Proceedings of the 1st RIAO Conference on User-Oriented
Content-Based Text and Image Handling (RIAO '88), 1988.

Using English for Indexing and Retrieving

Boris Katz
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

Historically, knowledge bases for artificial intelligence programs (and data bases for large computer systems) have been constructed by hand. This process requires a lot of time and effort. If, instead, knowledge bases were specified in English (or any other natural language), and then *automatically* transformed into an appropriate formal representation, the task of constructing, modifying, and querying the knowledge base would be greatly simplified. This is the role of START, a portable natural language processing system that enables efficient indexing and retrieving of information.

START (SynTactic Analysis using Reversible Transformations) consists of two modules. The *understanding* module analyzes English text and indexes the knowledge contained within it. The *generating* module produces English text from retrieved portions of the knowledge base. The user gains access to information stored in a knowledge base by querying it in English. The system then retrieves the requested information and frames its response also in English.

Researchers at MIT, Stanford University, and the Jet Propulsion Laboratory, have benefited from START in a variety of systems, *e.g.*, Winston's learning program [1982; 1984; Winston *et al.*, 1983], Doyle's causal modelling system [1984; 1988], and Katz and Brooks' spacecraft sequencing system [1987].

1. Understanding Language

Before we provide a detailed description of the START system we should make clear what we mean by "understanding." What does it mean for a machine to understand language?

Let us consider a situation where a mother gives instructions or tells a story to her daughter Jill. Hopefully, this child has "stored" the new information/knowledge in her memory. In this case, we say that the child understood her mother.

How can this be verified?

- If Jill heard a story, her mother can ask questions relevant to the story. If after searching her memory, Jill is able to utilize the acquired knowledge and answer the questions correctly, then she understood the story.
- Suppose instead of a story Jill heard a set of instructions for a task she is supposed to perform. If Jill is able to retrieve the knowledge given by the instructions and accomplish the task, then she understood the instructions.

We will use these two criteria as "Turing tests" to help us define what it means for a computer to understand English:

A. English text is typed into the computer and on the basis of this text a knowledge base is created. The user queries the knowledge base in English. If the computer's responses are correct, then we can say the computer understood the text.

B. A sequence of English commands or instructions is entered in the computer. If the computer carries out appropriate actions in response to them, then we can say the computer understood the instructions.

Remarkably, START passes these two tests in a variety of situations although it possesses no explicit theory of meaning.

As suggested by these two criteria, knowledge bases created by START can either be used in question-answering situations or they can provide input data for other computer systems. The START knowledge base is employed by both the language understanding and generating modules. These two modules also share the same Grammar (see Katz [1980], Katz and Winston [1982]).

2. Kernel Sentences

Most English sentences break up into units that we will call *kernel sentences*. Before we can formally define these units let us examine how START represents sentence elements internally. The system uses three types of building blocks for constructing a kernel sentence, the *noun-template*, the *verb-template*, and the *adverb-template*:

noun-template (NT) = [prep det mod adj* noun]

verb-template (VT) = [aux1 neg aux2 aux3 verb]

adverb-template (AT) = [mod adverb]

Here *prep, det, mod, adj, aux, neg* are, respectively, abbreviations for preposi-

tion, determiner, modifier, adjective, auxiliary, and negation. The superscript * indicates that a string of one or more symbols or their conjunction is allowed. All the elements in the templates are optional. Noun phrases and prepositional phrases in English can be constructed by reading off the slot values in instantiated noun-templates. In a similar way, verbs and their auxiliaries can be obtained by reading off slot values in verb-templates. For example, the well-formed instantiation of the noun-template, NT = [(noun **Mary**)], where most elements of the NT are omitted, produces a simple noun phrase *Mary*, but the same template with all its elements filled,

NT = [(prep **after**) (det **a**) (mod **very**) (adj **long**) (noun **flight**)]

generates a full-fledged prepositional phrase *after a very long flight*. Similarly, the verb-template may produce either one main verb, *launch*, when using VT = [(verb **launch**)], or a more complex string like *could have been watching*:

VT = [(aux1 **could**) (aux2 **have**) (aux3 **been**) (verb **watching**)]

Now let us define a *kernel structure* as the following sequence of templates:

(1) $NT^{initial}\ NT^{subject}\ VT\ NT^{object}\ NT^{final}$

Here $NT^{subject}$ and NT^{object} are noun-templates that represent the subject and the object in the sentence; $NT^{initial}$ and NT^{final} represent its initial and final prepositional phrases; VT is a verb-template.

We should point out that most of the elements in the kernel structure are optional. Also for simplicity the adverb-templates have been omitted although they may appear in (1) in a number of places. In addition, transformational rules, introduced in section 5, are allowed to modify the kernel structure and change the order of templates.

A *kernel sentence* is an English sentence obtained by reading off from left to right all slot values in all templates in (1). We define *parsing* as a process of syntactic analysis which, given an English kernel sentence as input, produces the corresponding kernel structure as output.[1] For example, given the sentence below:

> After the launch the commander will give additional instructions to the astronaut

the *parser* produces the following instantiated kernel structure where the order of templates follows that of (1):

[1] In sections 5 and 6 this notion of parsing is extended to a wider class of English sentences. See also Katz and Brooks [1987] and section 11 for examples from the "real world".

$NT^{initial}$	[(prep **after**) (det **the**) (noun **launch**)]
$NT^{subject}$	[(det **the**) (noun **commander**)]
VT	[(aux1 **will**) (verb **give**)]
NT^{object}	[(adj **additional**) (noun **instructions**)]
NT^{final}	[(prep **to**) (det **the**) (noun **astronaut**)]

Many natural language understanding systems restrict themselves to the parsing process just defined and stop there. However, this is clearly not enough to satisfy our definition of understanding. It is not enough to teach the computer to recognize different syntactic categories and fill in the slots in the kernel structure. Our goal is to enable the computer to use the knowledge encoded in the kernel structure; in other words, to *index* and *retrieve* this knowledge efficiently.

3. From Kernel Structures to T-expressions

Recall that in order to understand her mother, Jill had to perform two important operations. When listening to the story, Jill had to store (or *index*) the new knowledge in her memory. When answering the questions, she had to search her memory and *retrieve* the knowledge. These two operations, indexing and retrieving, are crucial in our model of understanding language. In this section we will describe the indexing procedure of START. The retrieval task is carried out by a matching procedure described in section 10.

Suppose we type the following English sentence on a computer terminal:

(2) Jane will recognize Paul tomorrow

and the parsing procedure constructs the appropriate kernel structure. Now, there are many things about this sentence that the computer should remember: that *Jane* is the subject of the sentence, *Paul* is the object, that *recognize* is the relation between them. There is more to remember: the tense and the aspect of the sentence, its auxiliaries, its adverbs. Was this sentence embedded in a larger sentence? Does it have a relative clause? Was the verb in the active or passive form? We certainly want all this information about the sentence to be stored in the computer's memory. However, we also want to be able to retrieve this information efficiently.

We could store all these sentence features in one long list. This approach, however, would not account for the fact that some of the features in a sentence seem more salient than others. A simple list of features in sentence (2) would also fail to capture its structural affinity with the following two sentences:

(3) Yesterday Jane could have recognized Paul.

(4) Paul wasn't recognized by Jane.

And finally, this approach would turn the matching/retrieval task into a computational nightmare.

We could try to emphasize the hierarchical nature of the English sentence by using the kernel structure representation. However, since most elements of the kernel structure are optional, its shape is too unpredictable to allow the system to match the kernel structures efficiently.

Our system, START, rearranges the elements of the kernel structure into embedded *ternary expressions (T-expressions)* by tying together the three most salient parameters of a sentence, the subject, the object, and the relation between them, ⟨**subject relation object**⟩. All three sentences (2, 3, 4) will yield the same *T*-expression

(5) ⟨**Jane recognize Paul**⟩.

Certain other parameters are used to create additional *T*-expressions in which prepositions and several special words serve as relations. The remaining parameters, adverbs and their position, tense, auxiliaries, voice, negation, *etc.*, are recorded in a representational structure called *history*. The history has a *page* pertaining to each sentence which yields the given *T*-expression. When we index the *T*-expression in the knowledge base, we cross-reference its three components and attach the history *H* to it. The resulting entry in the knowledge base, denoted ⟨**subject relation object**⟩$_H$, will be called a *T-entry*. For example, the *T*-entry ⟨**Jane recognize Paul**⟩$_H$ corresponding to *T*-expression (5) has a three-page history, assuming that all three sentences (2, 3, 4) appeared in the input text. One can thus think of a *T*-entry as a "digested summary" of the syntactic structure of English sentences.

The *T*-entry is the cornerstone of the representational hierarchy of the START system. It is the level of the hierarchy where the understanding and the generating modules meet. The understanding module analyzes English sentences and creates a set of *T*-entries. The generating module, in turn, retrieves these *T*-entries from the knowledge base and produces English text.

4. Referents for Noun Phrases

The subject and the object of *T*-expression ⟨**Jane recognize Paul**⟩ are proper names which are taken directly from sentence (2). However, the process is more complex if, for example, the subject of a sentence is not a proper name but a complex noun phrase:

(6) Jane's good friend from Boston recognized Paul.

In sentence (6) the system needs to establish the referent for the head noun, *friend*. The system has to come up with a *unique name* for this noun in case a different instance of *friend* appears later in the analyzed text. In order to do this, START computes the *name environment* E_1 for this occurrence of *friend*. We define E_1 as a list of adjectives (in this case, *good*), possessive nouns (*Jane's*), prepositional phrases (*from Boston*), *etc.* modifying that noun in the present sentence. Then START associates with this environment a unique name, say *friend-1*, which we will call a *referent*[2] for the noun *friend* in the environment E_1. The main T-expression for sentence (6) will therefore take the form **(friend-1 recognize Paul)**.[3]

The analyzed noun, *friend*, its name environment E_1, and its referent, *friend-1* are then recorded in the computer's memory. This bookkeeping gives the system the ability to compute efficiently the referent of a noun given its name environment. If, for instance, the same noun phrase,

Jane's good friend from Boston

occurs again in a different sentence later in the text, then its name environment would coincide with E_1 and hence the same referent, *friend-1*, would be retrieved and utilized in the T-expressions constructed for this sentence.

Suppose now that START encounters a new sentence where the noun *friend* appears in a slightly different noun phrase like

(7) Tracy's good friend from Pasadena.

The environment E_2 associated with the new noun phrase is different from E_1 and is not to be found in the computer memory. This means that there is no referent readily available for the noun *friend* in this sentence and the system needs to generate a new unique name, *friend-2*, to be associated with E_2.

START recursively employs the procedure just described to find a referent for *every* noun in the sentence. Thus, given the noun phrase

The young woman's good friend from the big city

the system first determines the referents of the nouns *woman* and *city*. Only after that, once the computation of its name environment becomes possible, does the head noun, *friend*, get its referent.[4]

Sometimes, however, the information in the name environment is not

[2]In calling a unique name a referent we deviate from standard usage, which reserves the term for an object in the world.

[3]The analysis of the subject noun-phrase in sentence (6) will produce three additional T-expressions (see section 6).

[4]In the remainder of this paper, for reasons of simplicity, we will use the nouns themselves in T-expressions rather than their referents.

sufficient to find referents for noun phrases. For instance, if a noun phrase is modified by a relative clause (see section 6) the entire knowledge base has to be consulted in order to determine the appropriate referent.

5. Transformational Rules

The standard kernel structure introduced in section 2,

(1) $NT^{initial}\ NT^{subject}\ VT\ NT^{object}\ NT^{final}$

allows the system to generate or parse only a limited variety of English sentences. To account for other kinds of sentences, START employs commutative *transformational rules* (see Chomsky [1957], Katz [1980]). For instance, consider how the kernel sentence

The probe reached Venus

is modified by several transformational rules, where each transformation is applied to the outcome of the previous one:

Transformation	Sentence
	The probe reached Venus.
Question	Did the probe reach Venus?
Negation	Didn't the probe reach Venus?
Passive	Wasn't Venus reached by the probe?

The transformations shown are executed by the generating module of START. In the understanding mode, the system's goal is to recognize which transformations were applied. In some cases, for instance, *Negation*, START simply makes the appropriate additions to the histories of the resulting T-entries. In other cases, the system must actually *reverse* the effect of the transformation.

All the examples of English sentences considered so far have been very simple. We can make them a little more complex by allowing simpler sentences to be embedded in larger sentences, as shown below:

(8) Spock wanted the computer to print the message

(9) For Spock to ignore the command would anger Kirk.

The transformational rules responsible for sentence embedding form a special class called *connective transformations* (see Katz [1980]). Each connective transformation takes two kernel sentences as input; these correspond to the *matrix* clause and the *embedded* clause in the resulting English sentence. For example, sentence (8) above consists of a matrix clause, *Spock wanted it*, and an embedded clause, *the computer printed the message*. We assume that one

of the noun-templates in the kernel structure of the matrix clause always contains *it* as a *joining point* for glueing the two kernels together. Table 1 shows examples illustrating the application of several different connective transformations.

Matrix clause	Embedded clause	Resulting Sentence
It angered Kirk	The computer ignored the message	That the computer ignored the message angered Kirk
Spock suggests it	McCoy is silent	Spock suggests that McCoy be silent
Spock watched it	Kirk read the message	Spock watched Kirk read the message
Kirk asked it	The computer repeated the message	Kirk asked the computer to repeat the message
Spock claims it	Spock has written the message	Spock claims to have written the message
It shocked Kirk	Spock ignored the command	Spock's ignoring the command shocked Kirk
Spock saw it	McCoy read the message	Spock saw McCoy reading the message
It angered Kirk	Kirk read the message	Reading the message angered Kirk
Kirk knew it	The computer ignored the message	Kirk knew whether the computer ignored the message

Table 1. Examples of applications of connective transformations.

Recall that in section 2 we defined *parsing* only for kernel sentences. Connective transformations allow us to extend this notion to a wider class of English sentences which includes embedded sentences (see examples in section 11). After START determines the connective transformation involved it reverses the connective transformation and splits the sentence into kernel sentences. Then it parses each kernel sentence separately and produces kernel structures. And finally, the indexing procedure utilizes the lexical material provided by kernel structures to build T-expressions and index them as T-entries.

In order to handle embedded sentences, START allows any T-expression to take another T-expression as its subject or object. Thus, sentence (8) leads to *right embedding*:

(10) ⟨**Spock want ⟨computer print message⟩**⟩

while the sentence (9) leads to *left embedding*:

(11) ⟨⟨**Spock ignore command**⟩ **anger Kirk**⟩.

Connective transformations may be recursively applied without any restrictions on the depth of embedding. This means that START can analyze and generate sentences with arbitrarily complex embedded structures.

6. Complex Noun Phrases and Relative Clauses

We have seen how START analyzes a sentence and produces a T-expression whose corresponding T-entry "summarizes" the syntactic structure of the sentence. Let us examine now how complex noun phrases result in the construction of several additional T-expressions. Consider sentence (6):

(6) Jane's good friend from Boston recognized Paul.

The head of the noun phrase, *friend*, is premodified by *Jane's* and *good* and postmodified by the prepositional phrase *from Boston*. As a result, along with the main T-expression, **⟨friend-1 recognize Paul⟩**, the system will construct three additional T-expressions: **⟨friend-1 is good⟩**, **⟨friend-1 related-to Jane⟩**, and **⟨friend-1 from Boston⟩**. In fact, every adjective or possessive noun in the sentence, as well as every prepositional phrase or relative clause, will cause new T-expressions to be built and stored in the knowledge base.

START can handle different types of relative clauses:

(12) The girl *who wants to become an astronaut* is young.
(13) The planet *which Voyager photographed yesterday* was shrouded in clouds.
(14) The man *we admire* walked on the Moon.
(15) The planet *the spacecraft flew behind* has a strong magnetic field.
(16) The satellite *to which the antennas were pointing* had an impact crater.
(17) The spacecraft *orbiting the Earth* photographed its surface.
(18) The satellite *launched by NASA* handles telecommunications.
(19) The space shuttle *whose protective tiles were damaged* underwent repairs.

Let us analyze sentence (13), which involves a full relative clause with an object relative pronoun. First, the system has to find the relative clause boundaries and identify the location of the *gap* (denoted by e) which is coreferent to the head noun phrase:

> The planet$_i$ [which Voyager photographed e_i yesterday] was shrouded in clouds.

Then the relative clause is "removed" from the sentence, the gap is filled with its antecedent and the modified clause is processed independently. As a result, the sentence (13) will be split into the following two sentences:

(20) Voyager photographed the planet$_i$ yesterday

(21) The planet$_i$ was shrouded in clouds.

Relative clauses do not need to be simple kernel sentences (see example (12), for instance). In fact, any two sentences that may be analyzed by START, with arbitrarily complex embedded structures, can be combined into

main and relative clauses of a larger sentence as long as they have a common noun phrase (see the sample passage in section). Moreover, several relative clauses may be recursively embedded inside one another.

7. Lexical Ambiguity

Every word in the sample sentences discussed so far was assumed to belong to a unique part of speech. Thus, *Jill*, *friend*, and *man* are nouns, *read*, *write*, and *tell* are verbs, and *old* and *good* are adjectives. This assumption however is not always correct. Most words in English can receive several alternative category assignments (that is, can serve as different parts of speech); the particular choice depends on the context. For instance, in the following sentence from a detective story

(22) The gangsters *can supply uniform* alibis

the word *can* is used as a modal auxiliary, but it could also serve as a noun; the word *supply* is a verb, but it could also be a noun or a modifier in a noun-noun modification sequence; the word *uniform* is an adjective that could be used as a noun in a different context. Sentence (22) will be analyzed correctly only if the system selects the right category assignments for each word; any other assignment will result in an error.

Lexical entries in START (see section 9) are allowed to specify more than one category assignment. The system is equipped with a mechanism for category disambiguation which uses error feedback from the parser (including context information and type of error) to efficiently resolve ambiguities. As a result, along with sentence (22), START is able to process successfully another sentence from the same detective story:

(23) But the policeman found the *uniform* in the *supply can*.

Notice that each of the three ambiguous words in sentence (23) is a different part of speech from what it was in (22).

8. Forward and Backward S-rules

In sections 3 and 5 we showed how START builds T-expressions using the pattern ⟨**subject relation object**⟩ at every level of embedding. As a consequence, T-expressions closely follow the syntax of analyzed sentences. This property incidentally is one reason why the language generator is frequently able to reconstruct the original English sentence almost verbatim. Unfortu-

nately, this property also implies that sentences which have different surface syntax but are close in meaning will not be considered similar by the system.

An example will clarify this point. Given as input the sentence (24) START will create an embedded *T*-expression (25):

(24) Jane presented Paul with a gift

(25) **((Jane present Paul) with gift)**

whereas a near paraphrase, sentence (26), will generate *T*-expression (27):

(26) Jane presented a gift to Paul

(27) **((Jane present gift) to Paul).**

Speakers of English know that sentences (24) and (26) both describe a transfer of possession: *the gift* is the transferred object and *Paul* is its recipient in both sentences, despite different syntactic realizations of these noun phrases. It seem natural that this kind of knowledge be available to a natural language system. However, the START system, as described so far, does not consider expressions (25) and (27) similar. As a result, given only sentence (24) as input, the system will not be able to answer the following question:

(28) To whom did Jane present a gift?

This is a serious problem since interactions between the syntactic and semantic properties of verbs such as these pervade the English language and cannot be ignored when constructing a natural language system.

START's solution to this problem requires us to introduce the concept of an *S-rule*. *S*-rules (where *S* stands for both *S*yntax and *S*emantics) are implemented as a rule-based system where the antecedents and consequents are schemata of *T*-expressions (blueprints in which the elements of *T*-expressions may be replaced by variables) of the knowledge base. *S*-rules operate in two modes: *forward* and *backward*.

When triggered by certain conditions, *S*-rules in the forward mode allow the system to intercept *T*-expressions produced by the understanding module, transform or augment them in a way specified by the rule, and then incorporate the result into the knowledge base. For instance, the following *S*-rule can be used to solve the problem posed by a verb such as *present*:

If ((subject **present** object1) **with** object2)
Then ((subject **present** object2) **to** object1).

Notice that *subject*, *object1*, and *object2* are matching variables in this *S*-rule while *present*, *with*, and *to* are constants. As soon as the *S*-rule encounters expression (25) produced by START,

(25) ⟨(**Jane present Paul**) **with gift**⟩

it creates a new T-expression

(27) ⟨(**Jane present gift**) **to Paul**⟩

and then adds the corresponding T-entry to the knowledge base.

S-rules have several functions. They may represent simple lexical information about the possible ways a verb can realize its arguments, as in (24) and (26). S-rules can also express more complex knowledge about the real world. Notice that all additional facts produced by the forward S-rules are instantly entered in the knowledge base. This forward mode is especially useful when the information processed by START is put into action by another computer system because in this situation START ought to provide the interfacing system with as much data as possible.

In contrast, the backward mode is employed when the user queries the knowledge base. Often, for reasons of efficiency, it is advantageous not to incorporate all inferred knowledge into the knowledge base immediately. The backward S-rules trigger only when a request comes in which cannot be answered directly. Then the rules initiate a search in the knowledge base to determine if the answer can be deduced from the available information.

In a more complex situation, S-rules are allowed to trigger each other and to ask each other for help. At any given moment hundreds of rules may be hidden in the computer's memory examining the output flow generated by START and waiting for their turn to participate in the deduction process. S-rules fundamentally expand the power of our language understanding system; they open a window into the intricate world of syntax-semantic interactions.

9. The Lexical Component

In order to understand an English sentence, the system needs to have morphological, syntactic, and semantic information about the words in the sentence. All the words that the system is aware of, along with information about their part of speech, inflection, gender, number, *etc.* are stored in the *Lexicon*. Virtually every branch of our system resorts to the Lexicon to accomplish its task. In the understanding mode, the Lexicon is used to recognize embedded clauses, to construct kernel structures, to build T-expressions. In the generating mode, the Lexicon is consulted when a noun or verb phrase is built, when a connective transformation is applied, when a question is answered.

Let us examine how lexical information about verb classes may be utilized by the S-rules. Suppose we typed the following sentence into the computer:

(29) Paul surprised the audience with his performance.

An English speaker knows that sentence (29) can be paraphrased as:

(30) Paul's performance surprised the audience.

Surprise is one of many verbs which exhibit these two different syntactic realizations of its arguments (see Van Oosten [1980] and Levin [1987]). This property also holds for a large class of verbs, among them:

(31) amuse, anger, disappoint, embarrass, frighten, please, worry ...

These verbs share a certain semantic property as well: they all denote *emotional reactions*. For this reason we identify a class of *emotional-reaction* verbs and say that the syntactic property of the verb *surprise* responsible for the alternations shown in (29) and (30) holds for all verbs that comprise the *emotional-reaction* class.

S-rules allow START to take advantage of these regularities in the Lexicon. Knowing that sentence (29) results in *T*-expression

(32) ⟨⟨**Paul surprise audience**⟩ **with performance**⟩

one can write a simple *S*-rule which will trigger not only on the verb *surprise* but on any verb from the *emotional-reaction* class:

If	⟨⟨subject verb object1⟩ **with** object2⟩
Then	⟨object2 verb object1⟩
Provided	verb ∈ *emotional-reaction*

After typing sentence (29), we may now ask the system:

(33) Did Paul's performance surprise the audience?

The *S*-rule described above (used in the backward mode) will trigger, since the *T*-expression

(34) ⟨**performance surprise audience**⟩

produced by the question matches the THEN-part of the rule, and furthermore, the verb *surprise* belongs to the *emotional-reaction* class. The correct answer to the question is deduced when the IF-part of the rule is matched to *T*-expression (32) found in the knowledge base.

This example shows how the transparent syntax of the *S*-rules coupled with the information about verb class membership provided by the Lexicon facilitates fluent and flexible dialog between the user and the language understanding system. (See also Katz and Levin [1988]).

Our current Lexicon contains several thousand entries. However, the process of *lexical acquisition* (adding new words to the Lexicon, with all relevant

information about them) is very simple. In fact, introducing a new lexical item in START amounts to little more than appending it to a list of similar words, adding a few idiosyncratic features when necessary. Acquisition of S-rules is equally simple. Adding a new S-rule to the system requires typing in a set of English sentences (such as sentences (24) and (26)) which capture a specific instance of the rule. START will analyze the sentences, query the user for additional information regarding elements of corresponding T-expressions (ascertaining whether they are matching variables, constants, or predicates), and then build and generalize the S-rule automatically. All this makes the system *portable*, *i.e.*, easily adaptable from one domain to another.

10. Answering Questions

Recall our two criteria A and B of section 1 which define whether the computer understood English text. We have already seen how START creates knowledge bases employed by other AI programs (Test B). In this section we will concentrate on the question-answering machinery in START (Test A). Suppose the system has analyzed and indexed a text containing the sentence

(35) Spock wanted the computer to print the message.

The knowledge base now contains the following T-entry:

(36) $\langle$**Spock want** $\langle$**computer print message**$\rangle\rangle_H$.

Suppose now that a user asks:

(37) What did Spock want the computer to print?

First the system needs to *reverse* the effect of the *Question* transformation applied in (37). In order to accomplish this, the system must find the place in the sentence that the *wh*-word *what* came from. This situation is very similar to the treatment of *relative clauses* discussed in section 6; the system again needs to find the *gap* e that is coreferent with *what*:[5]

(38) Spock wanted [the computer to print e]

Once the location of the gap has been found, the language understanding system leads the sentence (38) through the same flow of control as any other declarative sentence and produces the following T-expression:

(39) $\langle$**Spock want** $\langle$**computer print** $e\rangle\rangle$.

Treating e as a matching variable the system then feeds T-expression (39) through a matcher in order to determine whether there is anything in the knowledge base that matches (39). The matcher finds the T-entry

[5]The same computational machinery is used to handle these two phenomena.

(36) (**Spock want (computer print message)**)$_H$,

retrieves it from the knowledge base, and hands it over to the language generation system which produces the English response to question (37):

Spock wanted the computer to print the message.

Other types of English questions, including *Yes/No*-questions, *when*-questions, *where*-questions, *why*-questions, *etc.* are treated in a similar fashion.

In this example we implicitly assumed that the tense and the aspect of question (37) were identical to the tense and aspect of sentence (35) in the text. We also assumed that sentence (35) was used in the text only once and that it was not embedded in another sentence. All these assumptions need not necessarily hold, however. For instance, one might ask:

(40) Does Spock want the computer to print the message?

or

(41) Did the computer print the message?

A person answering these questions in the context of (35) would probably say "I don't know" since sentence (35) just states that Spock wanted a certain action to happen at one time in the past. Sentence (35) does not imply that Spock wants this action to happen in the present nor does it imply that this action actually happened.

To illustrate a different case, suppose that it is known from the text that

(42) The telescope is orbiting the Earth.

Now someone may ask the following questions:

(43) Has the telescope been orbiting the Earth?

(44) Can the telescope orbit the Earth?

In spite of the fact that in the original sentence (42) the auxiliaries and the form of the main verb are different from those in questions (43) and (44), a person would most likely answer *Yes* in both cases. Somehow people know when they can or cannot answer such questions.

What about computers, then? Although clearly world knowledge plays an important role here, the text itself may often provide sufficient data to determine whether the information in the system's knowledge base implies a definitive (*yes* or *no*) answer to the question. Matching the embedded T-expressions described earlier is only a "rough" first step in answering a question. The next step requires a more subtle analysis of the histories attached to the T-entries returned by the matcher. Additional sentence features stored in the histories, such as tense, auxiliaries, embedding, voice, *etc.* allow the

system to determine whether the T-entries from the knowledge base and from the question refer to the same time interval and whether the meaning of the modal auxiliaries used in the text imply the meaning of the question asked. For a *Yes/No*-question, for instance, there are three types of responses: *Yes, No,* and *Unknown.* The *Unknown* category is further broken down to reflect the reason for this response—certain types of embedding, wrong time interval, or disagreement in corresponding modal auxiliaries. All this information is employed to answer the user's question. Our experiments show that the system's final responses, made on the basis of this analysis, echo people's judgments in answering such questions.

11. A Dialogue with START

The following is a slightly shortened and modified passage from the *Testing Reading Skills* chapter of the book *How to Prepare for the Graduate Record Examination* (Brownstein and Weiner [1985]), as entered into the computer:

> *The establishment of the Third Reich influenced events in American history by starting a chain of events which culminated in war. The complete destruction of democracy, the persecution of Jews, the cruelty of the Nazis, and the plans of Germany for the conquest of the world caused great indignation in the United States. While speaking out against Hitler's atrocities, the American people generally favored isolationist policies and neutrality. The Neutrality Act of 1935 prohibited trade with any belligerent nation. In 1937 the President was empowered to declare an arms embargo in wars between nations at his discretion.*
>
> *American opinion changed after President Roosevelt's speech at Chicago in which he severely criticized Hitler's policies. Germany's seizure of Austria and the Munich Pact also aroused the American people. The conquest of Czechoslovakia in 1939 was another rude awakening to the menace of the Third Reich. The United States attempted to maintain neutrality in spite of sympathy for the European democracies. The Neutrality Act of 1939 repealed the arms embargo and permitted exports of arms to belligerent nations. In 1940 a strong national defense program was begun and a draft act was passed to strengthen the military services. In 1941 Japan launched the unprovoked attack on the United States at Pearl Harbor. Immediately thereafter, Germany declared war on the United States.*

The understanding module of START analyzes this text and indexes the knowledge contained within it into a knowledge base. The user is then able to interact with the system in English. Shown below is an actual dialogue with START. The user asks questions and the computer answers them. Note that the English sentences generated by START may differ from the original text. Any stylistic deficiencies are the responsibility of START and not of this author.

The astute reader will observe that in addition to syntactic analysis, indexing, matching, retrieving the information, and generating English sentences, the START system was quietly performing additional tasks, such as: resolving lexical ambiguities (*e.g.*, the word *attack* was used as a verb and as a noun); working with synonyms (*begin-start, permit-allow, etc.*); employing *S*-rules (*e.g.*, in answering the question about Japan attacking the USA); and providing further information (in parentheses) when the correct answer could not be found due to embedding, or disagreements in time intervals or modality.

Querying the Knowledge Base

What did Germany's plans cause?

> **Germany's plans for the world's conquest caused great indignation in the United States.**

Did the American people favor neutrality?

> **Yes—The American people generally favored neutrality while speaking out against Hitler's atrocities.**

What did the Neutrality Acts do?

> **1935's Neutrality Act prohibited the trade with any belligerent nation.**
> **1939's Neutrality Act repealed the arms embargo.**
> **1939's Neutrality Act permitted the arms' exports to the belligerent nations.**

Who declared the Arms embargo?

> **I don't know. However, I do have some information:**
>
> **The President was empowered to declare the arms embargo in the wars between the nations at his discretion in 1937.**
>
> **(The proposition 'The President declares the arms embargo' was embedded within this sentence.)**

When did the American opinion change?

> **The American opinion changed after President Roosevelt's speech at Chicago in which President Roosevelt severely criticized Hitler's policies.**

Who criticized Hitler's policies?

President Roosevelt severely criticized Hitler's policies in his speech.

Who maintained neutrality?

I don't know. However, I do have some information:

The United States attempted to maintain neutrality in spite of the sympathy for the European democracies.

(The proposition 'The United States maintains neutrality' was embedded within this sentence.)

Were exports of arms allowed?

Yes—1939's Neutrality Act permitted the arms' exports to the belligerent nations.

To which nations did the neutrality act permit exports?

1939's Neutrality Act permitted the arms' exports to the belligerent nations.

Why was the draft act passed?

The draft act was passed in order to strengthen the military services.

Will a national defense program be started?

I don't know. However, I do have some information:

The strong national defense program was begun in 1940.

(This sentence did not refer to the time interval about which you asked.)

Did Japan attack the USA?

Yes—Japan launched the unprovoked attack on the United States at Pearl Harbor in 1941.

Acknowledgments

From the beginning Patrick H. Winston has been actively involved in the process of designing and developing START. He has also been a very active user of the system.

A number of students have participated in the project, in particular, David A. Chanen, Robert Frank, Jill Gaulding, and Jeff Palmucci contributed significantly to various parts of the system.

I am grateful to Mikhail Katz and Beth Levin for their time and numerous helpful suggestions concerning this paper. In addition, useful comments were provided by Richard Doyle, Robert Frank, Jill Gaulding, Michael Kashket, David Kirsh, and Thomas Marill.

This paper describes research done at the Artificial Intelligence Labo-

ratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided in part by the Advanced Research Projects Agency under Office of Naval Research contract N0014-85-K-0124.

References

1. S.C. Brownstein and M. Weiner, *How to Prepare for the Graduate Record Examination*, Barron's Educational Series, Inc., Woodbury, N.Y., 1985.
2. N. Chomsky, *A Theory of Syntactic Structures*, Moulton & Co., 1957.
3. R.J. Doyle, "Hypothesizing and Refining Causal Models," M.I.T. Artificial Intelligence Laboratory Memo No. 811, December 1984.
4. R.J. Doyle, "Construction and Refinement of Device Models," Ph.D. Thesis, MIT, *in preparation*, 1988.
5. B. Katz, "A Three-step Procedure for Language Generation," M.I.T. Artificial Intelligence Laboratory Memo No. 599, December 1980.
6. B. Katz and R. Brooks, "Understanding Natural Language for Spacecraft Sequencing," *JBIS*, Vol. 40, No. 10, 1987.
7. B. Katz and B. Levin, "Exploiting Lexical Regularities in Designing Natural Language Systems," to appear, 1988.
8. B. Katz and P.H. Winston, "A Two-way Natural Language Interface," in *Integrated Interactive Computing Systems*, edited by P. Degano and E. Sandewall, North-Holland, Amsterdam, 1982.
9. B. Levin, "Approaches to Lexical Semantic Representation," in *Automating the Lexicon*, edited by D. Walker, A. Zampolli, and N. Calzolari, to appear, 1988.
10. J. Van Oosten, "Subjects, Topics and Agents: Evidence from Property-factoring," *Proceedings of the Berkeley Linguistics Society* 6, Berkeley, CA, 1980.
11. P.H. Winston, *Artificial Intelligence*, Addison-Wesley, Reading MA, 1984.
12. P.H. Winston, "Learning New Principles from Precedents and Exercises," *Artificial Intelligence*, vol. 19, no. 3, 1982.
13. P.H. Winston, T.O. Binford, B. Katz, and M.R. Lowry, "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," *National Conference on Artificial Intelligence*, Washington, D.C., 1983.