MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 194 August 1979

CONCEPTUAL PHRASES AND DETERMINISTIC ENGLISH PARSING

by

David Dill

Abstract

The grammar of many of the lower-level constituents of grammatical structures in English has not been a area of exciting new linguistic discovery, in contrast with study of clause-level constituents. The syntax of these *conceptual phrases*, as they are termed here, seems to be somewhat *ad hoc*, which presents problems for their specifation for the purpose of computer understanding of natural language.

This report concludes that their irregular behavior stems from a closer relationship between the syntax and the semantics of these than other English constructs. Conceptual phrases all correspond to objects in a single, tightly constrained semantic class, and as a result, semantic knowledge about them can be used to 'optimize' the process of communicating them.

The unique nature of conceptual phrases is exploited to provide a combined syntactic and semantic description for them, consisting of syntactically augmented *frames*, that is much simpler than individual syntactic or semantic descriptions. An example representation for numbers is given, along with an analysis of some problems that occur when a practical implementation is attempted.-

This report describes research 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 Reasearch Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643

Table of Contents

Acknowledgements

I must first thank Mitch Marcus, for suggesting this area of research and preventing me from straying too far afield, for his many invaluable ideas during our discussions, and for contributing greatly to my knowledge of, and interest in, natural language.

Bob Berwick was extremely helpful, both by taking the time to explain to me the inner workings of Parsifal, and by making many helpful suggestions.

My wife, Gail Kaiser, contributed to making this document much more comprehensible than it would have been by pointing out the gibberish while I was writing it. She also provided much moral support.

Finally, I would like to thank the people I have been working with over the last four years, in particular, Al Vezza, Tim Anderson, Dave Lebling, and Brian Berkowitz for making my time at MIT as pleasant as it has been, in addition to contributing to my education.

1. Introduction

This report was motivated by a desire to facilitate the construction of a more complete grammar for Parsifal, a computer program that parses written English. The original program and grammar were implemented in an attempt to provide a better computational description of linguistic phenomena, relying heavily on the results of current research in transformational grammar.

Since research in generative grammar has concentrated primarily on phenomena involving the structure and manipulation of syntactic constructs at the level of detail represented by clauses and noun phrases, the main focus in the development of the current Parsifal grammar has concentrated on explaining these 'higher-level' phenomena.

Parsifal has been successful in achieving its goal of explaining linguistic observations about the nature of these higher-level constituents. It is noted in the original paper describing Parsifal [5] that its grammar for clause-level syntactic structures, though simple, is more complete than either W. Woods' LUNAR system [12], or Winograd's SHRDLU [11], since these both tend to concentrate on parsing 'phrase-level' constituents.

However, there are many other issues that must be dealt with in order to arrive at a English understanding system that is truly practical. Some of the more obvious issues are how to use information derived from discourse context, and how to represent knowledge in a form that facilitates semantic interpretation. Another issue that is highlighted by Parsifal's clause-level orientation, is the question of how to deal with the 'special-purpose' phrases that seem to occur frequently in natural language. The last issue is the basis of this report.

Ideally, the constraints that allow such an elegant clause-level grammar would also allow the same simplicity in the description of more detailed phrases. Unfortunately, this is not the case. The grammars for numbers, names, and dates, which are 'lower-level' in the sense that they are always constituents of noun phrases, seem to lose any sense of either elegance or theory. It should be noted that Parsifal's grammars for these constructs are not necessarily any worse than grammars for such constructs in other English parsers; they simply fail to measure up to the quality of Parsifal's grammar for higher-level constituents. The contrast between the relatively ad-hoc implementation of lower-level constituents and the highly perspicuous representation for clause-level syntax leads to speculation that a different set of constraints might result in a better grammar for the lower-level phrases.

This report investigates the nature of these special-purpose phrases (which I will henceforth term 'conceptual phrases') in more detail. In addition, a method of representing them is presented that takes advantage of their distinct nature. Finally, the role of the resulting system is examined in the context of current research in natural language processing.

-Chapter 2 analyzes the existing grammar for conceptual phrases, and concludes that they correspond to single semantic entities (hence the name 'conceptual phrase').

-Chapter 3 develops a succinct representation for conceptual phrases by augmenting a semantic representation for them with a simple syntactic description.

-Chapter 4 gives a sample representation for numbers, while discussing many incidental issues.

-Chapter 5 discusses the problem of how to detect conceptual phrases.

-Chapter 6 summarizes the results of this effort.

2. Observations about the Behavior of Conceptual Phrases

While investigating the nature of conceptual phrases, it seems desirable to examine the existing Parsifal rules for lower-level constituents, and attempt to find properties and constraints that might allow better representation. The existing grammar includes rules for parsing numbers, times, dates, and proper names. In addition, rules have been constructed for parsing addresses, although they are not in the current grammar. These constructs are all 'lower-level' in the same sense as discussed previously, since they are all constituents of noun phrases.

Inspection of the grammar reveals that the rules for all these constructs follow the same general pattern. A packet of rules for each major type of constituent is introduced by an attention-shifting rule. The rules in this packet attach various constituents from the buffer to a node on to the stack. When no other rule applies, there is one default rule that deactivates the packet and restores the buffer.

It is unusual that each set of rules is introduced by an attention-shifting rule, considering that there are only two other attention-shifting rules in the entire system (NGSTART, which detects and parses noun groups, for use by clause-level rules, and POSSESSIVE-DET, which makes possessives 'look like' determiners to the noun-group parsing rules). As is noted in Appendix I, attention-shifting rules exist to allow the parsing of a phrase to be 'transparent' to the other rules in the system, which refer to the resulting constituent as if it were inserted into the buffer by the lexicon.

A noticeable difference between the rules for lower-level constituents and the other rules in the grammar is that many of the operations they perform consist of putting objects in registers and performing operations on the contents of registers. The only purpose of these actions is to semantically interpret the phrases parsed by the rule; they do not affect the structure of the final parse tree, nor do they influence the sequence of rules applied during remainder of the parse. Unlike the clause rules, there are no case rules for interpreting nodes built by the lower-level rules, since all semantic interpretation is performed by this manipulation of registers.

Yet another way these rules differ from most other rules is that many features used by the lower-level rules are semantic in nature. The rules for the highest level constituents (i.e. clause-level rules) test features that are clearly syntactic categories, or for the presence of a specific word. The rules for noun phrases also deal with features that are either syntactic categories or tests for individual words; however, there are some features (such as "quant", for a noun quantifier) that seem to reflect semantic properties of the words in addition to purely syntactic characteristics.

The rules for numbers, however, manipulate features such as: "ones" for a natural number less than ten, "tens" for words such as "twenty" "sixty", etc., "hundred+" for a number with the word "hundred" in it, and so on. Other than the tests for specific words, there are very few features in the number-parsing packet that are purely syntactic in the same sense as the features for higher-level constituents. This suggests that the syntax and semantics of lower-level constructs are more closely related than for, say, clause-level constructs. Specifically, the parsing process for lower-level constituents is primarily directed by semantic, not syntactic, categorization.

In addition to these observations, there are other facets of the behavior of lower-level constituents that might help us understand their true nature. These constituents behave like single lexical items in almost all contexts. For example, proper nouns behave exactly like nouns in all cases (justifying their designation), even though they may consist of several different words, or even several different constituents, each with its own substructure (for example, "Percy Flumbatz Memorial High School" could be analyzed as having the structure [[Percy Flumbatz] [Memorial [High School]]]). Also, numerals are treated exactly like single-word quantifiers in almost all contexts (except when they are used as arbitrary identifiers), even though they may contain more than one word.

Another aspect in which the behavior of low-level constructs differs from the behavior of higher-level constituents is that they do not appear to be a 'closed class'. Clause and noun phrase level syntactic structures are part of the permanent structure of the language. It is unlikely that a new top-level sentence structure would be added to a language under ordinary circumstances, or that a new order for noun-group constituents would become part of a language. It seems to be very difficult for a native speaker of a language to acquire the use and understanding of a new high-level construct in any natural way. On the other hand, it is not at all difficult to learn a different way of communicating a date (e.g. "23 June 79" instead of "June 23, 1979"), or to learn a new kind of name that provides different types of information.

The important point to abstract from these observations is that conceptual phrases represent a cross between clause level structures (such as complete sentences) and individual lexical items. They have the clause-like characteristics of structure and syntax, but also have properties that are 'lexical' in nature, such as representing single semantic classes, being freely added to the language, and filling the syntactic role of single words. The fact that Parsifal fails to recognize this fundamental difference accounts for the lack of perspicuity in the current grammar for numbers, names, times, and other conceptual phrases.

It should also be noted that the meanings of conceptual phrases could be conveyed using the general syntactic mechanism for clauses and noun phrases. Indeed, they may well have developed from such an approach. Consider the derivation of the name John Smith ("John the smith") or Fred Johnson ("Fred, son of John"), which have been derived historically from shorter constituents combined by higher-level syntactic mechanisms. Consider another type of conceptual phrase: numbers. In Britain, it is not unusual to hear phrases like "several millions of pounds" or even "five millions of dollars", where the "millions" is regarded more as a noun than can be quantified in the same manner as any other noun, including the application of rules of number agreement. This, in conjunction with the vestigial presence of the optional word "and" in number phrases, makes it clear that numbers have been thought of historicaly as smaller constituents conjoined by the mechanisms of clause and noun-phrase-level grammars.

If conceptual phrases were communicated using the same syntax as higher-level phrases, the only difference between them and other syntactic structures of the same form would be that they correspond to objects of a single semantic class. This leads to the view that the unique syntax that seems to exist for each conceptual phrase is an 'optimized' mechanism for communicating them. The additional constraints on the semantics of conceptual phrases allow some syntactic information to be left out that would otherwise have been provided by the 'standard' grammar; yet the ability of the listener to recover all of the useful semantic information is retained.

Of course, the constraint that a phrase must represent a single semantic class is not necessarily sufficient to allow a more efficient communication protocol. In addition, the members of the semantic category must have enough in common structurally so that significant assumptions can be made about the role of various constituents being communicated. For example, if addresses in the United States were not formalized at all, and consisted simply of descriptions of locations ("the second white house along the small alley next to highway that runs by the Flumbatz place"), it would not be possible to communicate them efficiently.

Now that we have arrived at some conclusions about the general nature of conceptual phrases, the original engineering problem of how to represent them in a manner that exploits their apparent simplicity still remains unsolved. The assertion that each type of conceptual phrase has a corresponding optimized syntax does not provide a method for deriving that syntax. It also does not provide a method for 'decoding' conceptual phrases. The next chapter is devoted to developing techniques that do.

3. A Combined Syntactic and Semantic Representation

3.1 Introduction

This chapter presents a technique for specifying the syntax of conceptual phrases in terms of their semantic representations. A prerequisite for this feat is the ability to represent the semantics of conceptual phrases, preferably by a means that has been implemented on a computer. There have been numerous attempts to design 'knowledge representation languages', with varying degrees of success. Some of the better known systems (in the Artificial Intelligence community, at least) are KRL [1], FRL [8], OWL [6], and NETL [3].

I selected the language FRL ('Frame Representation Language') for the purposes of this report, because it is implemented in MACLISP, which makes it possible to integrate it with Parsifal, and it is also (relatively) computationally efficient. Although FRL has been criticized on the grounds of insufficient expressive power [2] [9], it includes most of the features common to knowledge representation languages, and I have found that an FRL subset is sufficiently expressive for representing conceptual phrases. Most of the remaining discussion of semantic representation will be in the vernacular of FRL, so the reader unversed in it is urged to read the description in Appendix II before continuing.

3.2 Protocols for Frame Communication

In the previous chapter, it was determined that the major distinction between conceptual phrases and higher-level constituents is that everything that can be expressed by a particular type of conceptual phrase belongs to the same semantic class. In FRL terminology, every frame expressed by a single type of conceptual phrase is an *instantiation* of a generic *frame*. The approach used to combine the syntactic and semantic descriptions of conceptual phrases is to augment this generic frame with a syntactic description, which can be thought of as a formula for encoding and decoding the information in the frame.

I will defer the details of the syntactic description for now, in order to develop a detailed model for the process of communicating a particular instantiation of a generic frame. This model will clarify the reasoning behind the form of the syntactic description.

Consider the following situation: two people are attempting to communicate via speech. The 'speaker' wants to communicate a particular frame to the 'listener' (i.e., the speaker has a specific frame in mind, and wants the listener to 'build' that frame). For the time being, assume that listener already knows that this frame is an instantiation of a generic frame, and what the generic frame is. Both communicants have agreed on a protocol for communicating any instantiation of this generic frame (because they speak the same language). What type of protocol would suit the purpose of communicating the frame?

One simple, though inefficient, protocol would be to transmit each slot name of the frame immediately before or after its value. This would ensure that the listener could construct a duplicate of the speaker's frame without difficulty. However, this protocol fails to exploit the information that could be passed, implicitly, by the order in which the slot values are communicated.

An improved protocol would specify the order in which slot values should be communicated. The amount of information transmitted would be halved, since the slot names would no longer be transmitted. However, this protocol fails to take advantage of the knowledge that some slots are much more likely than others to have a particular value and that the communicants may not know or care about the values of other slots. In either case, the value need not be communicated explicitly. In the first case, the absence of a transmitted value would indicate to the listener that a default value should be used, and in the second case, communicating the value would be pointless. This would amount to a major loss in efficiency if most of the slots didn't need to be communicated most of the time.

A protocol is needed that can take advantage of the above optimizations in order to communicate with maximum efficiency. I propose the following protocol:

1.The values of slots should be communicated in a fixed order.

2.Reasonable defaults should be assumed for slots where one particular value is clearly more probable than any others.

3.The optional communication of slots may make it difficult to figure out to which slot a certain value corresponds. If there is any ambiguity, another word should be transmitted either immediately before or after that value to indicate the proper slot (this extra word will be referred to as a *synchronizer* later).

3.3 The Syntactic Description

This report takes the position that the above protocol is more than just an engineering exercise. In fact, the syntactic specification will be based on the assumption that this is actually how conceptual phrases are communicated by people.

The specific syntactic specification is tailored to the needs of the listener, who must decode the conceptual phrase. The listener must be able to compare constituents in his 'buffer' (under the assumption that he has a mental Parsifal in operation) with the restrictions specified in the frame on the possible values of each slot. Consequently, the syntactic specification for the frame must provide for tests on the contents of the buffer to check whether a given constituent can 'fit' into a particular slot in the frame. Also, he must be able to use synchronizers to select the correct slot for a constituent.

The specific representation I have chosen meets all of the above requirements. The frames are ordinary FRL frames, but there is an additional specification for the syntax of any frame (there is nothing to prevent two frames from sharing a syntactic specification, or one frame having multiple syntactic specifications). The syntactic specification consists of an ordered list of 'slot specifications' (in the same order as the slots are to be communicated). Each slot specification consists of a buffer pattern associated with name of one of the slots in the frame. The buffer patterns resemble the buffer patterns in normal Parsifal grammar rules except that they never examine a node on the stack.

If the 'listener' in the earlier communication scenario is Parsifal, with this specification technique for conceptual phrases, the communication process would proceed according to the following simple algorithm:

1.create a generic frame for this conceptual phrase and put it on the active node stack (assume that frames can be nodes).

2.compare each slot specification (pattern/slot pair) of the frame's syntax with the buffer. If the pattern matches, put the contents of the first buffer cell in the specified slot, and throw away all other constituents matched by the pattern. If the pattern does not match, do nothing and continue to the next pattern.

3.when the entire pattern has been exhausted, pop the frame currently on the stack into the first cell of the virtual buffer.

The stack should be in exactly the same state immediately after the execution of this algorithm as immediately before. The effect of the rule is to collect all the words of the conceptual phrase into a single frame, which replaces them in the buffer. From the point of view of the higher-level rules, the frame might as well have come directly from the lexicon.

3.4 Relation between the Protocol and the Syntactic Description

It is important to note that this algorithm does not allow the full range of syntax permitted by the general protocol presented earlier. For example, the restriction that only the contents of the first buffer cell can be put in a slot (see step 2) is a simplification based entirely on the observation that it seems to work empirically in English.

There is another constraint on the algorithm that does not exist for the protocol, but this one has independent justification. Remember that the protocol requires synchronizers when the correct slot for a value is not obvious from the conditions specified on that slot's value (via the $REQUIRED facet). The algorithm does not look through the possible slots, testing arbitrary predicates on a buffer constituent. Instead, the conditions must be checked entirely by feature tests on the contents of the cells of the buffer.

Theoretically, the features in the buffer could be made arbitrarily complicated, to reflect the results of computing all the predicates that occur in all the slots of all the frames. This would allow the feature test to have the same power as the process of testing all the predicates during the matching process (in actuality, this is just a pre-computation of the predicates). This is counterintuitive, computationally inefficient, and inconvenient, since it would require that all the features on constituents in the lexicon be updated whenever a new condition is placed on a slot. The process of testing the actual predicate at 'parse-time' instead of 'definition-time' (by analogy with compilers) would probably be better.

However, there is a computational justification for testing features that do not represent all the potential complexity of the predicates on a frame's slot. The $REQUIRED predicates can be arbitrarily complicated. For a human to parse a conceptual phrase in real-time, the complexity of the tests must be bounded in some way. If the tests can only check for an appropriately constrained set of features on buffer constituents, this requirement is fulfilled nicely. Even if the semantic restrictions on the value of a slot are not exactly equivalent to some simple feature test, it is possible that these restrictions can approximately be met by such a test.

This aspect of the algorithm is consistent with the observation of the previous chapter that the semantic objects corresponding to a conceptual phrase must be constrained in some way besides simply fitting into a single semantic class. That constraint can now be expressed in terms of the computational effort required to determine whether a particular value can 'fit' into a particular slot. As observed in chapter 2, the features tested by the current rules for conceptual phrases correspond either to tests for particular words or for semantic features (i.e. semantic categories). Perhaps the additional constraint of that chapter is that any frame expressible by a conceptual phrase must consist of a finite set of predesignated slots, whose conditions may be approximated by tests for individual words or the semantic classes of constituents.

3.5 The Representation and Meaning

This approach to the representation of conceptual phrases has a significant implication with respect to the nature of meaning. A commonly held view (even when not stated explicitly) is that a single 'concept' has a single canonical semantic representation, which is used whenever the meaning of something is needed (for communication, formation of other concepts, or some form of manipulation). This suggests a picture of natural language understanding as a transformation from a string of words in the language to this idealized concept.

The representation here would run into severe problems when evaluated in such a framework. Since the semantic representations must be chosen to reflect the structure of the syntactic representations, they may not be in the idealized form that tries to be all things to all people. If the structure of the semantic representation is not chosen to reflect the syntactic structure, the rules for translating from the semantics to the syntax, and vice versa, become extremely complicated.

In order to be able to use the semantic representation as a basis for the syntax of a conceptual phrase, and still retain the suitability of the semantics for other types of manipulation (e.g. structure building, computation, or deductive inference), we will have to take a different view. One view that seems to be particularly appealing is that the 'meaning' of something is not just a single semantic structure: it is a set of such structures, each suited for a special purpose or group of purposes. Such a system would need rules and procedures for translating among the various special-purpose representations.

The frames we have been referring to as 'semantic representations' are actually just a part of such a system. This is not to say that the only thing they are good for is communication (this would render the term 'semantic' meaningless). First, the existence of the generic frame provides a point of reference for finding various associations with an object. For example, once an address phrase has been parsed into an address frame, we have immediate access to all the generic properties of addresses. We also have a frame to match against memory to see if one can derive any additional information from knowledge about this particular address. Finally, such frames (or something vary close) could conceivably be the basic item manipulated during verbal reasoning, when people translate their thoughts into language to help them 'think things through'.

4. Numbers: A Sample Representation

4.1 A Representation for Numbers

This chapter develops a representation for English numerals (American dialect), in the form of syntactically augmented frames as described in the previous chapter. Although most of the discussion in this chapter relates specifically to numbers, it should understood that many of these issues are common to the representation of many other types of conceptual phrases. The representation of numerals was chosen as an example because they are a particularly clear and well understood form of conceptual phrase.

Below is a sample set of frames for natural numbers (less than one trillion). In this representation, numerals are actually represented as several different types of conceptual phrases, some of which can be contained within others. The digits "one" through "nine" are all in the lexicon as members of the semantic class of ONES. Numbers between twenty and ninety, inclusive, that are multiples of ten are in the semantic class of TENS. The semantic class SMALL-NUMBER also contains all legitimate combinations of TENS and ONES, such as "thirty six" and "ninety nine". Also, there are some SMALL-NUMBERs entered in the lexicon, namely, the numerals "ten" through "nineteen". The conceptual phrases for TENS and ONES can be constituuents of a SMALL-NUMBER. Another conceptual phrase is HUNDREDS: any SMALL-NUMBER followed by the word "hundred", optionally followed by another SMALL-NUMBER. Note that this allows phrases such as "fifty four hundred", which are not formally correct, but are almost universally used. Finally, the BIG-NUMBER frame includes those phrases that have the words "thousand", "million", or "billion" in them. This system cannot recognize numbers greater than "999 billion".

The frame for a SMALL-NUMBER is shown below. All LISP functions have been paraphrased in English, and is delimited by '%'.

```
(FRAME SMALL-NUMBER
   (TENS ($DEFAULT (ZERO))
         ($REQUIRED (%the value must be a kind of TENS%)))
   (ONES ($DEFAULT ZERO)
         ($REQUIRED (%the value must be a kind of ONES%)))
   (AKO ($VALUE (NUMBER)))
   (QUANT ($IF-NEEDED (%calculate the value by adding the
                        values of the TENS and ONES slots%))))

(SYNTAX SMALL-NUMBER
   ([=TENS] TENS)([=ONES] ONES))
```

The $REQUIRED slots in the above frame are entirely redundant, since they correspond exactly to the features specified in the syntactic description of the frame. The only reason they are defined here is to emphasize that, according to the analysis of the last chapter, the feature tests of syntactic description are an efficient approximation to constraints on the possible values of a slot. The constraint is expressed in the $REQUIRED facet, even though the syntactic description enforces it exactly, in this case.

For the conceptual phrase "fifty six", this frame would test if the constituent in the first place of the buffer had the feature TENS, put it in the TENS slot of a SMALL-NUMBER frame, then test "six" for the ONES feature, and put it in the ONES slot. The constituents for "fifty" and "six" would be frames themselves, but these frames are 'prefabricated', and associated with root words in the lexicon.

The $DEFAULT facets in the TENS and ONES slots of the frame indicate that if no value is available to put into a frame explicitly, the hearer should assume that it has the value "ZERO", if the value is needed (actually, in this model, getting a default value from a frame is a more basic action psychologically than 'assuming' something).

The result of parsing the conceptual phrase "fifty six" is the frame:

```
(FRAME SMALL-NUMBER11
        (TENS ($VALUE (FIFTY)))
        (ONES ($VALUE (SIX)))
        (AKO ($VALUE (SMALL-NUMBER)))).
```

As in any other FRL frame, this frame inherits any properties that are not explicitly defined in the frame from other frames in the AKO hierarchy. In this case, the frame is an instantiation of the SMALL-NUMBER frame, so among the inherited properties would be the default values for its slots and the $IF-NEEDED facet of the QUANT slot.

Notice that the QUANT slot does not appear in this frame. The value of the QUANT slot is the 'computer' concept of a number (i.e. LISP 'fixnum'). There are at least two ways this could be computed. The first would be to have a LISP program associated with both of the TENS and ONES slots which will calculate the value and add it to the contents of the QUANT slot. The other approach, which is the one used here, is to have a LISP program which calculates the value of the QUANT slot only if it is asked for. This method is more compatible with the view of 'meaning' taken in the last chapter, that a concept is a group of different representations, each suited for specific uses. If the the number is going to be used for computation after being parsed, the machine representation is clearly better. On the other hand, if the number is used to find associated knowledge (e.g. a year in history), the computer representation is not needed, and should not be computed.

The frame for HUNDREDS is:

```
(FRAME HUNDREDS
   (NUMBER-OF-HUNDREDS ($REQUIRED (%must be less than 100%)))
   (REST-OF-HUNDREDS ($DEFAULT (ZERO))
                    ($REQUIRED (%must be less than 100%)))
   (AKO ($VALUE (NUMBER)))
   (QUANT ($IF-NEEDED (%100 times the quant of the value of the
                       NUMBER-OF-HUNDREDS slot, plus the quant
                       of the value of the REST-OF-HUNDREDS
                       slot.%))))

(SYNTAX HUNDREDS
   ([* IS ANY OF SMALL-NUMBER, TENS, ONES][=*HUNDREDS]
    NUMBER-OF-HUNDREDS)
   ([* IS ANY OF SMALL-NUMBER, TENS, ONES]
    REST-OF-HUNDREDS))
```

(In the pattern for each slot, "* IS ANY OF" means that the pattern should match if *any* of the features are present. Also, a pattern of the form '*FOO' means 'match on the presence of the word "foo".)

In this case, the presence of the word "hundred" is a synchronizer; it differentiates between the conceptual phrase for the NUMBER-OF-HUNDREDS slot and the conceptual phrase for the REST-OF-HUNDREDS slot. Also, note that the SMALL-NUMBER rule should be transparent (e.g. "fifty six" should appear to be a single lexical) to this rule, since the HUNDREDS rule can have a SMALL-NUMBER as a constituent.

The generic frame for a BIG-NUMBER is represented as follows:

```
(FRAME BIG-NUMBER
   (BILLIONS ($DEFAULT (ZERO))
             ($REQUIRED %the value must be less than 1000%))
   (MILLIONS ($DEFAULT (ZERO)))
   (THOUSANDS ($DEFAULT (ZERO)))
   (REST-OF-BIG-NUMBER ($DEFAULT (ZERO)))
   (AKO ($VALUE (NUMBER)))
   (QUANT ($IF-NEEDED
             (%take the sum of the slots after multiplying the
              QUANT of the value of each one by the appropriate
              power of 10%))))

(Note: the $REQUIRED facets of the other slots have been
left out to avoid needless repetition)

(SYNTAX BIG-NUMBER
   ([* IS ANY OF HUNDREDS, SMALL-NUMBER, TENS, ONES] [=*BILLIONS]
    BILLIONS)
   ([* IS ANY OF HUNDREDS, SMALL-NUMBER, TENS, ONES] [=*MILLIONS]
    MILLIONS)
   ([* IS ANY OF HUNDREDS, SMALL-NUMBER, TENS, ONES] [=*THOUSANDS]
    THOUSANDS)
   ([* IS ANY OF HUNDREDS, SMALL-NUMBER, TENS, ONES]
    REST-OF-BIG-NUMBER))
```

This frame has many more synchronizers than the other frames in the numeral system, since there are many adjacent slots with identical semantic conditions. Also, this rule has a case of a syntactic condition approximating a semantic condition on a slot. Phrases such as "fifteen hundred" are allowed as constituents, (allowing "four million fifteen hundred thousand") since they are legitimate HUNDREDS. This situation could have been avoided by requiring a HUNDREDS to have a ONES as its first component, instead of a SMALL-NUMBER, or by creating an entirely new conceptual phrase for parsing numbers of the form ONES followed by a HUNDREDS, etc. I chose the representation presented here since, on the basis of informal experimentation, it seems that people did not notice the incongruity of numbers such as "one million fifty six hundred thousand five" until asked to write them down or perform arithmetic on them*.

Finally, I should note that this representation has left out at least two minor details of the grammar for natural numbers. First, conceptual phrases in the HUNDREDS category have an optional "and" after the word "hundred" (e.g. "six hundred and fifty seven"). This can be expressed very easily in the representation by allowing a slot-specification with no slot name, which would effectively discard the "and".

Alternatively, an *additional* syntactic description could be constructed for the HUNDREDS, having a slot specification of the form

```
(SYNTAX HUNDREDS
        ([=SMALL-NUMBER, TENS, ONES] [=*HUNDRED] [=*AND]
         NUMBER-OF-HUNDREDS)
        ([=SMALL-NUMBER] REST-OF-HUNDREDS))
```

accomplishing the same goal.

Another irregularity that is not accounted for here is the occurrence of phrases such as "a million and ten thousand", where a leading "one" can optionally be an "a". This should probably be taken care of elsewhere in Parsifal; since "a" and "one" have the same quantificational properties, perhaps the lexical phase of the parser should convert them into the same constituent.

4.2 The Validity of the Representation

Due to a lack of evaluation procedures, it would be very difficult to determine whether the numeral representation presented is 'the correct one'. A supporting point for this type of representation is the perspicuity of the

*A. Van Katwijk [4] makes a similar observation to justify the omission of context-sensitive rules in a grammar for Dutch numerals.

representations of individual conceptual phrases. The representation given here manages to express a broad class of numerals fairly succinctly. Even if another system could arrive at a more perspicuous representation for numerals in one of the domains of syntax or semantics, it seems unlikely that this could be achieved without greatly complicating the other domain. In addition, considering the earlier observation that conceptual phrases are an optimized means of communicating a member of a constrained semantic class, the fact that this representation manages to capture the inter-relationship between the semantic and syntactic knowledge for these expressions increases its attractiveness.

Additionally, a numeral in this representation can be useful for operations other than communicating or retrieving associated information. Although it would not be as suitable for some arithmetic operations as another representation might be, it would be adequate for many common operations. For example, two SMALL-NUMBERS can be compared easily in size by the simple algorithm of comparing the TENS (via 'table lookup') and if they are the same, comparing the ONES. It is also very easy to compare cardinal numbers of different semantic types. In general, all the members of any semantic class are either greater than or less than all the members of any distinct semantic class (the exceptions are comparing TENS and SMALL-NUMBERS, which can be tested by comparing the TENS slot of small-numbers, and "fifty six hundred" vs. "five thousand six hundred".**).

Also, simple addition and subtraction of numbers in this representation is fairly simple (both in terms of computation and short-term memory), as long as no

**If it could be shown that people empirically have greater difficulty in comparing numbers like "forty five hundred" with "six thousand three hundred" than with "sixty three hundred", it would lend support to the hypothesis that the semantic representations for these forms are different 'carrying' is required. Indeed, the fact that these operations are so much more difficult when 'carrying' is required, may indicate that this representation is the 'default' for general-purpose conceptualization.

5. When to Expect a Conceptual Phrase

When the protocol for frame communication was developed in Chapter 2, it was assumed that the listener knew which generic frame was being communicated by the speaker. Until now, the method by which this frame is recognized has remained unspecified. Since the syntactic description of the frame does not contain enough information, a priori, to determine when the frame applies, we must provide either some external indication of the 'triggering' condition, or some heuristic for deriving that condition from the syntactic description.

As an added complication, the recognition process cannot be entirely syntactic in nature. Consider the phrase "nine thirty" in isolation. This could be a time of day (9:30), a price ($9.30 or $930), part of an address ("930 Percy St."), or almost any type of number, depending on the context and the domain-specific expectations of the listener. This is exactly the same problem that occurs when trying to determine which word sense to associate with a given word, and once more indicates the partly lexical nature of conceptual phrases. Assuming that discourse context is not sufficient to guarantee that only one possible conceptual phrase can occur, there must be some syntactic preference involved.

For engineering purposes, it would probably be adequate to construct the triggering information for these rules by hand and represent it explicitly as a Pidgin rule. It would be much more convenient, however, to have an automatic 'trigger generator' produce the triggering information from the specification of the frame. In either case, some decisions must be made as to how the rules should interact, since this interaction has a major effect on the interpretation of the rules.

As an example of how the specific triggering conditions for a frame can affect the final result of the parse, consider the specification for numerals presented in the last chapter. If we adopted the philosophy that a frame should trigger whenever something appears that fits in one of its slots, the following scenario could occur: the word "one" could occur at some point in the input. It would be associated with the ONE frame in the lexicon, which would be in the semantic class ONES, and insert it into the buffer. The SMALL-NUMBER frame could then trigger, since the object in the semantic class ONES can fill the ONES slot of that frame. This would leave a SMALL-NUMBER in the buffer, associated with the phrase "one". Similarly, the HUNDREDS and BIG-NUMBER frames would trigger, and the result of the process would be the appearance of a BIG-NUMBER constituent in the buffer, in place of the phrase "one".

This type of behavior has problems in the interaction with the rest of the grammar. Since cardinal numbers are frequently used as quantifiers, the rule for number-agreement applies. Any noun quantified by the number "one" is singular, while a noun quantified by any other number is plural. If all cardinal numbers were BIG-NUMBERs distiguished only by the contents of their frames, the number-agreement rule would have to examine the *contents* of the BIG-NUMBER frame in order to function correctly. However, number agreement seems to be more of a syntactic than semantic phenomenon*** . In addition, if semantic description and syntactic descriptions for SMALL-NUMBERs, for example, already exist it seems artificial to say that they have no independent existence, except to trigger the HUNDREDS rule.

A less liberal triggering philosophy that required the phrase for, say,

***consider the acceptability of "1.0 cavities"

HUNDREDS to contain at least one other constituent besides a single SMALL-NUMBER would have a much different result. If a similar triggering criterion held for the other frames, each different type of number could without being part of another frame.

Even if the representation for numbers could be reformulated to avoid this difficulty, it would still be difficult to control interactions between frames, in general. There is no restriction in the representation to prevent different frames from having slots that permit the same types of constituents, under the same conditions. If this situation occurs in the grammar, and it seems probable that it will, there will have to be some general principle to determine which rule should trigger.

It seems to me that the proper behavior of the rules should be to prevent frames that contain only one constituent from occuring, whenever this is practical. I propose the following more specific principles for determining the triggering conditions for a frame:

1.All frames must trigger on at least two buffer constituents.

2.If two frames can trigger at the same time, then

> If one of the frames is a constituent of the other ('constituent' here means related by the transitive closure of the 'immediate constituent' relation), it should trigger first.
>
> If neither is a constituent, hope that some other mechanism, such as discourse context, will give one rule preference.
>
> If each rule can be a constituent of the other, the representation is probably wrong anyway. I'm certainly not going to worry about it.

If the triggering rules interact so as to adhere to this principle, the effect would be that the frame that triggers would be the 'lowest-level' conceptual phrase (thinking of the frames in terms of a constituent hierarchy). There should be no need to allow a frame that contains only one constituent, since that constituent would be a single semantic entity, and should be useful by itself.

An implementation problem is raised by this approach. Since we have been assuming that the conceptual phrases are parsed into frames in a manner transparent to the other rules (including the ones for parsing other conceptual phrases), the only really convenient trigger mechanism is attention-shifting rules (which explains the use of attention-shifting rules in the current grammar). Unfortunately, the implementation of attention-shifting rules in Parsifal constrains them to trigger on a single buffer cell, in order to prevent violations of theoretical constraints on 'look-ahead' in the buffer. It would still be possible to implement triggering by having an attention-shifting rule match on the first cell of the pattern for triggering the frame, and then have other (normal) rules check the rest of the pattern against the buffer, and restore it if none match. This addition to the complexity probably indicates a missed generalization, either in Parsifal or in our representation.

Additionally, from an engineering point of view, it is not convenient to have to examine all the other rules in the system whenever a new conceptual frame is learned. At this time, the problem of how best to specify the triggering mechanism remains open.

6. Conclusions

6.1 Where Conceptual Phrases Fit In

One of the major unsolved problems of both linguistics and artificial intelligence is to define the relationship between language and knowledge. The solution, if there is one, would have profound impact on both fields, and probably on many other fields as well.

This research represents an attempt to find a solution to that problem on a very small scale -- to develop a representation that captures the relationship between the syntax and the semantics of what I have termed 'conceptual phrases'. The result is a specification that consists of a frame representation of an object, augmented by a syntactic description of the frame.

Conceptual phrases fill a niche between the various phases of the parsing process. The different types of linguistic constructs, measured along the axis of complexity in semantic interpretation, can be seen as a spectrum. At one end of the spectrum, there are 'sentences', which are exceedingly general constructs in terms of the meanings they can express. The users of the language pay a price for this generality (as with most computational tasks), in terms of verbosity and the difficulty of semantic interpretation.

At the other end of the spectrum are individual words. The process of associating a word with its meaning is very simple. Simply look it up in the lexicon. Words, by themselves, have a very limited range of meaning, however.****

****when I say "word" here, I mean an individual lexical item corresponding to a single word in a sentence, where all the syntactic and discourse constraints have already acted to select the correct word sense from a fixed list of alternatives

Conceptual phrases are somewhere between these two ends of the spectrum. They are more general than words, but less general than clauses, for communicating concepts. Their range of meaning is sharply limited, but there are many more possibilities than for single words. The process of semantic interpretation is much simpler than for sentences, but more complex than for single words. It is interesting to speculate whether there are additional intermediate stages, each with its own special tradeoff between expressive power and efficiency.

There seem to be at least two major schools of thought, among linguists and in artificial intelligence, about the relationship between syntax and semantics. One school contends that the processes of language understanding and acquisition are merely facets of a general cognitive mechanism, and that syntax is an arbitrary set of rules for converting knowledge structures to linear streams of words, for the purpose of communicating it. The other school believes that syntax and semantics are separate (though connected mechanisms), and that people have a special 'cognitive processor' or processors that is devoted to the manipulation of language.

Parsifal has been most strongly influenced by the second line of thought, yet the mechanism proposed herein is highly reminiscent of the first philosophy. The combination of Parsifal's grammar language for higher-level constituents and this specification for the lower-level ones is a synthesis of the two viewpoints.

6.2 Directions for Future Research

In order to have a practical system for specifying conceptual phrases, there are some problems which still must be dealt with. First, there is the problem discussed in Chapter 4 of how to generate 'triggering' information from the augmented frame specification. This is part of a larger question about exactly how the frames should interact. Another task would be to determine how much power is required in the syntactic description, preferably stated in terms of linguistic universals rather than as ad-hoc observations about conceptual phrases in English.

There are interesting linguistic questions about conceptual phrases, if the analysis given here is correct. The communications model developed in Chapter 2 seems to imply that the process of generating a conceptual phrase is different from the process of composing a sentence. Can the representation given here be modified to make it bi-directional (suitable for the speaker as well as the listener)?

Additionally, there is the question of how conceptual phrases are learned. Since the syntax seems to be simpler than for higher levels, are they easier to learn? Chapter 2 mentions that conceptual phrases can be added to the language relatively freely. Does this imply that the process of learning conceptual phrases is substantially different than the process of learning other syntactic rules? The principles developed in Chapter 5 could be implemented as an algorithm which checks for potential triggering conflicts with other rules. Does this imply that the decision about how the frames should interact is made when the rule is acquired?

I. A Description of Parsifal

This appendix is meant to provide a description of those parts of Parsifal which are necessary for understanding the report. Parsifal is an English parser developed by Mitchell Marcus of the MIT Artificial Intelligence Laboratory. Unless otherwise noted, this description is abstracted from his Ph.D. thesis [5]. The reader is referred to that document for a detailed description of Parsifal and its implications for the fields of linguistics and artificial intelligence.

Marcus calls the underlying philosophy of Parsifal "strict determinism" (not to be confused with the concept of determinism in automata theory, which is far too broad to be of use in this context), but Swartout [10] claims that a more accurate view of the parser's operation would be that it adheres to the "principle of least commitment" -- it defers decisions until the latest possible moment, and when it makes a decision, it considers it final.

In addition to determinism, another goal of Parsifal is to tailor the capabilities of its 'abstract machine' (consisting of the data structures and the operations which may be performed upon them) as closely as possible to the task of syntactic parsing of natural language. This is done in the hope of allowing very brief and powerful definitions of natural languages that should provide insight into the psychological mechanisms at work when a person understands language. In addition, this tailoring should simplify the task of constructing grammars for practical natural language understanders.

Parsifal was designed with the theory of generative grammar very much in mind, and maintains the position that syntactic and semantic mechanisms form a nearly-decomposable system. In other words, there is a separate 'syntactic processor' that interacts in some limited way with the mechanisms that actually derive the meaning of a construct. This contrasts with many approaches to natural language understanding that assume (implicitly or explicitly) that there is no separate mechanism for syntactic analysis.

Marcus calls Parsifal a "grammar interpreter", since it can be divided conceptually into an "abstract machine" which interprets a set of "grammar rules", which are responsible for directing the parsing process. The abstract machine implemented by Parsifal consists two major data structures.

The first is a buffer, which consists of a series of three "cells" which may contain "constituents". A constituent is any grammatical structure, including the final parse tree, various subtrees and individual words. The rules may insert constituents into the buffer or delete constituents from it. The buffer is a 'compacting buffer', meaning that when an object is inserted, the contents of the buffer will 'move aside' to create an open cell for it, and when an object is deleted, the remaining contents will be shifted to 'close the gap'. A rule may only delete an constituent from the buffer in the process of attaching it to another constituent. Rules are allowed to insert constituents arbitrarily.

The buffer just described should be called the 'virtual buffer', because it is actually a three-cell window into a larger buffer, of unspecified length. No rule may examine, insert, or delete the contents of any cell outside of this window. There is an operation called 'offset' that allows this window to be shifted three or fewer cells to the right, and an operation 'restore' which moves the window back to its position before the last 'shift'. This capability is only used by "attention-shifting rules", which will be explained slightly later.

The second data structure used in Parsifal is a stack. It functions as temporary storage for constituents being constructed. The only way anything can be "pushed" onto the stack is by the creation of a "node". When a node is created, it becomes the "current active node", and stays on the top of the stack until another node is created, or until it is removed. Nodes are a type of constituent, and the terms will be used interchangeably in the rest of this report.

A rule can attach a constituent in the buffer to the current active node. When this occurs, the constituent is removed from the buffer (this is the only time the delete operation can occur). Nodes may be removed from the stack, in which case they are inserted into the first position of the buffer (relative to the virtual buffer start), and the next node on the stack becomes the current active node. These operations are used in building higher-level constituents from lower-level ones. Typically, a rule will create a node, attach several constituents to it from the buffer, and then take the node off of the stack and put it in the buffer. The effect of this is to combine several constituents in the buffer into a single node, which replaces them in the buffer.

The contents of the buffer and stack are manipulated by grammar rules, written in Pidgin, Parsifal's grammar definition language. Grammar rules are defined in the form of pattern/action rules. A pattern consists of a description of the contents of the buffer and stack. This description consists of a boolean combination of tests for the presence or absence of "features" on the node occupying that position. These features can be manipulated by the "action" part of the rules, but are also associated with words in the lexicon, and may be modified by the morphology of a word before it is inserted in the buffer.

Grammar rules have two other important attributes. Each rule has one or more "packets" associated with it. For a rule to take effect, not only must its pattern match against the buffer, but its packet must also be "active". Those packets that are active at any time are associated with the current active node, and when a new node is pushed, the packets associated with the old node become inactive until it becomes the current active node again. Packets may also be activated and deactivated explicitly by the action parts of rules. The other attribute is the rule's "priority". If two rules match, and both are in currently active packets, the one with the lowest priority is run.

Finally, there is a special type of rule called an "attention-shifting" rule, which operates in a manner somewhat different from "normal" rules discussed above. An attention-shifting rule has a buffer specification which examines only a single cell. If the any constituent in one of the virtual buffer cells matches the pattern, the rule has the effect of shifting the virtual buffer so that buffer cell will be the first in the window. The virtual buffer will remain shifted into this position until it is restored by some other rule. The parser checks to see if any attention shifting rule matches before it tests the normal rules.

Attention-shifting rules are used to make constituents of one type 'transparent' to rules of another type. For example, rules that operate on "clause level" constituents in the parser have patterns which match on the presence of a noun phrase in a specific cell of the buffer. Since noun phrases usually consist of a combination of lower-level constituents, some set of rules must build them. Ideally, the process of parsing the noun phrase would be invisible to the clause-level rules, so that they could treat an them exactly like individual words.

This effect is achieved by an attention-shifting rule, which detects the presence of a "noun group start", shifts the buffer, activates a packet of rules which parse the noun group. A final rule restores the buffer.

Currently, there is a rudimentary case frame system which operates in parallel with the parsing process. The details of this system are irrelevant to the purposes of this paper, beyond the fact that they are the only form of semantic interpretation existing in the basic Parsifal system.

The result of the parsing process is what Marcus calls an "annotated surface structure", which consists of a tree of nodes, representing various types of phrases. Each node is labelled with a set of features that describe the phrase it dominates (for example, a single noun might have the feature "ns" on it, meaning that it is a singular noun. This feature is used by the rule which checks for number-agreement between modifiers and nouns). The process of parsing proceeds something like this:

> The word is analyzed by the morphology program, which attempts to find its root word in the lexicon. The root has various features associated with it. The morphology program may modify some of these features or add some of its own, depending on the word ending (for example, the morphology program may decide that because a word ends in "-s" and is a noun, that it is plural, in which case it will mark it with the "npl" feature). This root word, with features, is inserted into the left-most empty cell in the virtual buffer.
>
> Some rule may trigger off of the buffer at this time. Say, for example, that the first constituent in the buffer is an "the". The the attention-shifting rule "NGSTART" will trigger, activating a packet of rules which attempt to parse a noun phrase. The rule "NGSTART" will create an "NP" node (which is defined to make it the current active node). Successive rules will then attach the contents of various cells in the buffer to the new node, until the end of the noun group is detected. Then the (completed) NP node will be dropped into the buffer, along with all the features placed on it during its construction, and the virtual buffer will be restored to the same absolute position that it occupied before NGSTART triggered. If the sentence is grammatically correct, other

rules will trigger on the completed NP, forming higher-level constructs, until the entire sentence is parsed.

Here are some example rules in the grammar:

```
{RULE IMPERATIVE IN SS-START
[=tnsless] -->
Label c imper, major.
Insert the word 'you' into the buffer.
Deactivate ss-start. Activate parse-subj.}
```

The rule is named 'imperative'. It is in the packet SS-START. It triggers if the constituent in the first cell of the buffer has the 'tnsless' (tense-less) feature (this is the pattern). If the pattern matches, it puts the feature 'imper' and 'major' on the current active node (represented by 'c' in Pidgin rules). It inserts the word 'you' in the buffer (transforming the sentence to a declarative). I then deactivates the packet that it is in and activates the packet for parsing declarative sentences. It is a testimony to the design of Parsifal's abstract machine that this one rule is the only addition to the grammar for declarative sentences that is necessary for it to parse imperatives.

A rule in the grammar for numbers is:

```
{RULE NINETY-NINE IN BUILD-NUMBER
[=tens] [=ones] -->
Label a new num node 99s.
Attach 1st to c as num1.
Attach 2nd to c as num2.
Set the quant of c to
            plus(the quant register of 1st,
                 the quant register of 2nd).
Transfer ord from 2nd to c.
Drop c.}
```

This rule matches on the presence of a constituent in the first place of the buffer with a 'tens' feature, followed immediately by a constituent with a 'ones' feature. It creates a new node (putting it on the stack, of course) called 'num', and puts the feature 99s on it. Then it attaches the two constituents from the buffer to it (deleting them from the buffer). It then calculates the computer word for the number, and drops the completed number into the first place in the virtual buffer.

II. A Description of FRL

FRL (Frame Representation Language) is a general knowledge representation language, implemented at the M.I.T. Artificial Intelligence Laboratory. The following approximate description is derived from "The FRL Manual" by R. Bruce Roberts and Ira Goldstein [8]. Since the FRL system is much more flexible and powerful than this brief description would suggest, the reader interested in its full capabilities is referred to that document.

FRL attempts to provide a practical implementation of Minsky's concept of a 'frame' [7]. The system centers around a data structure called a 'frame'. A frame has a name and 'slots', which can be thought of as named attributes that can hold information about the frame.

Each slot has a number of 'facets' associated with it. Each facet can be associated with a LISP object (often another frame). The facets are used for associating different types of properties with the slots of a frame. There are a fixed number of possible facets, and the only ones relevant here are the "$VALUE", "$DEFAULT", "$REQUIRED", "$IF-ADDED" and "$IF-NEEDED" facets.

The "$VALUE" facet may be associated with any LISP object, which will then be interpreted as the "value" of the slot. For example, if we had a PERSON frame with a NAME slot, we could make the name PERCY into the value of that slot by associating it with the $VALUE facet of that slot.

If an attempt is made to get the value of a frame's slot, and there is no $VALUE facet for that slot, the slot is inspected for a $DEFAULT facet. If there is such a facet, the object associate with it is returned as the value. If there is no $DEFAULT slot, the system attempts to find a $IF-NEEDED facet to the slot. If there is one, and it has a LISP function associated with it, that function is run. This feature is frequently used to compute values dynamically from other information in the system.

The purpose of the "$REQUIRED" facet is to allow restrictions to be placed on the possible values of a slot. An arbitrary LISP predicate can be associated with the "$REQUIRED" facet of a slot to test any value that is being associated with the "$VALUE" facet of the frame. If the predicate does not evaluate to T when a value is being added, the system complains.

There is one special slot in each frame, called the "AKO" slot ('a kind of'), that stores hierarchical information about the frames. The entire set of frames can be linked together in a hierarchy via this slot. A frame that has another frame as the value of its AKO slot is an 'instantiation' of the second frame.

The AKO hierarchy is used to allow information about a frame to be inherited. In general, a frame will inherit all the properties that are not specified in that frame, from all the other frames above it in the hierarchy. As a simple example of a case in which this feature would be useful, imagine a PERCY frame that is an instantiation of an AMERICAN frame. The value of the CITIZENSHIP slot of the AMERICAN frame could be set to USA, to avoid having to respecify it identically in all the instantiations of the AMERICAN frame. Thus, the value of the CITIZENSHIP slot of the PERCY frame would be USA, without explicitly defining it in the PERCY frame.

References

[1] Daniel G. Bobrow and Terry Winograd.
An Overview of KRL, a Knowledge Representation Language.
Cognitive Science 1(1):3-46, 1977.

[2] Jon Doyle.
Hierarchy in Knowledge Representation.
Technical Report Working Paper 159, MIT Artificial Intelligence Laboratory, 1977.

[3] Scott Fahlman.
A System for Representing and Using Real World Knowledge.
PhD thesis, Massachusetts Institute of Technology, September, 1977.

[4] A. Van Katwijk.
A Functional Grammar of Dutch Number Names.
In H. Brandt Corstius, editor, *Grammars for Number Names*, chapter 1. D. Reidel Publishing Company, Dordrecht-Holland, 1968.

[5] Mitchell Marcus.
A Theory of Syntactic Recognition for Natural Language.
PhD thesis, Massachusetts Institute of Technology, February, 1978.

[6] William A. Martin.
Descriptions and the Specialization of Concepts.
Technical Report TM 101, MIT Laboratory for Computer Science, 1978.

[7] Marvin Minsky.
A Framework for Representing Knowledge.
In Patrick Henry Winston, editor, *The Psychology of Computer Vision*, chapter 6. McGraw-Hill, 1975.

[8] Bruce R. Roberts and Ira Goldstein.
The FRL Manual.
Technical Report AIM-409, MIT Artificial Intelligence Laboratory, 1977.

[9] Andrew Schulert.
On the Evaluation of Representation Systems for Semantic Knowledge.
Bachelor's thesis, Massachusetts Institute of Technology, August, 1978.

[10] William R. Swartout.
A Comparison of PARSIFAL with Augmented Transition Network Grammars.
Technical Report AIM 462, MIT Artificial Intelligence Laboratory, 1978.

[11] Terry Winograd.

Procedures as a Representation for Data in a Computer Program for Understanding Natureal Language.
Technical Report TR 84, MIT Project MAC, 1971.

[12] William A. Woods.
The Lunar Sciences Natural Language Information System.
Technical Report BBN Report No. 2378, Bolt, Beranek, and Newman, 1972.