#8> A<6-7#8> M<8-9> A<10-11#8>
           D<12-13#8> E<14-15#8> I<16-17> K<18-19#8> O<20-21#8>
           U<22-23#8>

 ((:MOOD DEC) (SEM *PTRANS453)
  (PPADJUNCT
    ((PART MADE) (SEM *TIME) (ROOT *TIME)
    (:DAY-SEGMENT ((:CFNAME *MORNING)))))
  (SUBJ-CASE GA) (CAUSATIVE -) (PASSIVE -) (SUBCAT INTRANS)
  (:TIME (*OR* PRESENT FUTURE)) (SIYOU +) (CAT V) (ROOT IKU))
```

Both parses are syntactically and semantically sound in the domain of doctor-patient conversation. The first parse "atamagaitai" means "I have a headache", which is the intended sentence. The second parse "asamadeikou" means something like "let us go until morning", which is also semantically (marginally) sound in the domain.

1.2. Latest Version

Using the same scoring system as the above, the later version produced preferred parses with substantially higher scores. Two sentences were spoken into the microphone. The sequence of phonemes after (tran) is what was outputted from the speech recognizer as its recognition of the input speech, which is then supplied to the phoneme-based GLR parser. The result of the parse (an f-structure) was then supplied to our generator package, Genkit, to produce the English. Note that the elapsed real time is within the acceptable limits suggested earlier in this paper (20 seconds or less.)

```
(tran)
ATAMA*A*UKI*JUKISURU
ACCEPTED !
Real time :   1.54 s
Run time:     0.48 s
 1 parses found
 1 : (186)  A<2-3> T<4-5> A<6-7> M<8-9> A<10-11>
             *<12-13> A<14-15> Z<16-17#8> U<18-19> K<20-21>
             I<22-23> Z<24-25#8> U<28-29> K<30-31> I<32-33>
            S<34-35> U<36-37> R<38-39> U<40-41>
-- Direct vers. —
((OBJECT ((LOCATION ((CFNAME *HEAD)))
 (CFNAME *PAIN) (PAIN-SPEC *THROBBING)))
  (CFNAME *HAVE-A-SYMPTOM)
  (MOOD DEC)
  (TENSE PRESENT))

0 BAD parses found
English : I HAVE A THROBBING PAIN IN THE HEAD
NIL
```

```
(tran)
ME*EZIRIZIRISURU
ACCEPTED!
Real time :   18.26 s
Run time :      0.54 s
 1 parses found
1 : (154) M<2-3> E<4-5> *<6-7> A<8-9#8> H<10-11#8> I<12-13>
           R<14-15> I<16-17> H<18-19#8> I<20-21> R<22-23>
           I<24-25> S<26-27> U<28-29> R<30-31> U<32-33>
-- Direct vers. —
 ((OBJECT ((LOCATION ((CFNAME *EYE)))
  (CFNAME *PAIN) (PAIN-SPEC *BURNING)))
  (CFNAME *HAVE-A-SYMPTOM)
  (MOOD DEC)
  (TENSE PRESENT))

0 BAD parses found
English : I HAVE A BURNING PAIN IN THE EYE
NIL
(dribble)
```

REFERENCES

[1] Massimo POESIO and Claudio RULLENT.
Modified Caseframe Parsing for Speech Understanding Systems.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* Milan, August, 1987.

[2] 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.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* Milan, August, 1987.

[3] MORII S., NIYADA K., FUJII S. and HOSHIMI M.
Large Vocabulary Speaker-independent Japanese Speech Recognition System.
In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP85).* March, 1985.

[4] HIRAOKA S., MORII S., HOSHIMI M. and NIYADA K.
Compact Isolated Word Recognition System for Large Vocabulary.
In *IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing (ICASSP86).* Tokyo, April 1986.

[5] NYBERG E.
FrameKit User's Manual.
Technical Report CMU-CMT-88-MEMO, Center for Machine Translation, Carnegie-Mellon University, 1988.

[6] SHIEBER S.M.
The Design of a Computer Language for Linguistic Information.
In *10th International Conference on Computational Linguistics,* pages 362-366. Stanford, July, 1984.

[7] HAYES P.J., HAUPTMANN A.G., CARBONELL J.G. and TOMITA M.
Parsing Spoken Language: A Semantic Caseframe Approach.
In *llth International Conference on Computational Linguistics (COLING86).* Bonn, U.K., August, 1986.

[8] TOMITA M.
Efficient Parsing for Natural Language.
Kluwer Academic Publishers, Boston, MA, 1985.

[9] TOMITA M.
An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition.
In *IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing (ICASSP86).* Tokyo, April, 1986.

[10] TOMITA M.
An Efficient Augmented-Context-Free Parsing Algorithm.
Computational Linguistics forthcoming, 1987.

NOTES

(1) Some portions of this paper are taken from the following papers :
"Linguistic and Domain Knowledge Sources for the Universal Parser Architecture" by Tomita M, Kee M., Mitamura T. and Carbonell J.G.; in Terminology and Knowledge Engineering. INDEKS Verlag, Frankfurt / M., 1987.
"Parsing Noisy Sentences" by Saito H. and Tomita M.; in proceedings of COLING88, Budapest, 1988.
We should also like to thank other members of the Center for Machine Translation for useful comments and advice. Funding for this project is provided by several private institutions and governmental agencies in the United States and Japan.

(2) Note that these LISP programs, as well as phrase structure rules, are generated automatically by the compiler.

(3) For more details on pseudo-unification, see the document "The Generalized LR Parser/Compiler Version 8.1: User's Guide", and its updates, available from the Center for Machine Translation at Carnegie Mellon.

(4) Phonemes (e.g. /g/, /a/, /s/, etc.) are even lower level units than syllables.

(5) We distinguish *noisy* from *ill-formed.* The former is due to recognition device errors, while the latter is due to human users.

Address : Computer Science Department and Center for Machine Translation;
Carnegie-Mellon University;
Pittsburgh, Pa 15213, USA.

Received: 3 November 1988.