MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

A.I.Memo No.677 May, 1982

PARSING AND GENERATING ENGLISH USING COMMUTATIVE TRANSFORMATIONS

Boris Katz and Patrick H. Winston

ABSTRACT: This paper is about an implemented natural language interface that translates from English into semantic net relations and from semantic net relations back into English. The parser and companion generator were implemented for two reasons: (a) to enable experimental work in support of a theory of learning by analogy; (b) to demonstrate the viability of a theory of parsing and generation built on commutative transformations. The learning theory was shaped to a great degree by experiments that would have been extraordinarily tedious to perform without the English interface with which the experimental data base was prepared, revised, and revised again. Inasmuch as current work on the learning theory is moving toward a tenfold increase in data-base size, the English interface is moving from a facilitating role to an enabling one. The parsing and generation theory has two particularly important features: (a) the same grammar is used for both parsing and generation; (b) the transformations of the grammar are commutative. The language generation procedure converts a semantic network fragment into kernel frames, chooses the set of transformations that should be performed upon each frame, executes the specified transformations, combines the altered kernels into a sentence, performs a pronominalization process, and finally produces the appropriate English word string. Parsing is essentially the reverse of generation. The first step in the parsing process is splitting a given sentence into a set of kernel clauses along with a description of how those clauses are hierarchically related to each other. The clauses are used to produce matrix and embedded kernel frames, which in turn supply arguments to relation-creating functions. The evaluation of the relation-creating functions results in the construction of the semantic net fragments.

This research was done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505.

Introduction

This paper presents an implemented natural language interface that translates from English into semantic net relations and from semantic net relations back into English. At present, the system enables natural language interaction at the Massachusetts Institute of Technology with Winston's learning system [Winston 1981] and at Stanford University with Binford's ACRONYM vision system [Binford 1982, Winston et al. 1982].

To illustrate the parsing and generating abilities of the interface, let us consider the following English sentence: "Othello did not want to kill Desdemona because he loved her." The parser translates this sentence into the following semantic network fragment:

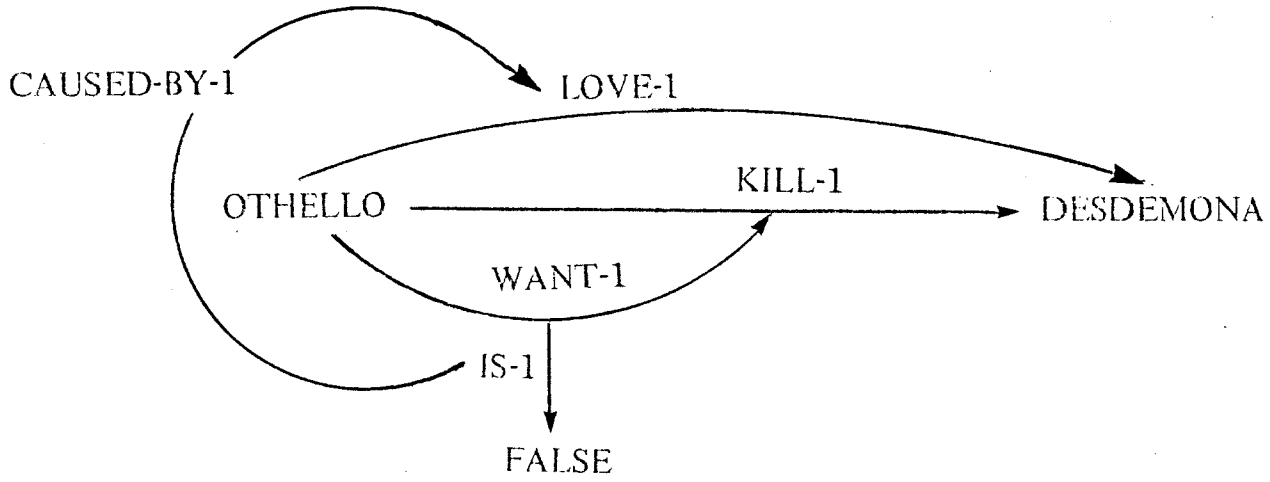

FIGURE 1: A semantic network fragment produced by the system after parsing the sentence "Othello did not want to kill Desdemona because he loved her."

Conversely, the language generation procedure, operating on CAUSED-BY-1 (see Figure 1), produces the original sentence.[1]

---

1. This example shows that the natural language system is invertible. However, no claim is being made that in general there should be a one-to-one correspondence between the semantic network fragment and the English output produced by the language generation procedure. For instance, there might be optional planning decisions depending on the values of certain discourse parameters. These questions will not be addressed in the present paper.

Purpose

The parser and companion generator were implemented for two reasons:

- To enable experimental work in support of a theory of learning by analogy;
- To demonstrate the viability of a theory of parsing and generation built on commutative transformations.

The learning theory is that of P. Winston [1980, 1981]. The theory was shaped to a great degree by experiments that would have been extraordinarily tedious to perform without the English interface by means of which the experimental data base was prepared, revised, and revised again. Inasmuch as current work on the learning theory is moving toward a tenfold increase in data-base size, the English interface is moving from a facilitating role to an enabling one. This marriage of work on learning with work on language is an example of the following trend:

- As researchers in Artificial Intelligence progress beyond toy-world domains, it becomes obvious that data bases prepared and accessed using English are necessary for research, not simply for research presentation.

The parsing and generation theory is that of B. Katz [1980]. The parsing and generation programs based on the theory have two particularly important features:

- The same grammar is used for both parsing and generation;
- In the generation process, the transformations of the grammar are commutative.

The paper is divided into two parts. First, we discuss how the language generation procedure converts network fragments into English sentences, establish the vocabulary relevant to the grammar, and give the necessary definitions. In the second part of the paper we describe the parsing process.

Representation

A semantic net is the common internal representation used for all three interacting subsystems: the parser, the learning system, and the generator. Each relation in the net is implemented as a *frame*, a *slot* in the frame, and a *value* in the slot. In such a structure, the *frame* position is occupied either by a noun or by an embedded relation; the *slot* is usually filled by a verb or a preposition; and the *value* can be filled by a noun, an adjective, or an embedded relation. Nodes and relations in the net are created using the function

RELATION. Consider, for instance, the following use of RELATION:

```
(RELATION OTHELLO
          WANT
          (RELATION OTHELLO KILL DESDEMONA))
```

Executing this expression causes the following list structures to be created:

```
(WANT-1 (FRAME (OTHELLO)) (SLOT (WANT)) (VALUE (KILL-1)))

(KILL-1 (FRAME (OTHELLO)) (SLOT (KILL)) (VALUE (DESDEMONA)))
```

In addition, a supplementary description node, a *comment frame,* is attached to the frame-slot-value combination and is used for identification and cross-referencing of relations:

```
(OTHELLO (KILL (DESDEMONA (KILL-1)))
         (WANT (KILL-1 (WANT-1))))
```

Generation Converts Network Fragments into Sentences

A network fragment becomes an English sentence in several steps. First, the fragment is divided into elements, taking into account the depth of embedding and certain values of the nodes. We will refer to these elements as *kernel* elements. Each of the kernel elements will ultimately be transformed into a matrix or an embedded clause in the resulting English sentence. In order to achieve this goal, we move into the next level in the internal representation of the natural language interface, the level of *kernel frames.* Thus, each kernel element of the semantic network is turned into a corresponding kernel frame.

A kernel frame is constructed from three types of instantiated templates: noun-template, verb-template, and adverb-template. The constituent structure of the templates is shown below:[2]

```
noun-template (NT) = (prep* det* adj* noun)

verb-template (VT) = (aux1 aux2 aux3 verb)

adverb-template (AT) = (mod adverb)
```

---

2. The template structures also provide information about the order of the constituents, and, therefore, act like phrase structure rules. Here *prep, det, adj, aux, mod* are, respectively, abbreviations for preposition, determiner, adjective, auxiliary, and modifier. The superscript * indicates that a string of one or more symbols or their conjunction is allowed.

Each template is implemented as an ordered list of pairs. The first element of the pair is the name of the constituent, the second is its value. All constituents are optional. If any constituent is absent, its value in the template is *nil*. Given below are some sample templates:

NT = ((prep nil) (det **a**) (adj (**young pretty**)) (noun **lady**))	"a young pretty lady"
NT = ((prep (**from out of**)) (det **the**) (adj nil) (noun **darkness**))	"from out of the darkness"
VT = ((aux1 **could**) (aux2 **have**) (aux3 nil) (verb **noticed**))	"could have noticed"
AT = ((mod **very**) (adverb **well**))	"very well"

The following two operations on templates are allowed:

1. Concatenation: `CONC (X1 X2) ---> X1 X2`

2. Conjunction: `CONJ (X1 X2) ---> X1` *conj* `X2`

where X1 and X2 are two templates of the same type, and *conj* is a conjunction such as *and, or*, etc. For instance, the noun-template representing the noun phrase "Othello and Desdemona" involves an application of the *conjunction* operation:

```
NT = ((prep nil) (det nil) (adj nil) (noun Othello) (conj and)
      (prep nil) (det nil) (adj nil) (noun Desdemona))
```

This template is also an example of a so called *prepositionless* noun-template, a noun-template that does not begin with a preposition.

Notice that in our grammar the noun-templates are used for constructing both noun phrases and prepositional phrases. The idea of having a prepositional constituent within a noun-template is not as far-fetched as it may seem. In languages with nominal case marking (like Russian), the case marking of noun phrases serves a semantic function similar to that of prepositions in English. A noun-template in such a language would have to incorporate a slot for recording the case of the noun phrase in order for its semantic role to be determined. Therefore, case marked noun phrases in many languages are used as counterparts of prepositional phrases in English. The decision to include the prepositional constituent into a noun-template reflects this linguistic generalization.

Each prepositionless noun-template in a sentence has a particular semantic role associated with it: *agent, goal*, or *theme*. This semantic role indicates how the noun phrase participates in the action denoted by the verb.

Agent is an entity that causes the action to occur.
Goal is the recipient or the beneficiary of the action.
Theme is an entity that undergoes a change of state or position.

The instantiated templates are combined to form a *kernel frame* with the following general structure:

$$NT^{initial}\ NT^{agent}\ VT\ NT^{goal}\ NT^{theme}\ NT^{final}$$

Here NT^{agent}, NT^{goal}, and NT^{theme} are noun-templates that play, respectively, the roles of *agent, goal,* or *theme* in a sentence. $NT^{initial}$ and NT^{final} are noun-templates that will be transformed later into the sentence initial and final prepositional phrases. In addition, adverb-template constituents can appear in a number of places: in the beginning, in the middle, or in the end of the kernel frame.[3] Most of the elements in the frame are optional.

For the network fragment anchored by IS-1 in Figure 1 (constructed from "Othello did not want to kill Desdemona"), the division process discovers two kernel elements: WANT-1 and KILL-1.

```
(WANT-1 (FRAME (OTHELLO)) (SLOT (WANT)) (VALUE (KILL-1)))

(KILL-1 (FRAME (OTHELLO)) (SLOT (KILL)) (VALUE (DESDEMONA)))
```

The instantiated templates that will make up the kernel frames in this case are simple, since most of the possible slots are unfilled:

```
NT = ((prep nil) (det nil) (adj nil) (noun Othello))
NT = ((prep nil) (det nil) (adj nil) (noun Desdemona))
VT = ((aux1 nil) (aux2 nil) (aux3 nil) (verb wanted))
VT = ((aux1 nil) (aux2 nil) (aux3 nil) (verb kill))
```

These templates are combined to form two kernel frames: a matrix kernel frame (MKF) and an embedded kernel frame (EKF). In order to construct sentences with embedded clauses, it is necessary that the matrix frame indicate the position where an embedded frame will be inserted. This position, the *joining point,* is marked by the word *it.*

```
MKF = ((NTagent ((PREP nil) (DET nil) (ADJ nil) (NOUN Othello)))
       (VT ((AUX1 nil) (AUX2 nil) (AUX3 nil) (VERB wanted)))
       (NTtheme ((PREP nil) (DET nil) (ADJ nil) (NOUN it))))

EKF = ((NTagent ((PREP nil) (DET nil) (ADJ nil) (NOUN Othello)))
       (VT ((AUX1 nil) (AUX2 nil) (AUX3 nil) (VERB kill)))
       (NTtheme ((PREP nil) (DET nil) (ADJ nil) (NOUN Desdemona))))
```

Once kernel frames are constructed, they are converted into *transformation frames* in preparation for the application of transformations. This involves conversion of the

---

3. The adverb-templates will be ignored in the rest of this paper.

noun-templates into word strings, separation of each auxiliary verb from its affix, and the insertion of certain dummy slots that will be used by certain transformations.

The noun phrases of the transformation frame are derived from the noun-templates of the kernel frame. Each noun phrase indicates one of three fixed positions in the transformation frame: NP_1-position, $NP_{1.5}$-position, or NP_2-position. Each noun phrase is associated with one of the prepositionless noun-templates NT^{agent}, NT^{goal}, or NT^{theme} in the kernel frame. We will assume that the noun phrases NP_1, $NP_{1.5}$, and NP_2 initially obtain their values from the templates NT^{agent}, NT^{goal}, and NT^{theme}, respectively.[4] If only two noun-templates NT^{agent} and NT^{theme} are present in the kernel frame the corresponding transformation frame will have only two NP-positions: NP_1 and NP_2.[5]

A special procedure, *affix stripping*, is performed on the auxiliary elements of the verb-template to separate each auxiliary verb from its associated affix. The affixes of the auxiliaries Modal, HAVE, and BE are, respectively, *0*, *-en*, and *-ing*. The notation reflects the fact that, in an English sentence, the auxiliary determines the form of the following verb: an infinitive follows a modal, the past participle follows HAVE, and the progressive form of the verb follows BE. If the sentence has no auxiliary verbs in it, the auxiliary *do* (without an affix) is inserted as a value of the AUX1.

Several dummy elements are inserted into the transformation structure. These are left unspecified until the appropriate transformation activates them and assigns the necessary values. The elements NEG1 and NEG2 that are used for the contracted and full forms of English negatives are inserted after the first auxiliary verb. The dummy element COMP is inserted at the beginning of the structure and the element INFL, before the first auxiliary verb.[6] They will be used in constructing different embedded clauses. In our example, the matrix transformation frame (MTF) and the embedded transformation frame (ETF) have the following form:

```
MTF = ((COMP comp) (NP1 (Othello)) (TENSE past) (INFL infl)
       (AUX1 do) (NEG1 neg1) (NEG2 neg2) (VERB want) (NP2 (it)))
```

---

4. By values we mean the actual word string derived from a noun template plus the information about its semantic role.

5. This is the reason for choosing the obscure indices 1, 1.5, and 2 for the noun phrases.

6. In the implementation, the values of unspecified elements NEG1, NEG2, COMP, and INFL are names of those elements.

```
ETF = ((COMP comp) (NP1 (Othello)) (TENSE past) (INFL infl)
       (AUX1 do) (NEG1 neg1) (NEG2 neg2) (VERB kill) (NP2 (Desdemona)))
```

The network fragment also determines which transformations should be applied. Once the transformation frames are prepared, transformations are applied. In the current implementation, there are 21 possible transformations. Some of these, the *connective* transformations,[7] prepare a matrix transformation frame and an embedded transformation frame for combination, while other transformations, such as negation, passivization, or *there* insertion, apply to only one frame.

In the example, the transformation *NOT* is applied to the matrix frame, along with a connective transformation, *0-0-TO*, which is applied to the embedded frame. This particular choice of connective transformation was made by taking into account the type of verb in the matrix clause, the kind of branching, and the identity of subject noun phrases in the main clause and in the embedded clause. Here are the altered transformation frames:

```
MTF = ((COMP comp) (NP1 (Othello)) (TENSE past) (INFL infl)
       (AUX1 do) (NEG1 neg1) (NEG2 not) (VERB want) (NP2 (it)))

ETF = ((COMP 0) (NP1 nil) (TENSE past) (NEG1 neg1) (NEG2 neg2)
       (INFL to) (AUX1 do) (VERB kill) (NP2 (Desdemona)))
```

Once all transformations have been applied, the altered transformation frames are attacked by four obligatory *adjustment* operations:

1. *Garbage-deletion* removes all unspecified elements;

2. *Do-deletion* deletes the auxiliary *do* when it immediately precedes a verb;

3. *Affix-hopping* recognizes situations in which verbs need affixes attached;

4. *N't-hopping* recognizes situations in which auxiliary verbs need *n't* attached.

It was shown in [Katz 1980] that the following two constraints must be imposed on the adjustment operations to generate a correct English sentence.

> *Separation Constraint*: The adjustments must be executed only after all the transformations have been applied.

---

7. The connective transformations are described in the next section.

Ordering Constraint: The adjustments must be applied in the order in which they are listed above.

In our example, all of the adjustments except *N't-hopping* have an effect, producing the following result:

```
MTF = ((NP1 (Othello)) (AUX1 did) (NEG2 not) (VERB want) (NP2 (it)))

ETF = ((INFL to) (VERB kill) (NP2 (Desdemona)))
```

Reading off the values in the adjusted transformation frames and substituting the embedded frame ETF at the joining point indicated by *it* in the matrix frame MTF, we have the final English form, "Othello did not want to kill Desdemona."

Thus, we have introduced two different classes of operations on the transformation frame: the optional transformations and the obligatory, intrinsically ordered adjustments, which are always activated after the last transformation has been applied in order to obtain the necessary English output. In contrast to adjustments, transformations are part of a planning vocabulary, that is, a decision must be made each time about the application of any of the transformations. Adjustments are meaning-preserving, purely syntactical operations on the transformation frame. Transformations, on the contrary, help determine the meaning or focus the emphasis of a kernel sentence. This separation of all syntactical operations into two distinct classes (transformations and adjustments) plays an important role in constructing our system. It isolates the semantically empty component of the generation process and clears the way for making the semantically relevant part of the system commutative.

By the commutativity of a set of transformations, we mean that any two transformations can be applied in either order, and both orderings produce the same result.[8] Commutativity makes system implementation and maintenance easier because no attention needs to be paid to transformation ordering and interaction.

Modularity is an important feature of the generation system. The production of kernel frames and the list of transformations to be applied are determined from the semantic network alone. This is completely independent of the production of transformation frames and the transformation process itself. Therefore, the bulk of the system is useful in other applications.

---

8. We assume here that all the sentences involved belong to the *domain of definition* of the corresponding transformations (see [Katz 1980] for details).

To see some of the real power of the generator, we will illustrate the effect of applying particular transformations. Suppose that the transformations were specified for the kernel and embedded transformation frames which correspond to the following kernel sentences:

This man asked it.
Othello killed Desdemona.

Given these kernels, the program can apply different sets of transformations sequentially to alter the individual sentences or combine them. Among the possible outputs are:[9]

1. After 0-NP_1-TO_1: This man asked Othello to kill Desdemona.

2. After QUESTION: Did this man ask Othello to kill Desdemona?

3. After NOT_1: Did this man ask Othello not to kill Desdemona?

4. After PASSIVE: Was Othello asked by this man not to kill Desdemona?

5. After N'T: Wasn't Othello asked by this man not to kill Desdemona?

Connective Transformations

Connective transformations are used to construct sentences with embedded clauses. In order to form such sentences, the system needs two kernels as input. Let us denote the corresponding transformation frames as MTF and ETF. They will be used as the basis for the matrix and the embedded clauses. The application of a connective transformation to the ETF produces an altered frame ETF^*, which is in the form appropriate for an embedded clause. Other transformations may change the frame MTF into MTF^*. After all the transformations have been applied, the two altered frames are combined by substituting the embedded frame ETF^* at the joining point *it* in the matrix frame MTF^*. A connective transformation is defined by three parameters, each referring to a position in the embedded transformation frame: the complementizer COMP, the first noun phrase NP_1, and the complementizer INFL. COMP indicates the element that introduces the embedded clause. It can receive one of the following values: {POSS, THAT, FOR, 0}.[10]

---

9. For convenience, we append "1" to the name of a transformation that will be applied to an embedded clause, and "2" to a transformation that will be applied to a relative clause. Nothing is appended to the matrix clause. The significance of the names of transformations, such as 0-NP_1-TO, will be explained below.

10. 0 means that there is no overt affix in this position.

The parameter NP_1 takes on one of two values: {NP_1, 0} indicating whether or not the first NP of the embedded clause is present in the resulting sentence. These values encode whether the subject of the matrix clause is coreferent with the subject of the embedded clause. INFL is considered to be an affix specifying how the verb in the embedded clause should be inflected. It can have one of four values: {ING, INF, TO, 0}. Since a particular set of values of the three parameters COMP, NP_1, and INFL is sufficient to determine the form of the embedded clause completely, we will use them to form the names of the Connective Transformations. Each name is represented as a triple consisting of the current values of the parameters in the following order: COMP-NP_1-INFL. A general description of the action of a connective transformation can be stated as follows:

(a) Current values of the parameters COMP and INFL are inserted into the transformation frame;
(b) If the value of the parameter NP_1 is 0, the element NP_1 is removed from the structure;
(c) If the value of INFL is not 0, elements NEG1 and NEG2 are moved before TENSE.

An example of the application of each of ten different connective transformations is given in Table 1, below:[11]

TABLE 1

Transformation	*Matrix Clause*	*Embedded Clause*	*Sentence*
THAT-NP_1-0	It angered Iago	Othello ignored the letter	That Othello ignored the letter angered Iago
THAT-NP_1-INF	Iago suggests it	Cassio is silent	Iago suggests that Cassio be silent
0-NP_1-INF	Iago watched it	Othello read the letter	Iago watched Othello read the letter
0-NP_1-0	Iago knows it	Othello ignored the letter	Iago knows Othello ignored the letter
FOR-NP_1-TO	It angered Iago	Othello ignored the letter	For Othello to ignore the letter angered Iago
0-NP_1-TO	Cassio asked it	Iago wrote the letter	Cassio asked Iago to write the letter
0-0-TO	Iago claims it	Iago has written the letter	Iago claims to have written the letter
POSS-NP_1-ING	It shocked Iago	Othello ignored the letter	Othello's ignoring the letter shocked Iago
0-NP_1-ING	Iago saw it	Othello read the letter	Iago saw Othello reading the letter
0-0-ING	It angered Othello	Othello read the letter	Reading the letter angered Othello

11. The computer output does not always adhere to historical reality.

The main verb of the matrix clause determines which kind of clause may be embedded under it, and, therefore, the kind of connective transformation that may be applied in each particular case. Consequently, the dictionary entry for any verb which may appear in a matrix clause must contain a list of permissible transformations.

A family of ten transformations for generating embedded clauses was introduced in this section. This family is completely defined by the values of three parameters COMP, NP_1, and INFL and, therefore, can be considered a single *connective transformation* whose surface manifestation has several different forms depending on the values of these parameters.

In summary, the language generation procedure converts a network fragment into kernel frames, chooses the set of transformations that should be performed on each frame, and then executes the specified transformations, combines the altered kernels into a sentence, performs a pronominalization process,[12] and finally produces the appropriate English word string.

Parsing is the Reverse of Generation

The first step in the parsing process involves splitting a given sentence into a set of kernel clauses along with a description of how those clauses are hierarchically related to each other. For instance, identifying the type of the verb and the values of complementizers COMP and INFL in the sentence appears to be sufficient for reconstructing the kernels (matrix clause and embedded clause) which form every complex sentence in Table 1.

Consider again the sentence "Othello did not want to kill Desdemona because he loved her." In this example, the *because*-clause is detached first. Now, since the remaining sentence consists of a matrix clause and an embedded clause, the problem is to determine the connective transformation involved and the position of the joining point. This position is determined through an examination of the branching structure of the sentence. The analysis of the "want to kill" construction leads to the conclusion that a right-branching connective transformation *0-0-TO* (see Table 1) is involved. As a result, the parser produces the following kernel clauses:

---

12. Both the parser and the generator include the pronominalization process (see Appendix for examples). Pronominalization, however, will not be discussed in this paper.

Othello did not want it.
Othello kill Desdemona.
Othello loved Desdemona.

The parser analyzes every word in the input kernel sentence, scanning it from left to right and mapping the appropriate pieces of the word string onto the corresponding noun-template, verb-template, or adverb-template. The templates must be filled out from left to right. If any element in the template is left unspecified, the parser inserts *nil* as the value of this element. When all the elements of the template are filled, the parser, depending on the next word and its place in the sentence, either begins to fill out the elements of another template or continues the construction of the current *complex* template using one of the operations of *concatenation* or *conjunction*. The procedure described converts the kernel clauses in the example into the matrix and embedded kernel frames below:

```
MKF1 = ((NTagent ((PREP nil) (DET nil) (ADJ nil) (NOUN Othello)))
        (VT ((AUX1 did) (NEG not) (AUX2 nil) (AUX3 nil) (VERB want)))
        (NTtheme ((PREP nil) (DET nil) (ADJ nil) (NOUN it))))

EKF1 = ((NTagent ((PREP nil) (DET nil) (ADJ nil) (NOUN Othello)))
        (VT ((AUX1 nil) (AUX2 nil) (AUX3 nil) (VERB kill)))
        (NTtheme ((PREP nil) (DET nil) (ADJ nil) (NOUN Desdemona))))

MKF2 = ((NTagent ((PREP nil) (DET nil) (ADJ nil) (NOUN Othello)))
        (VT ((AUX1 nil) (AUX2 nil) (AUX3 nil) (VERB loved)))
        (NTtheme ((PREP nil) (DET nil) (ADJ nil) (NOUN Desdemona))))
```

These kernel frames supply arguments to instances of RELATION, the relation-creating function mentioned earlier. In the case of a simple transitive sentence like "Othello kills Desdemona", the nouns and the verb of the kernel frame provide the values for the arguments:[13]

```
(RELATION OTHELLO KILL DESDEMONA)
```

If a connective transformation is involved, as in the sentence "Othello wants to kill Desdemona", the function RELATION must be applied again using the embedded relation `(RELATION OTHELLO KILL DESDEMONA)` as its argument:

```
(RELATION OTHELLO
          WANT
          (RELATION OTHELLO KILL DESDEMONA))
```

13. Tense information, supplied by the parser, is not carried into the semantic network representation. Currently, the learning program experiments do not use time.

The position of the embedded relation is determined by the type of branching found in the structure. Here the embedded relation is used as the third argument to the function RELATION because the first step of the parsing process determines that the sentence has a right-branching structure. In the case of a left-branching sentence, "That Othello killed Desdemona is terrible," the embedded relation will be the first argument:

```
(RELATION (RELATION OTHELLO KILL DESDEMONA)
          IS
          TERRIBLE)
```

For our first sample sentence, "Othello did not want to kill Desdemona because he loved her," the parser produces the following output:

```
(RELATION (RELATION (RELATION OTHELLO
                              WANT
                              (RELATION OTHELLO KILL DESDEMONA))
                    IS
                    FALSE)
          CAUSED-BY
          (RELATION OTHELLO LOVE DESDEMONA))
```

The evaluation of this expression results in the construction of the semantic net fragment shown at the beginning of the paper in Figure 1.

To demonstrate the parser's capabilities, a number of English sentences successfully analyzed by the parser are listed in the Appendix, together with the system's output in the form of relation-creating functions. The examples include a variety of syntactic constructions. Among them are sentences with multiple-embedding, conjoined noun phrases, possessive noun phrases, negation, prepositional phrases, relative clauses, raising, cleft-sentences, passive sentences, and other syntactic constructions.

Summary

Stages in language generation and parsing, as exhibited by an implemented system, have been shown. Parsing is essentially the reverse of generation. Most of the system is independent of the semantic network representation involved in the examples, facilitating its use in other applications. This independence is important because the system was built with a view toward producing an interface enabling other research efforts, rather than strictly for its own sake.

Acknowledgments

This paper was improved substantially by comments from Beth Levin, Robert Berwick, and Randy Davis.

Appendix

Example 1:

One application of the natural language system is to prepare semantic networks from abstracts for use in information-retrieval experiments. This summary of *Macbeth* is a typical abstract:

> MA is a story about Macbeth, Lady-macbeth, Duncan, and Macduff. Macbeth is an evil noble. Lady-macbeth is a greedy ambitious woman. Duncan is a king. Macduff is a prince. Lady-macbeth persuades Macbeth to want to be king because she is greedy. She is able to influence him because he is married to her and because he is weak. Macbeth murders Duncan with a knife. Macbeth murders Duncan because Macbeth wants to be king and because Macbeth is evil. Lady-macbeth kills herself. Macduff is angry. Macduff kills Macbeth because Macbeth murdered Duncan and because Macduff is loyal to Duncan. Remember MA.

Macbeth is retrieved when the system is asked to find a situation similar to the following:

> Let E be an exercise. E is a story about a weak noble, a greedy lady, a loyal prince, and a king. The lady is able to influence the noble because he is married to her. The noble murders the king because the noble wants to be king. The prince kills the noble because the prince is loyal to the king.

The actual terminal interaction for the example above, highlighting the system's generation ability, is as follows:

```
==> find  a  situation  similar  to  exercise-1.

You asked for a situation similar to EXERCISE-1.  MA is similar to
EXERCISE-1 because:

IN MA THERE IS A WOMAN, LADY-MACBETH, WHO IS ABLE TO INFLUENCE MACBETH
BECAUSE HE IS MARRIED TO HER.  THERE IS A PRINCE, MACDUFF, WHO KILLS
HIM BECAUSE MACDUFF IS LOYAL TO DUNCAN.  MACBETH MURDERS DUNCAN BECAUSE
MACBETH WANTS TO BE KING.
```

Example 2:

The generator is also used to describe if-then rules produced in learning experiments. For instance, one experiment produces an if-then rule that is represented internally in gobbledygook like this:

```
(RULE-2 (AKO (RULE (AKO-62)))
        (PART (MAN-10 (PART-18)))
        (THEN (AKO-64 (THEN-3)))
        (IF-PLAUSIBLE (AKO-64 (IF-PLAUSIBLE-1))
                      (IS-47 (IF-PLAUSIBLE-2))
                      (TO-8 (IF-PLAUSIBLE-3)))
        (IF (IS-49 (IF-3)) (AKO-65 (IF-4)) (AKO-63 (IF-5)))
        (CASE (STORY-1 (CASE-1))))
```

Translated into English, the same rule becomes transparent:

> Rule-2 concerns a man. If the man is not married and he is an adult, then he is a bachelor, assuming he is expected to be married and he is able to be married.

Example 3:

In another application, the system is used to prepare semantic network in for learning experiments. Some of these experiments involve functional descriptions of objects:

> Let X be a story. X is a story about a person, an object, and some soup. The object is a spoon because the person uses it to drink the soup, which is food. The person can use the object to eat the soup because the object is graspable and because it holds the soup. The spoon is graspable because it has a long and thin handle. The spoon holds the soup because the spoon's end is an open cavity.

Example 4:

Finally, here are the results of parsing an assortment of sentences. The function `(GET-INSTANCE MAN)` creates and returns `MAN-1` if there are no men yet in the current story; otherwise `(GET-INSTANCE MAN)` returns the man most recently mentioned in the story.

The man wasn't expected to help the woman with a loan.

```
(RELATION (RELATION (RELATION (RELATION (GET-INSTANCE MAN)
                                        HELP
                                        (GET-INSTANCE WOMAN))
                              WITH
                              (GET-INSTANCE LOAN))
                    IS
                    EXPECTED)
          IS
          FALSE)
```

It has been shown using an analogy that a man who Mary loved did not want to help Bill because Bill was loyal to her.

```
(RELATION MARY LOVE (GET-INSTANCE MAN))

(RELATION
 (RELATION (RELATION (RELATION (RELATION (GET-INSTANCE MAN)
                                         WANT
                                         (RELATION (GET-INSTANCE MAN)
                                                   HELP
                                                   BILL))
                               IS
                               FALSE)
                   CAUSED-BY
                   (RELATION (RELATION BILL IS LOYAL)
                             TO
                             MARY))
           IS
           SHOWN)
 USING
 (GET-INSTANCE ANALOGY))
```

John knew about the man's desire to find the woman.

```
(RELATION JOHN
          KNOW
          (RELATION (GET-INSTANCE MAN)
                    DESIRE
                    (RELATION (GET-INSTANCE MAN)
                              FIND
                              (GET-INSTANCE WOMAN))))
```

Tom and Bill knew that it has been proved that John found her.

```
(RELATION TOM
          KNOW
          (RELATION (RELATION JOHN FIND (GET-INSTANCE WOMAN))
                    IS
                    PROVED))

(RELATION BILL
          KNOW
          (RELATION (RELATION JOHN FIND (GET-INSTANCE WOMAN))
                    IS
                    PROVED))
```

The man's love for the woman forced him to find her.

```
(RELATION (RELATION (GET-INSTANCE MAN)
                    LOVE
                    (GET-INSTANCE WOMAN))
          FORCE
          (RELATION (GET-INSTANCE MAN) FIND (GET-INSTANCE WOMAN)))
```

John was known by Mary to have helped Bill's friend.

```
(RELATION (GET-INSTANCE FRIEND BILL)
          RELATED-TO
          BILL)

(RELATION (RELATION (RELATION JOHN HELP (GET-INSTANCE FRIEND BILL)
                    IS
                    KNOWN)
          BY
          MARY)
```

References

Binford, Thomas O., "Survey of Model-based Image Analysis Systems," *Robotics Research*, vol. 1, no. 1, Spring, 1982.

Katz, Boris, "A Three-Step Procedure for Language Generation," M.I.T. Artificial Intelligence Laboratory Memo No. 599, December 1980.

Winston, Patrick Henry, "Learning and Reasoning by Analogy," *CACM*, vol. 23, no. 12, December, 1980.

Winston, Patrick Henry, "Learning New Principles from Precedents and Exercises: the Details," M.I.T. Artificial Intelligence Laboratory Memo No. 632, May 1981.

Winston, Patrick Henry, Thomas O. Binford, Boris Katz, and Michael Lowry, "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," M.I.T. Artificial Intelligence Laboratory Memo No. 679, May 1982.