Knowledge Based Systems Group

Laboratory for Computer Science

June 21, 1977

# A Computational Approach to Modern Linguistics: Theory and Implementation

by

William A. Martin

- Draft Draft -

# A Computational Approach to Modern Linguistics

Theory and Implementation

by

William A. Martin

Volume I

# A Computational Approach to Modern Linguistics
Theory and Implementation

<u>Contents</u>

Acknowledgements

This work was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract #N00014-75-C-0661. I especially wish to acknowledge the informed suggestions and steadfast support of William Carlson, the contract monitor.

Terry Winograd and Andee Rubin were helpful to me when I first became interested in computational linguistics. Early versions of this work were developed with G. A. Gorry and Rand Krumland. The current version was inspired by an attempt to reformulate the earlier work to conform to ideas of Lowell Hawkinson, and owes much to discussions with him, Gretchen Brown and Peter Szolovits. The notation was designed primarily by Hawkinson. Students in the MIT A/I lab, in particular Mitch Marcus, have also been helpful.

Much of the material here has been discussed and debated at numerous meetings of the OWL system development team. Their efforts at implementation have raised many important issues.

Finally, Ellen Lewis has been outstanding in helping with the many revisions of this material.

## Preface

This work has been motivated by a desire to create knowledge based computer systems which can give advice in such areas as business, law, and medicine. An earlier version of these ideas was implemented as OWL I. This is a good introduction to the representation of knowledge to be used in OWL II. However, it does not discuss OWL II as a programming language, features of OWL I or II introduced as solutions to problems of implementation, or issues of computational efficiency. Therefore, anyone wanting to use OWL I, or OWL II, will need to seek information beyond that given here.

## 1.1 The Complexity Barrier

In the last two decades linguists have made impressive and accelerating progress in understanding the syntax and semantics of the English language. During the same period computational linguists have also made advances, constructing systems for language translation and for answering questions against a data base. However, by some measures, the progress of computational linguists has been disappointing. In many fields, the process of implementing theories in computer programs has led to new insights and understanding, and the practical value of the functions performed by the programs has led to increased support for and interest in the field. It is difficult to make this argument for computational linguistics. In large measure the insights of linguists have been more sophisticated and detailed than what computational linguists have been able to implement, so that the implementations have been too crude to force consideration of new linguistic issues. While the coverage of English grammar by linguists is quite complete, the coverage in implemented programs has been sketchy. There is uncertainty whether computer users could live within the subsets of English so far provided. Unless the situation places the discourse in a limited domain, it seems clear that they can not.

The problem has been one of complexity. Every major natural language program so far produced is so complex that it taxes the capabilities of even its author to understand it. This <u>complexity barrier</u> has been reached before the program encompassed all the major syntactic structures of English and with only the simplest of semantics. This problem of the complexity of the description of operational systems is one which linguists have not addressed. Use of the Augmented Transition Network (ATN) to represent the syntactically correct word orders of English was a major breakthrough by computational linguists. But this alone was not enough; the complexity barrier remained.

There have been two principal reactions to the complexity problem, the general and the particular. The generalist reaction is to provide the programmer with more tools. Through partial automation, details are removed from his consideration and he is given bookkeeping aids to augment his memory. Typically, the generalist feels that the human mind has some sort of general purpose machinery for building and using very complex information processing capabilities. To do as well in machines one must somehow match these facilities.

The particular reaction is to make a deeper study of the problem at hand, with an eye to simplification. Clearly, both approaches are important, but while there is no doubt that better programming aids will produce a reduction in complexity, it is unlikely to be a dramatic reduction. On the other hand, the reduction in complexity to be achieved by careful study of a particular problem is entirely problem dependent. It can be either dramatic or insignificant.

I believe that natural language is open to dramatic complexity reduction through careful study and insightful programming. In this book I describe a computational theory of English and an implemented system for processing English sentences which is more complete than those previously implemented. I will refer to this computational theory of English as CTE. Since any natural language must continually be learned by new generations it is not surprising that English has strong organizing principles. In fact, linguists have already identified many of these. So far, however, they have made little effort to unite these principles into systems for sentence processing and they have done relatively little work on what they term "perceptual strategies". This work, the formation of theories like CTE, remains to the computer scientist willing to familiarize himself with the structure of language.

The viewpoint adopted here is not to deny the complexity of language, but rather to claim that it is open to the techniques of good systems design - to the kind of thinking found in books on structured programming and in Simon's book, <u>The Architecture of Complexity</u>. For example, the theory of grammar presented in this book can be broken down as shown in Figure 1.1. In Chapter 2 a relatively few key ideas and decisions will be presented. Once these are determined, Chapter 3 will show how they can be used to solve a number of "hard nut" problems which face any language processing system. Extension to a basic grammar of English in Chapter 4 is then strongly determined by the decisions made to handle the "hard nut" problems. With a basic grammar in hand, one can go on to consider problems of word order introduced by stylistic alternatives such as <u>I picked up the block</u> vs <u>I picked the block up</u> and by the fact that sentences like <u>Do good deeds</u> and <u>Do good deeds help?</u> have an identical left segment. Finally the question of parsing strategy is addressed.

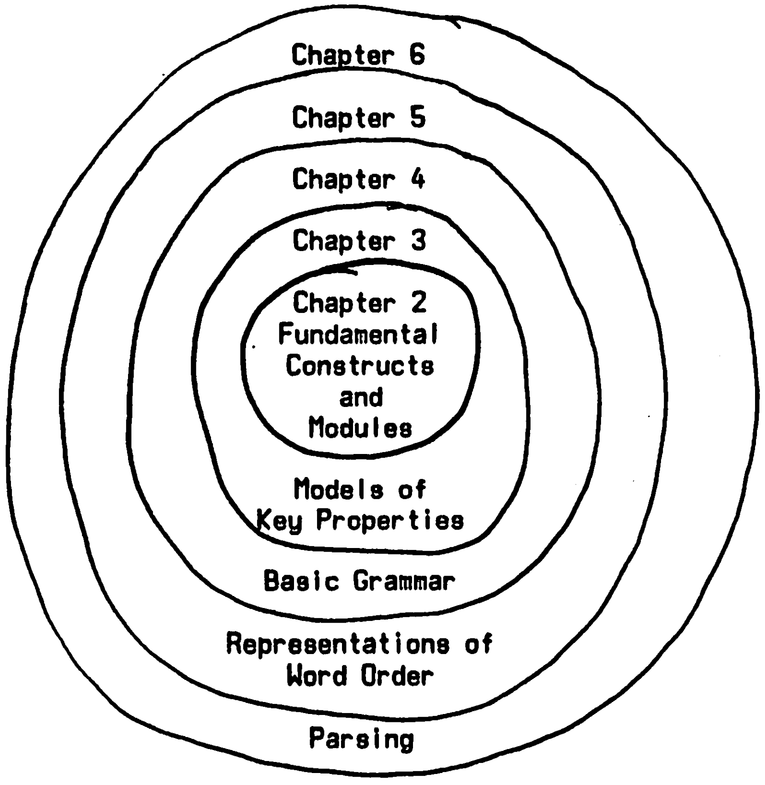

Figure 1.1
Hierarchical Structures of a Theory of Grammar

A good theory of grammar and sentence processing must be able to treat the language accurately and in depth. Consequently, this book will go into considerable linguistic detail. In reading it, the linguistically naive reader will be at a disadvantage. He may be somewhat overwhelmed by the barrage of linguistic facts and thus have difficulty evaluating the processing advantages which accrue from casting the modern ideas about language into the framework given here. For this reason it seems worthwhile to attempt an overly simplified model of the linguistic insights and computational strategy which form the heart of CTE. With this model in mind, the reader can see in the chapters to follow how it is modified to account for the additional complexities actually faced, and how it is in fact implemented in terms of an extremely simple basic structure. Any grammarians who feel this model is too simple to account for full English grammar should look at the references to get an idea of the kind of material which has been incorporated in the full treatment to follow.

The CTE design is the result of careful system modularization and careful choice of data representations. Modes of representation typically used to represent linguistic facts are the augmented transition network (ATN), the parenthesized expression, the semantic network, and the procedure. Facts expressed in different modes of representation may be mathematically equivalent yet lead to different levels of complexity in an implemented system. In CTE, all of these modes of representation are used; each has its role.

## 1.2 ATN Networks

Consider the following word strings

1.1. John kicked the bucket.
*1.2. John kicked bucket the.

The first is a grammatical sentence in English; the second is not. (Strings judged ungrammatical are preceded by "*".) The trouble with 2 is that in English the determiner (the, a, etc) must precede the noun. In CTE we distinguish a finite set of syntactic categories like noun, verb, determiner, and proper noun. Every word (or, more precisely, every sense of a word) is assigned to a syntactic category. The CTE parser recognizes legal word orders by tracing through ATN's with these categories on the arcs. For example, Figure 1. 2 shows an ATN for a noun phrase.

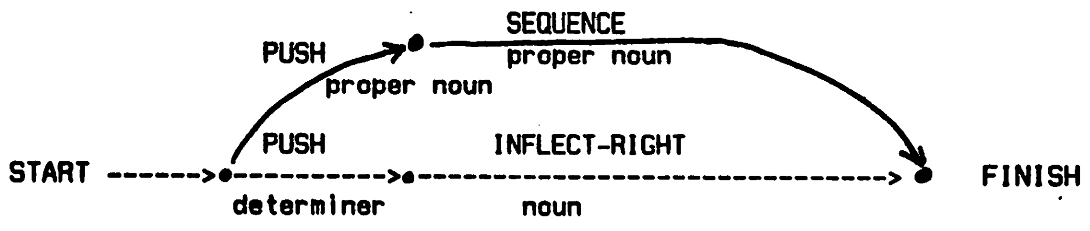

Figure 1.2
Simplified ATN for a Noun Phrase

The ATN is a set of nodes connected by directed arcs. Underneath the arc is written the syntactic category of a constituent required to make the transition represented by that arc. Above the arc is written the name of a function which must be successfully executed in order to make the transition. A push-down stack, a marker on the sentence, and a marker on the ATN are further used to effect the transition. The ATN is easiest to explain by giving an example of its use in parsing the noun phrase <u>the bucket</u> during the parsing of sentence 1. Refer to Figure 1.3.

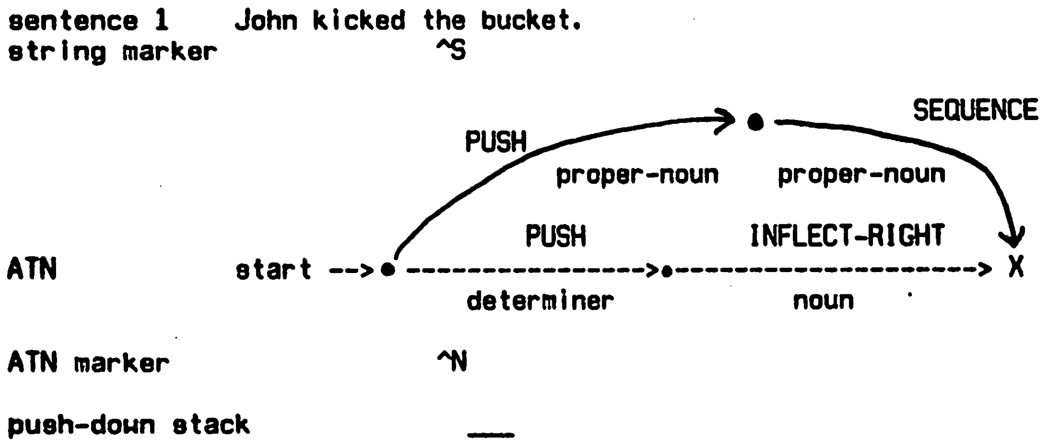

Figure 1.3

When the CTE parser encounters the word <u>the</u> during the analysis of sentence 1, the parser will find the syntactic category of <u>the</u>, which is <u>determiner</u>. It does this by looking in the CTE <u>world model</u>, which is a semantic net. The parser then looks in the world model to see what if any ATN a determiner could start. (A given syntactic category can start at most one ATN in CTE.) The parser finds that a determiner can start a noun phrase ATN. As shown schematically in Figure 1.3 the parser then prepares to try to recognize a noun phrase starting with the word <u>the</u>. It places a string marker, ^S, at <u>the</u>, and a corresponding ATN marker, ^N, at the starting state of the noun phrase ATN. It also establishes an empty <u>push-down stack</u>.

The parser is now ready to attempt the first transition. The starting node of the noun phrase ATN has two arcs leading out of it, but since the is not a proper noun the first arc is not applicable. To make the transition on the second arc the parser must successfully execute the function PUSH. This particular function is always successful, it just places a copy of the constituent pointed to by ^S onto the push-down stack. When the PUSH function succeeds, the parser advances the ^S and ^N pointers giving the configuration shown in Figure 1.4.

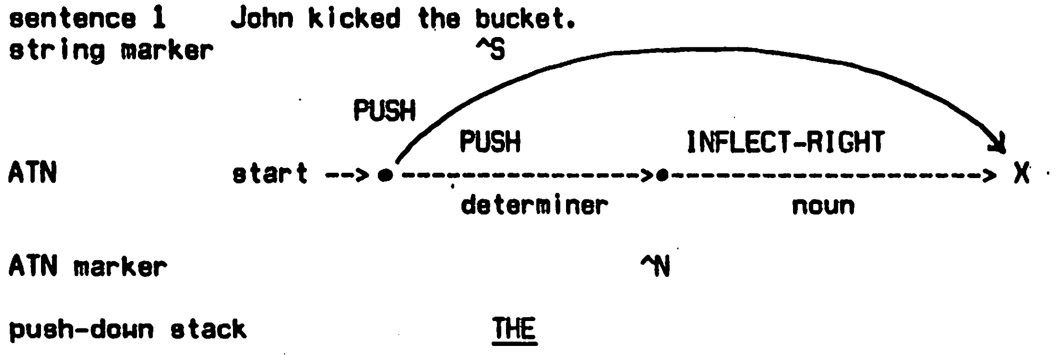

Figure 1.4

The parser proceeds to attempt another transition. As shown in the Figure, the next arc requires a noun. In the world model, the parser finds that bucket is a noun so the transition will succeed if the function INFLECT-RIGHT can be executed successfully. The explanation of this function requires understanding of the CTE semantic net.

## 1.3 The CTE Semantic Net

The nodes of the CTE semantic net are called concepts. In what follows, a mapping will be established from English affixes, words, and phrases into concepts. Concepts are written as parenthesized expressions. That is, each node of the CTE semantic net is an expression! We will thus establish a mapping from English affixes, words, and phrases into expressions. For example, the concept corresponding to the bucket can be written as the expression, (BUCKET*X THE) - the notation for concepts will be explained momentarily.

Associated with each concept is a set of slots. For example, the concept KICK corresponding to the verb kick would have, among others, a subject slot and an object slot. The subject is the thing which does the kicking and the object is the thing which gets kicked. Associated with each slot is a description of how it can be filled. A slot of a concept is also a concept and so to describe a slot we must describe a concept. Any concept, and thus a slot, can be described by one or more predicates, and by one or more characterizations. Predicates and characterizations are also concepts. A predicate gives a property of something, while a characterization gives an alternative way of looking at something. The distinction is seen in saying "a toilet is white" vs. saying "a toilet is a siphon". Describing it as a siphon we can go on to set up a correspondence between the parts of a toilet and the corresponding parts of a siphon. Similarly, to say that someone is male is merely to assign him the masculine property. To say that he is a male, however, is to provide an alternative characterization of him, with the implication that his behavior can be predicted by making a correspondence between his behavior and that of a male. In the example at hand, we could characterize the subject slot of kick as a person or an animal, or we could just give this slot the predicate animate. This is shown schematically in Figure 1.5, (ignore the *A's for the moment).

```
KICK   Characterizations:  None
       Predicates:  None
       Slots:  (SUBJECT*A KICK)    Characterizations:  None
                                   Predicates:  ANIMATE
                                   Slots:  None
               (OBJECT*A KICK)     Characterizations:  MATTER
                                   Predicates:  None
                                   Slots:  None
```

Figure 1.5
A Fragment of the World Model for KICK

Every concept has a reference list. This reference list is similar in function to the property lists used in programming languages like LISP. The process of placing a concept, A, which represents a predicate, characterization, or slot of a concept, B, on B's reference list is called attachment. A reference list has separate sections for predicates, characterizations, and slots, and thus whether A is a predicate, characterization, or slot of B is determined by what section of B's reference list it is on.

The description of concepts by predicates, characterizations, and slots which are themselves concepts is what makes the CTE world model a semantic net. The fact that concepts are written as expressions, so that predicates, characterizations, and slots are in fact assigned to expressions, is not a common sort of thing and distinguishes the CTE from other theories. When combined with the notions to follow, this turns out to be a powerful method of describing how language is at the same time both idiomatic and productive of new constructions.

Examples of concepts written as parenthesized expressions have been given, i.e. (BUCKET*X THE), (SUBJECT*A KICK), (OBJECT*A KICK), but the notation used has not yet been explained. The expression for any concept, C, is constructed from two other concepts termed the genus and the specializer of C. The general form of the expression is

(genus*meta-attribute-abbreviation specializer).

There are seven possible meta-attributes as shown in Figure 1.6.

| <u>meta-attribute</u> | <u>abbreviation</u> | <u>example of use</u> |
|---|---|---|
| SPECIES | S | bull dog → (DOG*S BULL) |
| STEREOTYPE | T | lap dog → (DOG*T LAP) |
| INSTANCE | I | Fido → (DOG*I FIDO) |
| ASPECT | A | subject of kick → (SUBJECT*A KICK) |
| RESTRICTION | R | fat dog → (DOG*R FAT) |
| INFLECTION | X | the bucket → (BUCKET*X THE) |
| PARTITIVE | P | flock of sheep → (FLOCK*P SHEEP) |

Figure 1.6

By definition, all references to a given concept denote the same node in the CTE semantic net. For example, the specializer of (SUBJECT*A KICK) or (OBJECT*A KICK) is the same KICK concept which has these two concepts as slots. The genus, meta-attribute, and specializer uniquely identify a concept and thus serve to locate it in the semantic net.

By convention, every concept inherits the predicates, characterizations, and slots of its genus, whenever those are not contradicted by description on the concept itself. Herein lies the origin of the term genus. Because of this convention, it makes sense to think of the concepts as being organized in a hierarchy. <u>Concepts are pictured as being "under" their genus.</u> Further, one refers to (DOG*S BULL) as a species of its genus, DOG, (DOG*T LAP) as a stereotype of DOG, etc. More generally, any concept is termed a <u>specialization</u> of its genus.

The primary role of the specializer of a concept, C, is to distinguish C from all other concepts having the same genus and meta-attribute; thence the term specializer. For example, BULL distinguishes (DOG*S BULL) from all other species of DOG. LAP distinguishes (DOG*T LAP) from all other stereotypes of DOG.

In review, note that while both the genus and specializer must be used to distinguish a concept from others and thus to locate it in the semantic net, it is the genus which primarily determines the "behavior" and use of a concept since predicates, characterizations, and slots are inherited from the genus.

We return now to the recognition of the bucket as a noun phrase. At this point the reader may find it helpful to review what has been said thus far.

## 1.4 Should Syntax be Represented in the ATN or the World Model

Having pushed THE onto the push-down list and recognized bucket as a noun, the parser attempts to execute the function INFLECT-RIGHT to complete the transition to the final state of the noun phrase. To succeed, INFLECT-RIGHT must succeed in inflecting the concept BUCKET corresponding to the word, bucket, by the concept on the top of the push-down stack, THE. We say "succeed in inflecting" because INFLECT-RIGHT will only form this inflection if it is permitted by the rules of grammar encoded in the semantic net. That is, the ATN gives only the legal orders of syntactic categories and which adjacent constituents can be combined. Further constraints on what can be combined come from the semantic net.

To clarify what is going on here, consider the strings in Figure 1.7.

| | |
|---|---|
| 1.3 | the bucket |
| *1.4 | bucket the |
| 1.5 | a bucket |
| *1.6 | a water |

Figure 1.7

Given that the and a are determiners and bucket and water are nouns, the noun phrase ATN will rule out 1.4 because there is no series of arcs corresponding to the ordering of syntactic categories: noun, determiner. The noun phrase ATN will permit 1.3, 1.5 and 1.6 since they all have the order: determiner, noun, provided that the given determiner can inflect the given noun. The rules for what determiners can inflect what nouns are encoded in the semantic net. These rules permit 1.3 and 1.5, but prohibit 1.6.

By refining the syntactic category noun to mass noun and count noun and then further dividing count nouns into singular and plural count nouns we could replace the noun phrase ATN with the one shown in Figure 1.8

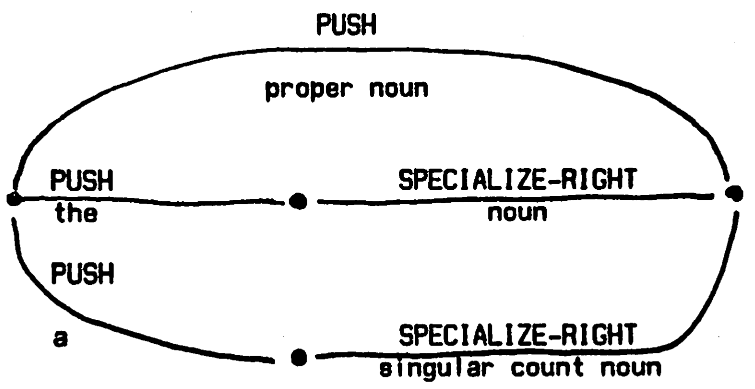

Figure 1.8

The advantage of doing this would be that since water is not a singular count noun, the ungrammatical combination a water would be sifted out by the legal syntactic categories on the arcs and would never be passed to INFLECT-RIGHT. The disadvantage is an increase in the size and complexity of the ATN. It is a question of where these constraints should be expressed, in the ATN arcs, in the ATN arc functions, or in the semantic net.

Bucket the is unacceptable because the word order is incorrect. Since the ATN is the only formalism in CTE capable of describing word order, bucket the must be ruled out by the ATN. A water is incorrect because the normal sense of water is a mass noun, and a requires a singular count noun. There is in fact a productive mechanism by which we can create a count noun sense of water. For example, if offered a choice from glasses of water and coke, one could conceivably answer I'll take a water. Allowable word senses do not involve word order and need not be expressed in the ATN.

It takes less computation to rule out a combination by failure to belong to a syntactic category than it does to rule it out by failure of a function on an arc. This would seem to be an argument for elaborating the ATN. However, when one considers how infrequently combinations like a water occur he sees that in terms of average system performance little is gained by ruling them out more efficiently.

In CTE we have chosen to make the syntactic categories on the ATN arcs as generic as possible. This greatly simplifies the ATN's, but, obviously, it places a greater burden on the semantic net and the functions on the arcs to rule out ungrammatical combinations. Several simpler ATN systems have been implemented (Brown, Saccerdoti) which allow more articulated categories on the arcs. These have the advantage in construction that one does not have to discover the appropriate generic syntactic categories. However, if these systems grow larger the price in complexity will become obvious. Conceivably, one can alleviate this problem by inventing an ATN "compiler" which will automatically discover the appropriate syntactic generalizations.

## 1.5 Filling Slots

Writing the CTE ATN functions and semantic net rules turns out to be relatively straight-forward in most cases. The key is to utilize the hierarchical structure of the semantic net and the inheritance of slots.

Nouns and determiners, for example, are arranged in the hierarchies exemplified in microcosm in Figure 1.9. These hierarchies, incidently, show us how the parser can determine if the concept corresponding to a particular sense of a word is in a given syntactic category.

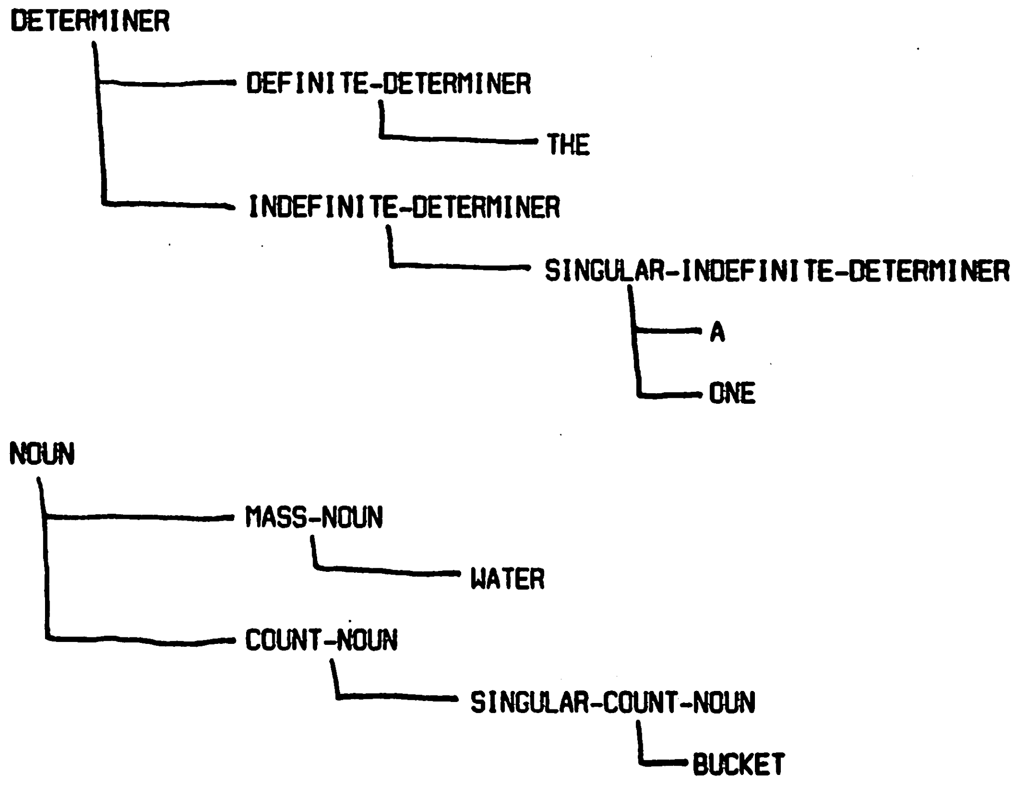

Figure 1.9

Since a concept is formed from its genus, meta-attribute, and specializer; whenever we have located a concept, we can easily obtain these three components. It is quite simple, then, to trace through the semantic net from a concept to its genus, and on to the genus of that genus, etc. That is, to go "up" the semantic net toward more generic concepts. A concept, B, is defined to be in the <u>class</u> of a concept, C, if C is eventually reached by tracing up from B as mentioned above. Since locating the genus of a concept takes only a couple of machine instructions in the current implementation of CTE, the test for class membership is very fast.

To indicate which determiners can inflect which nouns, each determiner is given a slot called the INFLECTEE. For example, consider the world model fragment shown in Figure 1.10.

```
THE
    Characterizations:
    Features:
    Slots:                  (INFLECTEE*A THE)
                                     Characterizations:   NOUN
                                     Features:
                                     Slots:
SINGULAR-INDEFINITE-DETERMINER
    Characterizations:
    Features:
    Slots:                  (INFLECTEE*A SINGULAR-INDEFINITE-DETERMINER)
                                     Characterizations:   SINGULAR-COUNT-NOUN
                                     Features:
                                     Slots:
```

Figure 1.10

The following rule of grammar can then be stated:

G1) A concept, C, can inflect a concept, B, if B is a member of a class formed by a characterization of the INFLECTEE slot of C.

For example, THE can inflect BUCKET because BUCKET is a member of the class formed by NOUN (as shown in Figure 1.9) and NOUN is a characterization of the INFLECTEE slot of THE (as shown in Figure 1.10). Similarily, A can inflect BUCKET because BUCKET is a member of the class SINGULAR-COUNT-NOUN and since A inherits its INFLECTEE slot from SINGULAR-INDEFINITE-DETERMINER, the INFLECTEE slot of A is characterized by SINGULAR-COUNT-NOUN.

When given THE and BUCKET as arguments, INFLECT-RIGHT

a) First checks to see if the concept (BUCKET*X THE) is already present in the semantic net. If so, it replaces the top element, THE of the push-down stack with this concept, (BUCKET*X THE). INFLECT-RIGHT then notifies the parser it has succeeded.

b) Otherwise, it checks the semantic net to see if the formation of (BUCKET*X THE) is permitted under the rule of grammar, G1, given above. (This rule grammar is implicit in the operation of the INFLECT-RIGHT procedure, i.e. it is represented procedurally.) INFLECT-RIGHT locates the INFLECTEE slot of THE, sees that this is characterized by NOUN, and then verifys that BUCKET is in the class formed by NOUN. (BUCKET*X THE) being permitted, it is formed, added to the semantic net, and placed on the push-down stack as in a). INFLECT-RIGHT then notifies the parser it has succeeded.

c) Failing a) and b), INFLECT-RIGHT notifies the parser it has failed.

The implications of the above procedure are quite far-reaching. The effect of step b) is to cause every grammatical expression actually encountered to be permanently remembered as a new concept in the semantic net. As such, it can be recognized on subsequent encounters in step a) from its genus, meta-attribute, and specializer. The potentially expensive test to see whether one concept can fill a slot of another is done only on the first encounter. This seems an attractive option in a world of exponentially declining computer memory prices and relatively constant computation speeds. The retention of absolutely everything is not essential to CTE, but the distinction between grammatically permitted and actually existing concepts is considered quite important. Once remembered, a new concept immediately inherits descriptions from the concepts from which it was formed, but it also can begin to acquire its own set of predicates, characterizations, and slots, which override or supplement those inherited. In particular, as we shall see once the parser reaches the phrase <u>kicked the bucket</u>, it can stand for an idiomatic meaning.

Once INFLECT-RIGHT has placed (BUCKET*X THE) on the push-down stack in place of THE, the parser can take the transition to the final state of the noun phrase ATN. The parse of the noun phrase is complete. In CTE, the ATN's are so written that the concept corresponding to the string recognized by an ATN is always the only concept on the stack when the final state of the ATN is reached.

## 1.6 Inflections

In English, the determiner and the noun must agree in number. A determiner, like the which is unspecified as to number can go with either a singular or plural noun, the bucket, the buckets. A determiner specified as to number requires agreement, *these bucket. Where the noun has the same singular and plural form, the correct sense must be chosen, these sheep. As we have seen, this constraint is implemented in CTE, by appropriate characterization of the INFLECTEE slot of the determiner. Formulations other than CTE frequently employ special feature checking functions - a source of additional complexity.

Other implementations also copy features from the noun and determiner to the expression created to represent the entire noun phrase. For example, since one says the buckets are, but the bucket is, it is necessary to achieve number agreement between the verb and the subject noun phrase. Commonly, the number is copied to the noun phrase and is then fetched from there for comparison with the verb. In CTE no copying is done. Instead, attributes such as number are inherited by the concept representing a noun phrase according to the general rules for inheritance of attributes in the semantic net. This has two advantages. First, a reduction in complexity is achieved because no copying functions need be written. Second, the complexity of the new data structure created when a new sentence is parsed is reduced to a bare minimum. The concept representing the new sentence is constructed from existing ones using specialization. No other constructive operations need be done in the semantic net.

Recall that a concept inherits the predicates of its genus. The concept (BUCKET*X THE) would thus inherit the number predicate of BUCKET as required. To avoid copying, it is also necessary for (BUCKET*X THE) to inherit predicates, such as DEFINITE, from THE. The solution adopted in CTE is to allow an inflection to inherit properties from both its genus and specializer. If an inflection, C, has genus, A, and specializer, B, then a predicate, characterization, or slot on C overrides those inherited from A and B and a predicate, characterization, or slot on B overrides those inherited from A. This convention has proven to be extremely useful. The inflection is one of the key computational devices in CTE.

Language is known to be open ended. Whenever something new is found, a new word can be invented to name it. When one examines the syntactic categories one by one, however, he sees that they are not all open in the same sense. New determiners are invented very slowly in comparison with nouns. In fact, only nouns, verbs, and adjectives are truly open. These three categories contain words for concepts which may be defined extra-linguistically. Obvious examples are words for emotions, color or taste. While the language may determine which concepts are formed in these categories, it is not the sole source of their definitions.

Other categories, like adverbs, do have infinitely many possible members. However, CTE assumes that word senses in these categories can be divided into those native to the category, like <u>not</u> and <u>very</u> in the case of adverbs, and words which arise through conversion from another category, as the adverb <u>slowly</u> comes from the adjective <u>slow</u>. CTE assumes that the native portion of such a category is finite and, in fact, can be usefully structured using language specific principles. For example, English provides features by which determiners can be classified in a tree as shown in microcosm in Figure 1.9.

CTE assumes that the finite categories and the finite portions of the open categories are provided by the language for the interpretation of the information in the truly open categories. The features used to structure members of a finite category provide a definition of a member in terms of its distinction from other members. An understanding of these features is key to understanding the language.

To review, CTE assumes a finite set of syntactic categories. Only three of these syntactic categories - noun, verb, and adjective - have a potentially infinite amount of material native to them. The remaining categories have a rather small finite number of word senses all of which can be included in a computer program for computational linguistics.

Any category besides noun, verb, and adjective which has a potentially infinite amount of material gets it by conversion (slow → slowly) or other forms of affixation (fry → fryable) from other infinite categories.

In CTE we assume that conversion is always done by inflection. The specializer is in the category converted to - the genus in the category converted from. Therefore, to find the syntactic category of a concept, one traces up the genus except in the case of an inflection, for which one traces up the specializer. For example, if we make -LY an adverb then (SLOW*X -LY) is in the semantic class of SLOW, but in the syntactic category, <u>adverb</u>. Compare

1.3. He walks slowly.
1.4. His walking is slow.

In both sentences, an adjective, slow, is used to describe a verb, walk. In 3 the adjective has been converted to an adverb. In 4 the verb has been converted to a noun. The language requires that an adjective modify a noun and an adverb modify a verb. In constructing the parsed sentences these conventions are obeyed by the syntactic categories of the concepts involved. However, the semantic classes of the concepts remain the same in both examples.

It is with these thoughts in mind that determiners are taken to inflect nouns.

## 1.7 Naming

To parse John kicked the bucket the CTE parser must trace through the sentence ATN shown in Figure 1.11.

```
                                                  ONE-OBJECT
  Start        PUSH            PUSH               MAKE-TENSE    INFLECT-LEFT
--------->-------------->--------------->--------------->-------------->
          (noun*x          (verb*x          (noun*x                      .
             determiner)      -ed-s-null)      determiner)
```

Figure 1.11
A Simplified Sentence ATN

The third arc of this ATN requires the sequential execution of two functions - ONE-OBJECT then MAKE-TENSE - something which didn't arise in the noun phrase ATN. The steps in tracing through this ATN for John kicked the bucket are shown schematically in Figure 1.12.

```
sentence       John kicked the bucket.
string-marker  ^S
stack          ____

sentence       John kicked the bucket.
string-marker      ^S
stack          (JOHN*X NULL-DETERMINER)

sentence       John kicked the bucket.
string-marker              ^S
               (KICK*X -ED)
               (JOHN*X NULL-DETERMINER)

sentence       John kicked the bucket.
string-marker                      ^S
stack after    ((KICK*X (BUCKET*X THE))*X -ED)
ONE-OBJECT      (JOHN*X NULL-DETERMINER)

stack after    (((KICK*T (BUCKET*X THE))*X PAST-TENSE)*T
MAKE-TENSE      ((MALE*T (NAME*S JOHN))*X NULL-DETERMINER))
```

```
sentence       John kicked the bucket.
string-marker                        ^S
stack          ((((KICK*T (BUCKET*X THE))*X PAST-TENSE)*T
                 ((MALE*T (NAME*S JOHN))*X NULL-DETERMINER))*X DECLARATION)
```

Figure 1.12
Steps in parsing John kicked the bucket

When the parser is called to parse a string of words into a sentence it prepares to trace through the sentence ATN; it puts the string marker ^S at the beginning of the sentence and sets up an empty pushdown stack for use in tracking through the sentence ATN.

The first arc of the sentence ATN is labeled with (noun*x determiner). By convention in CTE, this can be matched only by a determiner inflecting a noun. The first word in the string, John, corresponds to the concept JOHN, which is a proper noun, not a determiner. The parser is faced with the dilemma of matching a proper noun against (noun*x determiner). There is also a further difficulty. John has two word senses, shown in

1. John kicked the bucket.
1.5. John is a common name.

A male can kick a bucket, a name can't. When the parser is looking at the first word of 1 or 5, it can't know which is the correct sense for that sentence.

There are two basic strategies which a parser can follow in dealing with multiple word senses. The first is to try each sense in turn - returning multiple parses when more than one sense results in a correct parse. This strategy was used in early systems because of its simplicity, and was found to be computationally explosive. Most words have many senses.

The second strategy is known as the wait and see strategy. At any decision point, the possible alternatives are lumped into classes according to how they affect the decision at hand. The computation is brought forward for each such distinct class, rather than for each alternative. To implement the wait and see strategy each class must be represented and a way of getting from the class to the individual alternatives provided.

In terms of the problem at hand, there is no need to split the senses of a word any farther than required by the arc transition attempted. Since both word senses of <u>John</u> are nouns, both, if determined, would satisfy the first arc of the sentence ATN. Suppose we let JOHN stand for both of these senses, which we write (NAME*S JOHN) and (MALE*T (NAME*S JOHN)). We enter into the world model that JOHN <u>names</u> (NAME*S JOHN) and (MALE*T (NAME*S JOHN)). Naming is represented schematically with an arrow, as shown in Figure 1.13

```
JOHN     →     (NAME*S JOHN)
JOHN     →     (MALE*T (NAME*S JOHN))
```

Figure 1.13

The first arc of the sentence ATN requires a noun inflected by a determiner. It is easy, however, to come up with sentences where the subject of a sentence is a noun without a determiner.

1. John kicked the bucket.
1.6. Water is required for life.

It is common in this case for grammarians to speak of a null determiner. Since the absence of a determiner can be detected syntactically just as well as any specific determiner can, no determiner can stand as a determiner.

In CTE such null elements are explicitly inserted to obtain a uniformity of representation. The insertion of the null determiner is done by naming. For example suppose

```
JOHN  →  (JOHN*X NULL-DETERMINER)
```

is placed in the world model. When the CTE parser finds that it cannot match JOHN to (noun*x determiner) it looks for all the concepts named by JOHN. It finds (NAME*S JOHN), (MALE*T (NAME*S JOHN)), and (JOHN*X NULL-DETERMINER). Using the computationally fast syntactic category test, it eliminates all but (JOHN*S NULL-DETERMINER) as being in the wrong syntactic category. Since John is a noun, this last one matches.

The mechanism employed here is a very general and important one in CTE. A concept is taken to name another concept which is in a different syntactic category. In order to achieve a match, the parser will trace down naming links looking for concepts of the syntactic category required by the pattern to be matched. Thus, the replacement of names by the concepts they name is driven by the matching process - that is, by the immediate context of the name.

The wait and see strategy is implemented in the grammar and parser by requiring that the parser never replace a name with a concept it names if the name itself will match the pattern. That is, naming links are used only when a match cannot be otherwise achieved.

Note that two slightly different uses of naming have been introduced. In the first, JOHN stands for (NAME*S JOHN) and (MALE*T (NAME*S JOHN)), two concepts in the same syntactic category as JOHN but with particular meanings lacked by JOHN. In the second JOHN stands for (JOHN*X NULL-DETERMINER), a concept in a different syntactic category - the syntactic category controlling the interpretation of the meaning of JOHN.

If it were in fact necessary to write a naming rule of the form <u>proper name → (proper-name*x null-determiner)</u> for every proper name, the naming mechanism would be too unwieldly to implement. It is necessary to define naming <u>productively</u>. In CTE, if any concept A names an inflection of itself, (A*X B) then any concept C in the syntactic class of A is inferred to name (C*X B). For example, instead of placing JOHN → (JOHN*X NULL-DETERMINER) in the world model it is sufficient to use PROPER-NAME → (PROPER-NAME*X NULL-DETERMINER). Since JOHN is a PROPER-NAME, the fact that JOHN names (JOHN*X NULL-DETERMINER) is then inferred by the parser from (PROPER-NAME*X NULL-DETERMINER).

Having completed the transition on the first arc of the sentence ATN, the parser tries to match the pattern on the second arc, (verb*x -ed-s-null), with a concept corresponding to kicked. In the world model the parser finds that the suffix -ED may be productively applied to verbs. (Of course, there are exceptions for small classes of verbs which override this general rule.) The parser thus matches (KICK*X -ED) against (verb*x -ed-s-null).

The third transition faces the parser with matching (noun*x determiner) again. The string marker is pointing at the, which does not match this pattern; nor does it name anything which matches this pattern. But, as was already explained, the parser recognizes that the can start a noun phrase. In the current implementation of CTE the semantic net contains the information that a noun phrase can indeed match the pattern (noun*x determiner). This has been included so that the parser can avoid building a noun phrase if it could not possible match the pattern at hand.

The parser builds the noun phrase (BUCKET*X THE). To do this it sets up a separate string pointer and push-down-stack for the noun phrase. During the formation of the noun phrase the sentence string pointer and push-down-stack remain unaltered. To complete the transition it must execute the functions ONE-OBJECT and MAKE-TENSE. The function ONE-OBJECT expects the top element on the stack to have an OBJECT slot. Its goal is to fill the OBJECT slot of this top element with the noun phrase starting at the current location of the string marker. In the case at hand this means filling the OBJECT slot of (KICK*X -ED) with (BUCKET*X THE). (KICK*X -ED) inherits its OBJECT slot from KICK. As shown in Figure 1.5. The OBJECT slot of KICK has been characterized as MATTER. To fill this slot with (BUCKET*X THE), the function ONE-OBJECT must be able to characterize (BUCKET*X THE) as MATTER. Assuming that BUCKET is in the class MATTER, this is easy, but in the actual CTE implementation such a characterization could require some rather expensive pattern matching.

Recall that in CTE, inflections are used to implement the syntactic constraints of a language as illustrated by

3. He walks slowly.
4. His walking is slow.

| | Subject | Predicate |
|---|---|---|
| 3. | (WALK*X PRESENT-TENSE) | (SLOW*X ADVERB) |
| 4. | (WALK*X NOUN) | SLOW |

In 3 the subject is in the syntactic category TENSE and the predicate in the syntactic category ADVERB. In 4 the subject is in the syntactic category NOUN and the predicate is in the syntactic category ADJECTIVE. While the syntactic categories of 3 and 4 differ, the semantic classes do not. When filling slots other than the INFLECTEE the parser works with the semantic classes. The INFLECTEE slot is filled based on syntactic categories.

Having discovered that (BUCKET*X THE) can fill the OBJECT slot of (KICK*X -ED), ONE-OBJECT forms the expression corresponding to <u>kicked the bucket</u>. The convention followed here is to stereotype KICK with (BUCKET*X THE), forming (KICK*T (BUCKET*X THE)) and then to inflect this with -ED, forming ((KICK*T (BUCKET*X THE))*X -ED).

At the semantic level, (KICK*T BUCKET) is viewed as a stereotypical kind of KICK about which information not applicable to KICK in general may be known. For example, if cows kick the bucket, the milk is spilt. (KICK*T BUCKET) may have different slots than KICK. When combined with KICK, the concept (BUCKET*X THE) performs two distinct functions. First, it selects a specialization of KICK, (KICK*T (BUCKET*X THE)). In CTE, a convention has been implemented which insures that if a concept C is in class B, then the concept (A*meta-attribute C) is in class (A*meta-attribute B). This convention is called derivative subclassification, because concepts with the same genus are thus classified into the hierarchy of their specializers. In our example, derivative subclassification means that since (BUCKET*X THE) is in the class BUCKET, (KICK*T (BUCKET*X THE)) is in the class of (KICK*T BUCKET) and thus inherits any slots of (KICK*T BUCKET) which override those of KICK.

Besides selecting a specialization of KICK, (BUCKET*X THE) also fills the OBJECT slot of the selected specialization. The fact that BUCKET fills the OBJECT slot of (KICK*T (BUCKET*X THE)) can be determined from the expression (KICK*T (BUCKET*X THE)) because in CTE the slots of a verb are constrained to stereotype it in a specific order starting with the OBJECT.

The use of concepts to fill slots and simultaneously select specializations is a very important aspect of CTE. The importance of selecting specializations of a verb is seen clearly when we compare the difference in meaning of, for example, shoot pool, shoot rapids, shoot picture, shoot gun, and shoot rabbit.

Having replaced the top item on the push-down stack with ((KICK*T (BUCKET*X THE))*X -ED) the parser begins execution of MAKE-TENSE, the second function which must be executed to make the transition on the third arc of the sentence ATN.

A TENSE has two slots, a SUBJECT and an INFLECTEE. MAKE-TENSE tries to insure that the item on the top of the stack is of syntactic type, TENSE.

It does this by attempting to match the item on the top of the stack to (verb*x tense). To match (KICK*X -ED) to (verb*x tense), the parser must use a third type of naming.

The world model contains a naming rule of the form

(VERB*X -ED) → (VERB*X PAST-TENSE)

This type of naming rule is also productive on the genus. The parser can therefore infer that (KICK*X -ED) names (KICK*X PAST-TENSE) and make the transition. The SUBJECT slot of the resulting specialization is then filled with the next to top item on the push-down stack. Obviously, the item filling the SUBJECT of the TENSE must also satisfy the SUBJECT slot of the verb kick. As shown in Figure 1.12, this means that JOHN must be replaced with (MALE*T (NAME*S JOHN)) because a PROPER-NOUN cannot kick a bucket. The mechanism by which this is insured will be explained momentarily in describing the passive transformation.

The final step in parsing John kicked the bucket is to inflect the entire expression with DECLARATION to distinguish it from the question John kicked the bucket?

## 1.8 The Passive Transformation

A basic tenet of transformational grammar is that sentence pairs like 1 and 7 are transformationally related.

1. John kicked the bucket.
1.7. The bucket was kicked by John.

In CTE, this relationship is seen as one between the slots of the predicates kicked and was kicked. The sentence 7 satisfies the same set of four ATN arc transitions in the sentence ATN as, for example, 8. Namely, noun phrase, tensed be, adjective, prepositional phrase.

1.8. John was sure of success.

In both cases, the functions on the arcs must determine if the subject noun phrase can fill the subject slot of the adjective and if the prepositional phrase can fill a slot of the adjective. The transformational insight is that the requirements for filling the subject slot of 7 are those for filling the object slot of 1; and the requirements for whether the prepositional phrase of 7 can modify the adjective are those for filling the subject slot of 1. This is expressed by the notion of slot shift. Specifically, the SECOND-PARTICIPLE is taken to form a syntactic category which for simplicity can be considered a subcategory of adjective. A world model fragment describing the second participle is shown in Figure 1.14. This Figure shows that the INFLECTEE of a second participle must be a verb and the SUBJECT of a second participle must be the OBJECT of its INFLECTEE. Further, this OBJECT of the INFLECTEE has the predicate SHIFTED, which means that it will not occur as the direct object of the verb, as it normally would.

```
SECOND-PARTICIPLE:
    predicates:
    characterizations:
    slots:              (INFLECTEE*A SECOND PARTICIPLE)
                             predicates:
                             characterizations:      VERB
                             slots:
                        (SUBJECT*A SECOND-PARTICIPLE)
                             features:
                             characterizations:  (OBJECT*A (INFLECTEE*A
                                                      SECOND-PARTICIPLE))
                                                   predicates:  SHIFTED
                                                   characterizations:
                                                   slots:
                             slots:
```

Figure 1.14

In parsing 7, the parser will find (KICK*X -ED) which it converts to (KICK*X SECOND-PARTICIPLE) in order to make the third transition of the sentence ATN. Then, when the parser is looking for the SUBJECT of (KICK*X SECOND-PARTICIPLE) the rules of inflections state that slots inherited from SECOND-PARTICIPLE override those inherited from KICK. Thus the SUBJECT of (KICK*X SECOND-PARTICIPLE) is found from the SUBJECT of SECOND-PARTICIPLE. From Figure 1.14, this is the OBJECT of the INFLECTEE. The INFLECTEE in this example being KICK. Thus the SUBJECT of (KICK*X SECOND-PARTICIPLE) is the OBJECT of KICK.

The filling of a slot with <u>by John</u> is handled in a similar manner. This is spelled out in detail in the next chapter.

## 1.9 <u>Idioms</u>

<u>John kicked the bucket</u> is an idiomatic way of saying <u>John died</u>. The syntactic scope of this idiom is demonstrated by

1. John kicked the bucket.
1.9. John has kicked the bucket.
1.10. When will John kick the bucket?
1.11. John's kicking the bucket upset us.
1.12. John is kicking the bucket.
*1.13. John is kicked the bucket.
*1.14. John kicked a bucket.
*1.15. The bucket was kicked by John.

Adding to the world model the naming transformation

```
(KICK*T (BUCKET*X THE))  →
        (DIE*S (KICK*T (BUCKET*X THE)))
```

permits 1 to 12. Sentence 13 is disallowed by the ATN network which does not permit the formation of ((KICK*T (BUCKET*X THE))*X -ED) after forms of <u>be</u>. 14 is disallowed because it uses <u>a bucket</u> and the naming rule uses <u>the bucket</u>. 15 is ruled out because under the passive transformation (BUCKET*X THE) is no longer used to specialize KICK as object. A more sophisticated treatment of this idiom would have to include the fact that the progressive 12 is marginal, as is the progressive reading of 11.

Under the wait and see strategy this idiomatic naming transformation would not be taken unless dictated by context - a subject beyond the scope of this overview.

## 1.10 Other Transformations

As Chomsky has recently observed, "noun phrase movement" as exhibited by the passive transformation accounts for many of the transformations in traditional transformational grammar. However, there are three other types of transformations, WH- movement, extraposition, and minor movement. Examples of these are

<u>WH- movement</u>
1.16. John kicked <u>what</u>?
1.17. <u>What</u> did John kick?

<u>Extraposition</u>
1.18. <u>That you are here</u> is great.
1.19. It is great <u>that you are here</u>.

<u>Minor Movement</u>
1.20. I picked the bucket <u>up</u>.
1.21. I picked <u>up</u> the bucket.

Extraposition and minor movement transformations are coded into the ATN. WH- movement requires special procedures to be written into the parser. To see exactly how these things are done the reader will have to read the remaining chapters. We have reached a level of sophistication which requires us to go back and develop the ideas already presented more fully and precisely.