6 THE MEANING OF MEANING

It boggles the mind to think that one day computers will be endowed with the language understanding abilities so far exclusively possessed by human information processors. Hastening that day will take hard work, to be sure, because dealing with unconstrained natural language surely requires strong, as yet undiscovered techniques for representation, constraint exploitation, and other things yet unknown. Still, something is already known, and this chapter's purpose is to show how certain systems of programs exhibit a kind of understanding as they participate in conversations about domains like that of the limited world of blocks, pyramids, and boxes.

THE BLOCKS WORLD

In studying how to make computers understand human language, each question seems invariably to generate another:

- What can it mean to say that a computer has understood something? Is it enough if the computer offers back an intelligent-sounding response?
- How much understanding is required to translate from one human language into another? How does this depend on the subject matter?
- How can a domain of discourse be delineated which is rich enough to be a vehicle for studying the central issues, yet simple enough to facilitate the progress?

Limited Domains of Discourse Are the E. Coli of Language Research

Perhaps the question of domain delineation is the most critical one because understanding language phenomena is like understanding molecular biology.
Neither problem succumbs to frontal assault, and one must work toward limited objectives with the hope that tactical victories will lead to strategic insights. For molecular biology, this means thorough study of very simple organisms, the intestinal organism *E. Coli* in particular. For artificial intelligence, the simple organisms are the strongly limited domains of discourse, the blocks world being an example.

**Robbie Converses about the Blocks World**

We are about to look at some fragments taken from a conversation between a human and a simulated one-armed robot created by Winograd in his landmark 1971 thesis. Winograd called his robot SHRDLU, but here he will be called Robbie to keep things consistent. Think of SHRDLU as an alias.

Robbie’s world is a simulated world of blocks, pyramids, and boxes. It is displayed as in figure 6-1 for the convenience of the human participant. Robbie’s responses are actions and upper case English.

![Figure 6-1](image)

**Figure 6-1** For language exposition, the blocks world consists of colored blocks together with a one-armed robot equipped with a magnetic hand. (Figures 6-1 through 6-5 are from *CAES Study Guide on Artificial Intelligence* by Patrick H. Winston, copyright by Center for Advanced Engineering Study, Massachusetts Institute of Technology, Cambridge, Massachusetts. Used with permission of the Center for Advanced Engineering Study.)

- **Pick up a big red block.**
  
  OK.

  To handle the command, and then plan a motion sequence to produce the result.

- **Find a block which is taller than the box.**
  
  OK.

  Robbie must deal with problem of...

- **What does the box contain?**

  THE BLUE PYRAMID

  Robbie must decide which...

- **Paint the green block blue.**

  OK.

  To handle the command, and then plan a motion sequence to produce the result.
Pick up a big red block.

OK.

To handle the command, Robbie must decide which block is the big red one and then plan a motion sequence suitable for getting to it. Figure 6-2 shows the result.

Find a block which is taller than the one you are holding and put it into the box.

OK.

Robbie must deal with pronoun references.

What does the box contain?

THE BLUE PYRAMID AND THE BLUE BOX.

Robbie must decide which objects are in the box and determine how they can be described to the human.
Figure 6-3 Here the robot had to put aside the green pyramid to get at the red cube required for the stack.

- What is the pyramid supported by?
  THE BOX.
  There is more than one pyramid, but Robbie knows that the human must be talking about the same one that was just reported to be in the box.
- Can a pyramid be supported by a block?
  YES.
  There is an example, so it must be possible.
- Will you please stack up both of the red blocks and either a green cube or a pyramid.
  OK.
  This requires some problem solving.

Figure 6-4 Modern language constructions as long as the

- Which cube is sitting on top?
  THE LARGE GREEN.
  Robbie tries to give
- Is there a large block?
  YES, THREE OF CUBE, AND THE
- Put a small one onto
  OK.
- Put the littlest pyramid
  OK. The result is sh
modern language understanding programs can cope with complicated constructions as long as the domain of discourse is limited. the answer to the question is yes.

- which cube is sitting on the table?
  the large green one which supports the red pyramid.
  robbie tries to give enough information for a unique determination.
- is there a large block behind a pyramid?
  yes, three of them: a large red one, a large green cube, and the blue one.
- put a small one onto the green cube which supports a pyramid.
  ok.
- put the littlest pyramid on top of it.
  ok. the result is shown in figure 6-4.
Does the shortest thing the tallest pyramid's support supports support anything green?

YES, THE GREEN PYRAMID.

When did you pick up the green pyramid?

WHILE I WAS STACKING UP THE RED CUBE, A LARGE RED BLOCK, AND A LARGE GREEN CUBE.

Robbie must remember what he has done and in what sequence.

Why?

TO GET RID OF IT.

Reasons must be remembered in some sort of goal tree. Here Robbie moves from one goal up to the goal's parent. How questions require going from parent to offspring.

Why?

TO CLEAN OFF THE RED CUBE.

Continuing with the same question will eventually result in the answer, “Because you told me to.”

Put the blue pyramid on the block in the box.

OK.

The sentence is ambiguous unless one looks at the scene being discussed. See figure 6-5.

Thank you.

YOU'RE WELCOME!

Most people are amazed by this dialogue. There is no doubt that the machine seems to be quite competent with respect to what goes on. Robbie’s world view is myopic since Robbie is no good at any question outside the world of the simulation. Nevertheless, Robbie’s block world is particularly convenient for studying many issues central to natural language processing, and we shall use it to illuminate some of the resulting ideas. We will only scratch the surface, to be sure — our interest will focus on the process by which object names are derived from their English descriptions.

Robbie Has Very Little to Talk About

The main reason the blocks world is so useful as a framework for beginning a study of natural language is that the things one can talk about are naturally limited. Let us be precise about this. One can talk about the following:

- **Objects**, such as specific blocks and pyramids;
- **Relations between objects**, such as IN-FRONT-OF and SUPPORTED-BY;
- **Actions**, such as PICK-UP, STACK-UP, etc. Now, what can be said that the other is moving a blue cube.
Figure 6-5 Sometimes knowledge of particular situations is required to disambiguate sentences. Here one choice is moving a blue pyramid which is on a block into the box, and the other is moving a blue pyramid onto a block which is in the box. One is eliminated by the known facts.

- Relations between objects and intrinsic properties, such as color and size;
- Actions, such as PICK-UP, PUT-ON, and STACK.

Now, what can be said in connection with these categories? We will look at questions and commands and ignore the problem of assimilating facts. Relations
and properties will appear in sentences of two types: questions about whether a
relation or a property exists and commands to make a relation exist:

*Is the red cube in the box?*
*Is the red cube big?*
*Put the red cube on the blue block."

What is needed to handle these and similar sentences? What kind of knowledge? How much? What in particular?

**Understanding Requires Multilevel Expertise**

Even the simple language of blocks-world questions and commands is hard to deal
with. Words must be assembled into groups, groups into sentences, and sentences
into coherent histories. There is a need, therefore, for several specialists:

- The *syntactic specialist* bounces about in a sentence with only the modest goal
  of segmenting it into meaningful word groups. The syntactic specialist divides
each simple sentence into a verb group and one or more noun groups. The verb
group consists of a verb together with accompanying auxiliaries and adverbs.
Noun groups similarly consist of a noun and various modifiers. To do its job
the syntactic specialist must know about how word ordering depends on
syntactic word features.

- The *sentence specialist* determines how the objects described by the noun groups
  relate to the action described by the verb group. The sentence specialist needs
to know how to use both syntactic and semantic information in this analysis.

- The *scenario specialist* understands how individual sentences relate to one
  another and to the general story that they collectively tell. In the simple
blocks-world situation, the scenario specialist must handle pronoun references
and the time order of all actions performed. In less simple situations it must
understand how groups of sentences convey implications, evoke expectations,
recall memories, establish contexts, "but" - ors-

Thankfully the natural limits on blocks-world conversation permit concentration
on the syntactic level, with only brief forays into the sentence-level and scenario-
level problems.

**Case Analysis is Straightforward in the Blocks World**

The purpose of case analysis is to sort out how the noun groups in a sentence fill
various roles. For commands, who is to do the action? For whom? To what? In the
blocks world, most of the case slots are easily filled by using tightly bound defaults
or by applying very simple syntactic criteria. Robbie, the robot, is the only entity
that can do anything, hence Robbie is always the agent. Similarly there is no question
about for whom the action is done because the human giving commands is the only
possibility. And finally, the object involved in the action must be some sort of block
and it is generally the only one in the preposition. The instrument, overall:

**agent:** Robbie
**beneficiary:** human
**instrument:** Robbie
**location:** Robbie
**object:** noun group

Indeed there is so much redundancy in noun groups into cases it

The *Robbie Uses Object Descriptions* dialog.

For the most part, actions like put, above, and red. The
dialogue reveals that quite often objects by description, specifications? Presumably
specifying utterance and "pyramid," is somehow interpreted by our vision apparatus. It
is vague. The necessary equivalence is not clear. There is no external physical word
a particular object in this

The first is the analysis concealing
the blocks world in hand
are named directly, not by
avoid associating unique
of the primitive actions

All of this means that
The first is the analysis of
second is the use of these
In the following example we call to the PICK-UP procedure

Pick-up a big red block
Pick up BLOCK3.
(PICK-UP BLOCK3)
questions about whether a
ation exist:

What kind of knowledge?

mmands is hard to deal
sentences, and sentences
ral specialists:

ith only the modest goal
ntactic specialist divides
our groups. The verb
uxiliaries and adverbs.
oderisers. To do its job
ordering depends on

ributed by the noun groups
tence specialist needs
formation in this analysis.

sentences relate to one
ively tell. In the simple
andle pronoun references
rple situations it must
onitions, evoke expectations,
and develop plots.

tion permit concentration
entence-level and scenario-

h groups in a sentence fill
or whom? To what? In the
sing tightly bound defaults
robot, is the only entity
larily there is no question
iving commands is the only
ust be some sort of block

and it is generally the only one inside a noun group lacking an introductory
preposition. The instrument and the location are obvious, giving the following
overall:

agent: Robbie
beneficiary: human master
instrument: Robbie's hand
location: Robbie's simulated world
object: noun group without a preposition
containing a block, pyramid, etc.

Indeed there is so much constraint in the blocks world, the problem of sorting the
noun groups into cases is not a significant task for most sentences.

Robbie Uses Object Descriptions and Action Names

For the most part, actions, relations, and properties are denoted directly by words
like put, above, and red. This is not the case with objects. Instead, inspection of the
dialogue reveals that quite a number of the words are devoted to specifying the
objects by description, not name. What can it mean to understand these
specifications? Presumably we humans form some sort of association between the
specifying utterance and the visual perception. "A big red block which supports a
pyramid," is somehow tied up with an interior model of physical reality built up
by our vision apparatus. How this is done is unknown so the statement is admittedly
vague. The necessary equivalent for a computer system is not vague, however. There
is no external physical world, only the simulated world in memory. To ask about
a particular object in this world, one must get at its name, which would be something
like BLOCK3 or PYRAMID2 or 31415. The trick then is to derive these names
from a description couched in an English utterance. No such trick is required by
the blocks world in handling the relations and actions relevant in it because these
are named directly, not described. We say pick up and behind. There is no need to
avoid associating unique names with these concepts in our speech because there are
few of them. This is true in general. We could not hope to give a unique name to
every physical object we deal with in our experience, but we do have names for most
of the primitive actions that we can perform and the relations and properties we
can perceive.

All of this means that blocks-world understanding involves two semantic steps.
The first is the analysis of object description and substitution of object names. The
second is the use of these names and the other words of the sentence to do something.
In the following example, "Pick up a big red block," ultimately becomes a simple
call to the PICK-UP program:

Pick-up a big red block.

Pick up BLOCK3.

(PICK-UP BLOCK3)
We shall see that blocks-world translations of this sort can be effected by cooperating semantic and syntactic programs. The semantic specialist program uses adjectives, nouns, and verbs to supply program fragments which are pasted together under the direction of the syntactic specialist.

**Understanding Connects Utterances to the World**

Language consists of meaningful groups of words arranged in sentences. Since we can talk about things, there are noun groups. Since we can talk about actions, there are verb groups. Somehow the noun groups become associated with the objects of the physical world. “Note the big block,” exhorts the listener to look at a particular object. “Note 27182,” would be simpler and less susceptible to ambiguity, but humans denote objects mostly by description, not by name.

Now what can it mean to understand these descriptions? Presumably we humans use our vision hardware to create descriptions of our three-dimensional physical environment. Perhaps these descriptions are symbolic structures with hierarchical levels. Understanding a noun group may correspond to establishing symbolic links between noun groups discovered by language procedures and the symbolic structures developed by vision processing. This is vague. With Robbie we can be much more precise:

- A noun group is understood by identifying the name of the object described.

What is in a name? Since Robbie uses only an internal simulation, not a real physical world, getting a name from a noun group is all important. If we ask Robbie about the color of the big block, he must work through 27182, the object’s name. If we command Robbie to pick it up, again the name is the key to the location information which Robbie needs to position its hand. Indeed almost all of what Robbie knows about objects is accessed by object name just as most of what we know of addresses and telephone numbers is accessed by the names in the telephone book. There is no way to get the telephone number of the beautiful person with the blonde hair.

**Translating Noun Groups into Programs Makes Sense in the Blocks World**

Let us state a series of claims:

- Understanding a blocks-world utterance involves a process by which one description is translated into another. The first description is in natural language. The second description is one that is useful in generating a relevant response.

- Very often the target descriptions for blocks-world sentences are best expressed in the form of programs. These programs, when run, make relevant responses.

- If the initial utterance is a command, the target can be a program with arguments which accomplishes the required action. If a question is asked, the program can be one that retrieves information, makes deductions, and gives answers.

How can this idea be put into practice? It depends heavily on the natural language on which it is based. Each sentence is like a program to be run and understood. There are various ways of achieving the goal, some arguing toward a common sense or cause-effect relation, others toward functions made when things happen, and some just about the whole system.

What about translating the natural language predicate calculus on which we are working? It does not look much like a program. The natural language representation takes, at best, the form of something much like a program, and perhaps a little like something. If the translation is to be useful, the program must persuade us that it is a program that does the data really does answer the question.

**Noun Group Programs**

Robbie’s ability to do this is a result of the fact that NP specialists that look at natural language and run those programs to help.

- Programs themselves.

Thankfully there is little need for formal programming. The context is enough.

**THE STRUCTURE OF NOUN GROUPS**

The simplest noun groups have the following form:

Determiners are words that tell you what, where, which and how many.

There are many different kinds of determiner, if there is one, after any noun. After seeing the noun, there may be either an adjective or article. Thus the syntactic analyst is faced with a very likely. Sometimes the noun is also a classifier, but we will ignore that here.

- Present information, makes deductions, and gives answers.
How can this idea of translation into programs be defended? The explanation depends heavily on the absence of connected dialogue in blocks-world conversations. Each sentence is likely to be independently formulated and independently understood. There are no paragraphs developing an explanation of a situation or arguing toward a conclusion or preparing background context for understanding a cause-effect relationship. Consequently the analogy between sentences and functions made when we discussed the constraints that underlie case grammar are just about the whole story of blocks-world meaning.

What about translating utterances into some sort of interlingua in the form of predicate calculus or networks of concepts or some other representation that does not look much like a program? The answer is that no matter what form the representation takes, there is a program lying camouflaged somewhere. Certainly some representation of a function with arguments may be used which does not look much like a program, but if a response is to be made, then the responder must do something. If the translation goes into something that looks like data, then some program must peruse the data to respond relevantly. But this is a way of saying that the data gives implicit instructions about what to do, an idea that is just one step away from saying that the data is an embodiment of an implied program. Some analysis is required to convert the implicit form into explicit actions perhaps, but the data really does act like a disguised programming language.

**Noun Group Programs Find Names**

Robbie’s ability to deduce object names from descriptions rests on a collection of specialists that look at the noun groups, use them to write search programs, and run those programs to get the names. This involves a beautiful idea:

- Programs themselves can write and run their own programs.

Thankfully there is no need to put off further discussion until we do some programming. The concepts are handled here and details deferred.
Grammars Are Mechanisms for Describing Language

Of the many schemes for representing constraints about language, the context-free grammar, the transformational grammar, and the augmented-transition-network grammar are particularly prominent. All are theories of language since all contain descriptions of the constraints that determine if a word sequence is a valid sentence or just disordered garbage. Shortly we will concentrate on grammars represented as augmented transition networks because they suit our purpose best. First however, it is appropriate to look at the other grammars just to know that there are alternative approaches.

To understand both context-free and transformational grammars, think first of a black box which has no input, just an output. From the output come valid English sentences, first the one-word ones, and then progressively longer ones. Inside the box, a typical grammar consists of a set of rules, each of which accepts a word string, modifies it, and delivers the new result. These rules can have a distinctly production-like flavor as shown by these examples:

R1
If the symbol S is present, then replace S with the symbols SUBJECT VERB.

R2
If the symbol SUBJECT is present, then replace SUBJECT with the symbols THE PROGRAMMER.

R3
If the symbol VERB is present, then replace VERB with the symbol PROGRAMS.

Sentences are generated by applying the rules over and over again to an initial start symbol, S. Following the rules in the example is easy because there is never any point where more than one rule applies to the same symbol. Thus applying rule R1, R2, and R3, in that order, effects the following changes.

S
SUBJECT VERB
THE PROGRAMMER VERB
THE PROGRAMMER PROGRAMS

While bigger grammars could be written in the same way, linguists habitually use some sort of shorthand notation. This is certainly true when the rules are the so-called context-free rules, those which look for a single symbol and then just replace it. Using the standard shorthand for such rules, the previous grammar can be written as follows:

GRAMMAR 1: S → SUBJECT VERB
SUBJECT → THE PROGRAMMER
VERB → PROGRAMS

In the next, slightly more complex example, we see how the general notion of a single symbol. More specifically, the symbol S could be replaced by an equivalent context-free grammar.
In the next, slightly larger grammar, more than one rule frequently applies to the same symbol. More interesting sentences can follow.

**GRAMMAR 2:**

- **S → SUBJECT VERB**
- **S → SUBJECT VERB OBJECT**
- **SUBJECT → DETERMINER NOUN**
- **OBJECT → DETERMINER NOUN**
- **DETERMINER → A**
- **DETERMINER → THE**
- **NOUN → COMPUTER**
- **NOUN → PROGRAMMER**
- **VERB → PROGRAMS**
- **VERB → LOVES**

Such context-free grammars can become very large indeed, but eventually it becomes clear that to get around a bigger part of language, more general rules are needed. The popular extension is to enlarge the situation and action parts of the rules so that more than one symbol determines whether the rule applies and so that symbols can be rearranged and deleted as well as simply replaced. These more general rules are called context-sensitive rules. Studying grammars that use such rules, the so-called transformational grammars, takes one into territory belonging mostly to linguists. Rather than visit there, we will turn to representing facts about language in augmented-transition-network grammars, which are more convenient, it seems, for the development of language-understanding programs.

**Recursive Transition Nets Also Can Hold Syntactic Information**

A Transition Network is a structure which conveniently captures a variety of facts about syntax. The first part of figure 6-6 gives the basic transition network equivalent for simple noun groups. Each circle represents a state, and labeled arcs show how words of a particular type cause transitions from one state to another. Adjective transitions loop back to the same state since there can be any number of adjectives in a noun group. One can think of a valid noun group as a set of instructions for moving along a path from the starting state to a so-called accepting state represented by a double circle. The node S3 is a double circle because getting there means success. Analysis is complete. A string of words which drives through the network to an accepting state is interpreted to be a noun group. Whenever a sequence of words leads to a state from which all transitions are incompatible with the next word, then analysis is derailed. Such a sequence cannot constitute a simple noun group as defined here.

Prepositional groups are basically noun groups introduced by prepositions. The second part of figure 6-6 shows how to represent them by references to the already defined noun group structure. Note that arcs labeled with names of groups require successful traversal of the named network rather than the presence of a single word. By convention, names of groups appear on arcs in upper case.
Figure 6-6 Transition networks describe grammatical constraints. In the first diagram, a simple transition network defines a noun group as a determiner, any number of adjectives, and a noun. In the second diagram, a prepositional group is shown to consist of a preposition followed by a noun group. In the third diagram, recursion is introduced by an arc which specifies that optional preposition groups may follow a noun group's noun.

Very often a noun group description is strengthened by one or more prepositional groups following the noun and further specifying it. The third part of figure 6-6 pictures this generalization. An example requiring it is the noun group, "a red pyramid on the big block," which contains the prepositional group, "on the big block."

Since noun groups can contain prepositional groups and prepositional groups can contain noun groups, the combination is recursive. Imbedding can go on to an arbitrary depth:

_A red pyramid_
_on the big block_
_nearly the empty box_
_by the furry purple cube...._

Traversing a Transition Network

The dictionary contains word groups. It also contains prepositional groups.

For the most part, the words they contain are in the dictionary.

Word type

- The _DETERMINER_ may correspond to the noun group.
- The _PREP_ may have symbols like FOUR in the network.

Transition networks also describe the ways where information flows.
Traversing a Transition Net Accumulates Facts

The dictionary of words contains the necessary word-type information needed to use word groups to steer paths through transition networks. Naturally enough, the dictionary contains more information about words than just the basic type. For understanding how Robbie does things, two other kinds of information are important:

- The word dictionary contains other syntactic features in addition to type.
- It also contains program fragments.

Let us deal with the features first. The features are properties of a word that determine the word's grammatical role without evoking much of a semantic image. The common possibilities for the type feature are DETERMINER, NOUN, VERB, ADJECTIVE, NUMBER, PARTICLE, and PREPOSITION. One of these is in force for each word of a sentence. Others include selections from among SINGULAR and PLURAL for nouns, SINGULAR, PLURAL, DEFINITE, and INDEFINITE for determiners, and PRESENT, PAST, and TAKES-PARTICLE for verbs.

It makes sense to attach features to noun groups too. These are inherited from the words they contain, since only words, not groups of words, have entries in the dictionary.

For the most part the features for a noun group are named for the word types that appear in noun groups. Determiner and number are two important examples. We have seen the determiner word type. The number word type, not previously introduced, identifies a word or group of words which indicates how many, as in "the three tall pyramids." Thus the terms determiner and number are used in two ways. For noun groups, determiner and number are the names of features which have values; for words, type is the name of a feature which has some value, DETERMINER and NUMBER being particular possibilities.

- The DETERMINER is DEFINITE or INDEFINITE depending on whether the noun group refers to something definite.
- The NUMBER is usually SINGULAR or PLURAL. More general noun groups may have symbols like NO, ALL, EXACTLY THREE, or MORE THAN FOUR in the number slot.

Transition networks deduce features as a side effect of successful traversal. Here is where information for the noun group features comes from:

<table>
<thead>
<tr>
<th>Word type</th>
<th>Noun group feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETERMINER</td>
<td>DETERMINER (DEFINITE OR INDEFINITE)</td>
</tr>
<tr>
<td>NUMBER</td>
<td>NUMBER (SINGULAR, PLURAL, NO, ETC.)</td>
</tr>
<tr>
<td>NOUN</td>
<td></td>
</tr>
</tbody>
</table>

The first diagram, a number of adjectives, consist of a preposition are calls to the named specifies that optional by one or more it. The third part it is the noun group, special group, "on the prepositional groups ending can go on to an
Evidently the number feature for noun groups coalesces and summarizes information from several sources. If the noun group has a number word or group of words, then that usually dominates. Otherwise the determiner or the noun may at least specify singular or plural. The determiners a and this and that are singular while these and those are plural. The is neither. If the number feature of the determiner clashes with that of the noun, the combination is not grammatically correct. Only small children can get away with “this blocks” or “those pyramid.”

Augmented Transitions Prescribe Side Effects

Notes hanging on the arcs indicate certain actions to take. These notes and their consequences distinguish augmented transitions from ordinary ones. Figure 6-7 demonstrates how these notes augment the noun-group transition network. As shown, a note on the determiner arc checks the determiner word to see if it is either definite or indefinite. If so, the DEFINITE or INDEFINITE feature is transferred up to the noun group and written into a memory slot reserved for the noun-group’s determiner feature. At the same time another note on the determiner arc checks to see if the determiner is either SINGULAR or PLURAL feature.

Of course the noun group’s number slot may be there that may be there involves a test as well as recording, recall, or ATNs.

Robbie’s Dictionary

A world model contains patterns:

- An object name
- An object name
- An object name

Here are some examples:

```
31415
TABLE
31415
27182
31415
27182
31415
21415
27182
```

Eventually it would be a way that template pattern looks like this:

```
?X HAS-COLOR
```

Thus patterns are listed that begin with ? signals when memory match the fact

```
31415 HAS-COLOR
```
to see if the determiner word is known to be either singular or plural, and if so, writes SINGULAR or PLURAL into a memory slot reserved for the noun group's number feature.

Of course the note on the noun arc can also affect the information in the noun group's number slot, but only if nothing is there already. More specific information that may be there should not be lost. Action is therefore conditional, and the note involves a test as well as an action. Recursive transition networks with notes enabling recording, recall, and testing of facts are called Augmented Transition Networks or ATNs.

Robbie's Dictionary Contains Program Fragments

A world model contains many facts about objects which conform to one of three patterns:

- An object name, a relation name, and another object name.
- An object name, the relation IS-A, and a class name.
- An object name, a property naming relation, and a property.

Here are some examples:

<table>
<thead>
<tr>
<th>Object</th>
<th>Relation</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>31415</td>
<td>SUPPORTS</td>
<td>27182</td>
</tr>
<tr>
<td>TABLE</td>
<td>SUPPORTS</td>
<td>31415</td>
</tr>
<tr>
<td>31415</td>
<td>IS-A</td>
<td>BLOCK</td>
</tr>
<tr>
<td>27182</td>
<td>IS-A</td>
<td>PYRAMID</td>
</tr>
<tr>
<td>31415</td>
<td>HAS-COLOR</td>
<td>RED</td>
</tr>
<tr>
<td>27182</td>
<td>HAS-COLOR</td>
<td>BLUE</td>
</tr>
<tr>
<td>31415</td>
<td>HAS-LOCATION</td>
<td>(0 0 0)</td>
</tr>
<tr>
<td>27182</td>
<td>HAS-LOCATION</td>
<td>(0 0 10)</td>
</tr>
<tr>
<td>21415</td>
<td>HAS-SIZE</td>
<td>(10 10 10)</td>
</tr>
<tr>
<td>27182</td>
<td>HAS-SIZE</td>
<td>(10 10 15)</td>
</tr>
</tbody>
</table>

Eventually it will be clear how such facts can be recorded in memory in such a way that template-like patterns can be used to retrieve them. A typical retrieval pattern looks like this:

?X HAS-COLOR RED

Thus patterns are like facts in memory except for the possible presence of names that begin with ? symbols. These are special names that serve as anything-goes signals when memory is searched for instances of a pattern. The pattern sample will match the fact

31415 HAS-COLOR RED
just as if 31415 were in the first position of the pattern because names beginning with ? match anything.

Pattern-matching programs allow memory searches which incorporate several basic patterns into a general requirement. Here is an example which differs from actual program form only in unimportant syntactic details:

FIND ALL X SUCH THAT
?X HAS-COLOR RED
?X SUPPORTS ?Y
?Y IS-A PYRAMID

These four lines find all red objects that support pyramids. This is done in two steps. First the patterns are used to retrieve potential values of X and Y from the facts in memory. Then all pairs are tested to see which combinations simultaneously fit all slots.

Much of the program’s content comes straight from program fragments stored in memory with each adjective and noun:

<table>
<thead>
<tr>
<th>Word</th>
<th>Code property</th>
<th>property</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>? HAS-COLOR RED</td>
<td></td>
</tr>
<tr>
<td>SUPPORTS</td>
<td>? SUPPORTS ?</td>
<td></td>
</tr>
<tr>
<td>PYRAMID</td>
<td>? IS-A PYRAMID</td>
<td></td>
</tr>
</tbody>
</table>

The trick is therefore to have a program-writing program which builds search programs out of such fragments. The search programs then come up with the names of objects.

There are many variations which require attention to the number of satisfactory value combinations. The program for the noun-group, “a red object which supports three pyramids,” specifies that three pyramids must be found:

FIND > 0 X SUCH THAT
?X HAS-COLOR RED
THERE ARE > 2 Y SUCH THAT
?X SUPPORTS ?Y
?Y IS-A PYRAMID

Thankfully this introduces a general framework. The expressions > 0 and > 2 fill slots which determine if the search is successful. Filling these slots differently produces many different translations:

<table>
<thead>
<tr>
<th>Fragment</th>
<th>X slot</th>
<th>Y slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>... a red object which supports a pyramid.</td>
<td>&gt; 0</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>... a red object which supports the pyramid.</td>
<td>&gt; 0</td>
<td>= 1</td>
</tr>
<tr>
<td>... a red object which supports three pyramids.</td>
<td>&gt; 0</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>... a red object which supports exactly three pyramids.</td>
<td>&gt; 0</td>
<td>= 3</td>
</tr>
</tbody>
</table>

Thus the individual value combinations determine if fulfillment is complete. Combined together in memory they give a general framework of meaning with precise boundaries.

Evidently, then, the procedure is not difficult. It can be easy, given only a program which builds search programs out of unimportant details.

A Group’s Features Determine Its Interpretation

Syntactic analysis gives names to features which are exactly the same as examples, are the determinant of features. The only ways of completing them are:

| Determiner | indefinite | definite | indefinite | indefinite | no |

The patterns supplied by the program-writing program are usually supplied by other word combinations.

INTERPRETING QUESTIONS

Once a program understands the meaning of a noun-group, all relationship questions are answered in the same way. For example, are the determiners, and how many in front of the noun?

What bricks are there?

How many bricks are there?

What bricks support a pyramid?

How many bricks support a pyramid?

Answers to all four questions:

FIND ALL X SUCH THAT
?X IS-A BRICK
?X HAS-COLOR RED
?X SUPPORTS ?Y
?Y IS-A PYRAMID
... a red object which supports no pyramids.  \[> 0 = 0\]
... the red object which supports a pyramid.  \[= 1 > 0\]
... the red object which supports the pyramid.  \[= 1 = 1\]

Thus the individual pattern lines are retrieved from the word dictionary and the success slot specifications are built out of \(=, >, <, \text{ALL}, \text{and the integers}\). Combined together in a simple, program-like structure, they express great variety of meaning with precision, clarity, and economy.

Evidently, then, translation from blocks-world noun groups to search programs can be easy, given one basic structure and knowledge about how to fill the slots. A simple program can do the job. Let us see what such a program needs to know.

**A Group's Features Determine the Shape of the Corresponding Program**

Syntactic analysis gives determiner and number features for noun groups. These features are exactly the information needed to fill the success slots. Here, for example, are the determiner, number, and consequent success slot fillers for various ways of completing the noun group, "... a red object which supports ...."

<table>
<thead>
<tr>
<th>Determiner</th>
<th>Number</th>
<th>Success slot</th>
<th>Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>indefinite</td>
<td>singular</td>
<td>(&gt; 0)</td>
<td>... a pyramid</td>
</tr>
<tr>
<td>definite</td>
<td>singular</td>
<td>(= 1)</td>
<td>... the pyramid</td>
</tr>
<tr>
<td>indefinite</td>
<td>(&gt; n)</td>
<td>(&gt; n)</td>
<td>... (n) pyramids</td>
</tr>
<tr>
<td>indefinite</td>
<td>(&lt; n)</td>
<td>(&lt; n)</td>
<td>... at most (n) pyramids</td>
</tr>
<tr>
<td>no</td>
<td>plural</td>
<td>(= 0)</td>
<td>... no pyramids</td>
</tr>
</tbody>
</table>

The patterns supplied by adjectives and nouns are like bricks and the features supplied by other words are like mortar.

**INTERPRETING QUESTIONS AND COMMANDS**

Once a program understands noun-group translation, many property and relationship questions are easy to answer. They consist of question words like *what* and *how many* in front of a verb and an adjective or a verb and a noun group.

What bricks are blue?

How many bricks are blue?

What bricks support a pyramid?

How many bricks support a pyramid?

Answers to all four questions are found by these two programs:

FIND ALL X SUCH THAT

?X IS-A BRICK

?X HAS-COLOR BLUE
FIND ALL X SUCH THAT
?X IS-A BRICK
THERE ARE > 0 Y SUCH THAT
?Y IS-A PYRAMID
?X SUPPORTS ?Y

Some WHAT and HOW MANY Questions Are Easy

Of course these programs are exactly like ones that identify objects described by certain noun groups. Indeed, many questions telescope into noun group-programs. This is no surprise since before a noun group can be useful, there is the implied question of what the noun group represents. The difference is in purpose, not construction. For dealing with command sentences, such programs supply arguments to manipulation functions. With questions, the results are enumerated if what is the question word, and they are counted if how many is used. Direct questions simply bring the answering process up front.

Since question forms seem to translate into program structures in the same way ordinary noun groups did, there is no reason to go into more detail. Simply note that dealing with property and relationship questions also requires enumeration of the English variations that occur and design of the proper programs to go with each. As before, the number of possible combinations is smaller and easier to capture than might be supposed. The only difficulty is that internal object names like 31415 are not well received by human questioners. The enumeration required by what, which, and how many questions should be done by object description not by object name. Generating suitable descriptions is a chore.

Commands Require Mating Function Names with Object Names

Now Robbie's interpretation of commands can be understood in more detail. Consider

Pick up a red object which supports a pyramid.

Syntactic specialists divide the sentence into verb and noun groups. Since the world is so constrained, there is never ambiguity in a command about what fills the agent, instrument, and object case slots. Robbie is always the agent, its hand is the instrument, and the sentence supplies the object.

The verbs generally specify what is to be done directly. In contrast to the great number of individual objects possible, there are but few actions, and each can therefore have its own unique name. There is no need for intermediate computation in which descriptions become search programs and search programs which then find names. Program names come directly from the verbs that specify them. The pick up combination causes reference to a program named PICK-UP, taking one argument, the occupant of the object case slot.

Then comes "pyramid," into program producing like 31415.

Finally the command program producing the sake of the LI blocks-manipulation analysis. The command is complete, and a description suitable for action.

![Figure 6-8 In the block analysis, particle and noun groups are like arguments. Verbs produce program names only after internal computation.](image)

**SUMMARY**

- It can be argued that general intelligence seems necessary.
Then comes translation of the noun group, "a red object which supports a pyramid," into program form. Running the program generates a name, something like 31415.

Finally the computed object name is brought together with the retrieved action program producing a function and argument pair, surrounded by parentheses for the sake of the LISP language's syntax: (PICK-UP 31415). This is something that blocks-manipulation specialists can deal with directly. There is no further English analysis. The conversion from English command to an executable function call is complete, and a description suitable for communication has become a description suitable for action. Figure 6-8 summarizes.

![Figure 6-8](image)

In the blocks world verbs often correspond to function calls and noun groups are like arguments. Verbs produce function names directly, but noun groups produce object names only after intermediate construction and execution of search procedures.

**SUMMARY**

- It can be argued that making a computer understand language insures its general intelligence because the same issues and the same problems are involved.
- Understanding natural language is an incredibly complex phenomenon. It seems necessary to start with limited domains to gain a foothold.
For the blocks world, the problem of translating English descriptions of objects into internal names is a key activity.

Augmented Transition Networks capture facts about word ordering constraints and provide a convenient, perspicuous way of employing those facts in analysis. The term *augmented* is used because the user is free to write into and later test various memory slots as transitions are made.

An ATN can assemble together the material for search programs in the course of identifying a noun group. The actual assembly of the program is done by a specialist that uses both the raw material drawn by the ATN from the word dictionary and the structure of the noun group's determiner and number features.

Having found a way of associating descriptions with internal names, certain questions and commands are easy to handle. Very similar search programs turn up the names with a view toward reciting them, counting them, or passing them as arguments to action programs.

REFERENCES


Using Augmented Transition Networks for natural language seems to have started with a 1968 paper by Thorne, Bratley, and Dewar, of Edinburgh. Work by Bobrow and Fraser and by Woods developed and popularized the idea soon thereafter. Woods, especially, became a major contributor with his work at Harvard and at Bolt, Beranek, and Newman, Inc. The following papers were particularly important to development:

- William A. Woods, "Transition Network Grammars for Natural Language Analysis," *Communications of the Association for Computing Machinery*, vol. 13, no. 10, 1970. This paper not only lays out the basic ideas but also discusses some general issues involved in selecting a representation for syntactic knowledge.