PARSING FRENCH VERBS USING AUGMENTED TRANSITION NETWORKS

By

TIMOTHY R. DUGAN

Bachelor of Science

Oklahoma State University

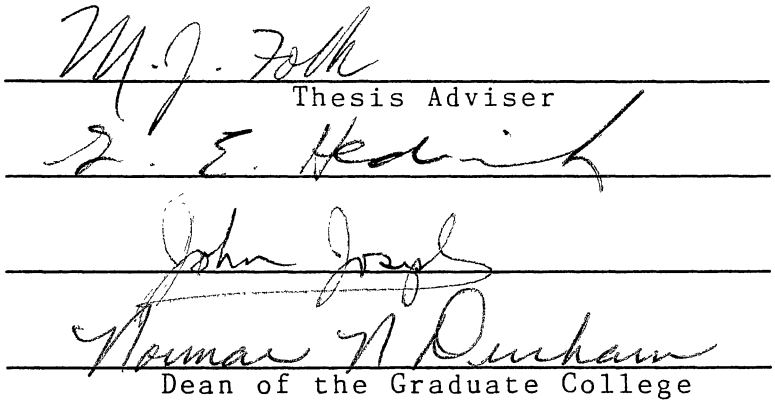
Stillwater, Oklahoma

1983

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1985

PARSING FRENCH VERBS USING AUGMENTED TRANSITION NETWORKS

Thesis Approved:



Thesis Adviser

Dean of the Graduate College

PREFACE

I would like to express my gratitude to my major adviser, Dr. Mike Folk, and to the other committee members, Dr. G. E. Hedrick and Dr. John Joseph. I would also like to thank Bill Zoellick for the use of his word processor and my fellow employees of TMS, Inc. for their support. Also, my family and friends deserve special consideration, especially Litsa, who is very dear to me.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Motivation

The two prime reasons for research in the area of natural language processing are to enhance the usefulness of computers and to broaden our understanding of human language. The augmented transition network (ATN) is a model for natural language grammars. The motive for studying the ATN is to explore the utilization of attributes of words and phrases in directing a parser using this type of grammar. French verb constructs provide a target grammar that is sufficiently complex to illustrate many of the facets of this problem while not requiring an exhaustive grammar for a language.

Research Strategy and Scope

The methodology used in this research was to develop a model and program it. The realization of a model as a computer program provides a test of practicality and compels a certain amount of rigor in the form of source code which is mathematically descriptive.

The strategy used here in conducting research has been to study an isolated phenomenon within language. This strategy is useful in testing specialized theories of restricted lan-

guage phenomena. For example, the project described here involves the analysis of the syntatic structure of sentences and phrases.

The scope of this paper is limited to issues related to parsing with a subclass of ATN's known as augmented transition trees (ATT's), to the use of attributes of phrase constituents to guide the parsing process, and to the representation of French verb syntax in the form of these trees. A top-down, left-to-right parser is presented which accepts a phrase as input and produces all acceptable parses as output. Each acceptable parse is a tree structure that is representative of the form of a phrase called the deep structure.

Outline

A certain amount of background knowledge is necessary to appreciate this project and to make intelligent criticisms regarding it. Chapter II contains a description of relevant general linguistic issues. Chapter III contains a description of specific syntax issues in French. Chapter IV provides a description of augmented transition networks and an examination of some literature dealing them. Chapter V contains a more detailed definition for an augmented transition tree grammar and a description of an accompanying parser. Sample parses of French phrases are shown and explained. After this project description, a summary will be provided in Chapter VI and conclusions will be drawn. Appendices in-

clude the LISP source of the parser and a sample French grammar, an accompanying dictionary.

CHAPTER II

LINGUISTIC ISSUES

Introduction

This chapter contains an overview of linguistic issues related to the project described in Chapter V. Issues are developed from a historical perspective in order to build from earlier theories. This description is not intended to be exhaustive, but merely to give sufficient background information to allow the reader to understand the project presented later. Chapter III will apply many of these concepts to the syntax of French verbs.

Language

Any system that deals with natural language should take certain properties of language into accoun. (Bolton: 1981). Firstly, language is productive; a speaker can produce new sentences at any time. For example, many of the sentences in this paper have never occurred before. Secondly, language is arbitrary, the phonological representation of a word does not necessarily have a direct relationship with its meaning. The onomatopoeic word "cock-a-doodle-doo" is descriptive of the sound it names, but there is no such relationship between the word "dog" and the same animal. A third quality is

duality, a given sound or sequence of sounds can have more than one meaning. The word "fast" exhibits this property in the sentences "He is fast." and "He stood fast." The keys to distinguishing the correct interpretation are the context in which the word or phrase occurs and the structure of the sentence.

Structural Linguistics

Traditionally, words have been categorized as "parts of speech" which were originally derived from Latin grammar. The category of a word indicates what it can represent. For example, in English a noun is defined to be a person, place, or thing. Pragmatically, the system of parts of speech is not as descriptive as it might seem. A noun can be used as an adjective, as in "cat food" while the sentence "Seeing is believing." illustrates how words normally thought of as verbs can be used in the same way that nouns are. To be able to analyze sentences and phrases as a whole, the structure relating the words needs to be understood. As one author phrased it, "Structural grammar focuses on clusters of structures--sounds, forms, word groups, phrases--working from smaller to larger units." (Heatherington, 1981, 336).

One developement of structural linguistics described by Heatherington was the use of a technique known as immediate-constituent analysis. Sentences are viewed as being constucted from groups of words that are paired together to form new groups, rather than from a series of single words.

An example is the sentence "Sentences are formed from constituents." which can be broken down into constituents as follows:

"[Sentences] [[are formed] [from constituents]]."

As a result of the immediate consituent analysis, the structuralists developed basic syntactic patterns called sentence formulas For example, the pattern

Noun/Pronoun + Verb + Noun/Pronoun

can be used to generate a sentence such as

Constituents form sentences.

Still, this system does not account for the fact that this sentence, at a deeper level, is very similar to the previous sentence, which is written in a passive voice.

Transformational-Generative Grammars

The structural model described above was an improvement on the traditional model, but, as indicated, it still did not account for many problems in language analysis. In 1957, Noam Chomsky published a book called Syntactic Structures in which he proposed a new model called Transformational-Generative Grammar. This new model utilized some of the ideas of the traditional and structural grammars but extended existing theories to show how sentences are produced. Over the years, Chomsky's theories have evolved, and, at present, many points of his theories are hotly debated.

One suggestion supported by Chomsky was that structures of constituents could be generated by a set of rules called

phrase structure (PS) rules. A phrase structure rule consists of the name of a phrase element and a sequence of constituents that can be produced from it. For example, the phrase structure rules of Figure 1 can produce the sentence "John has gone."

```
(1)  S   -> NP  VP
(2a) NP  -> N
(2b) NP  -> PRO
(3a) VP  -> V
(3b) VP  -> AUX V
```

Figure 1. Phrase Structure Rules

A sentence is generated by starting with a phrase consisting only of the sentence symbol S. Symbols in the phrase that occur on the left-hand side of PS rules are repeatedly replaced by strings of symbols that occur on corresponding right-hand sides. Using rule (1), the symbol S produces a noun part NP and a verb part VP:

S => NP VP.

In turn, NP produces a noun, N, by rule (2a):

=> N VP.

If we decide that "John" is the noun that interests us, the sentence becomes

=> John VP.

Choosing rule (3b), we arrive at

=> John AUX V.

If we decide that the auxiliary is "has" and the verb is "gone", we end up with

=> John has gone.

This generative process can be represented by a tree in Figure 2.

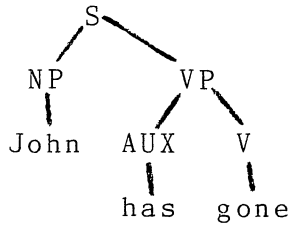


Figure 2. A phrase Structure Tree

The type grammar shown above is far from complete as a description of how language is generated. For instance, one might wish to form a question from the above sentence: "Has John gone?" This sentence cannot be derived directly from the PS rules, as there are no rules that would allow an auxiliary to come before the subject. One could add a whole new set of rules to generate the interogative sentences, but it would be useful to take advantage of the similarities of questions to affirmative sentences. A better solution begins by adding a few new rules to the grammar in order to add a new constituent to the phrase determining whether it is af-

firmative or interrogative. The new PS rules are shown in figure 3.

```
(0)  P      -> Voice S
(1)  S      -> NP  VP
(2a) NP     -> N
(2b) NP     -> PRO
(3a) VP     -> V
(3b) VP     -> AUX V
(4a) Voice  -> Affirm
(4b) Voice  -> Interog
```

Figure 3. Extended PS Rules

The final phrase generated be determined by the PS rules, a transformational rule, and phonological rules. Figure 4 shows a simple diagram of the relationship between rules in the transformational model.

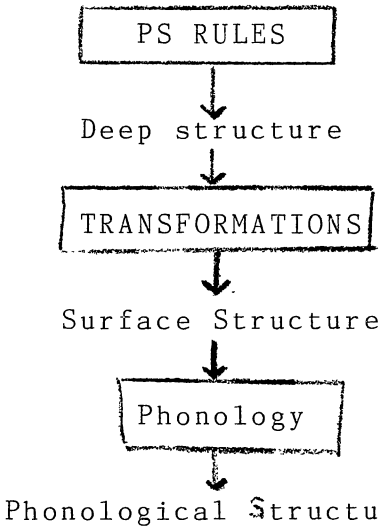


Figure 4. The Transformational Cycle

Now, after making the start symbol be P instead of S, a new form of the sentence can be generated:

P => Voice S => ... => Voice John has gone.

If we let Voice be Affirm, the sentence becomes

Affirm John has gone.

and if Voice is Interog, it becomes

Interog John has gone.

The phrase structure generated by the PS rules shall be refered to as the deep structure. The deep structure of a phrase is a representation formed from the information that will occur within a phrase. It is fashioned according to the PS rules. In addition to new phrase structure rules, a transformational rule that modifies the surface structure will allow the grammar to explain the similarity of

questions and statements. The affirmative sentence undergoes no change in structure. Because the Voice constituent does not have words or sounds associated with it, it will only be apparent because of intonation. On the other hand, the interogative sentence might undergo a transformation described by a transformational rule, such as

Interog N AUX V -> Interog AUX N V.
1 2 3 4 1 3 2 4

A transformational rule changes a structure by adding, deleting, or rearranging constituents. This rule indicates that the auxiliary should move in front of the noun to form the phrase

Interog Has John gone.

which, when spoken, becomes the phonological form to which the speaker is accustomed:

Has John gone?

It is important to know that some transformations are obligatory and others are not. For instance, the interogative nature of the phrase above could just as easily be made clear by intonation:

John has gone?

Verbs and Case

In some instances, the possibility of having a certain constituent as part of a phrase depends on properties of other parts of the phrase. These properties can be described as being context sensitive attributes. One such instance

involves the syntactic uses of noun phrases associated with a verb.

One can define the surface case of a noun group as its position within the surface structure of a phrase. For example, the subject of a phrase has the nominative case, and the object of a transitive verb has the accusative case. We can also define a deep case in a corresponding way within a deep structure. The two are not necessarilly the same because a transformation can cause a constituent to change its case, as is illustrated by the phrases

(A) She kissed him.

and

(A') He was kissed by her.

Both sentences have the same deep structure, except that (A') has the passive voice. "She" performs the action described by the verb, and thus has the nominative deep case. "He" is the object of the kiss and is assigned the accusative deep case. But, due to the transformation, "He" in sentence (A') becomes the subject and gains the nominative case in the surface structure.

Sentence (A) would no longer seem correct if we substitute the word "looked" in place of the verb:

*She looked him.

The verb "to look" cannot have an accusative object. We can distinguish this difference between "to look" and "to kiss" by saying that "to kiss" has an accusative attribute and "to look" does not. However, one can say

(B) She looked <u>for</u> him.

because the verb "to look" has an attribute for a prepositional phrase of the form "for <someone or something>". The attributes define what forms of objects and prepositions a verb can have in the deep structure, which may or may not translate to other cases in the surface structure:

(B') He was looked for by her.

Note that the cases of noun phrases that can be associated with a given verb are arbitrary and are only meaningful in conjunction with the semantics of the verb. Substituting "to seek" for the verb in sentence (B) illustrates this fact:

(C) She sought him.

The object of the verb is of a different form, but the meaning remains essentially the same.

Morphology

Morphological analysis makes possible a generalized knowledge of base words from which other forms can be construct- A morpheme is defined as the minimal linguistic unit which has meaning or grammatical function. (<u>Language Files</u>: 1981). Some examples of morphemes include words such as "dog" and "grow" which are called free morphemes because they can stand alone. Functional morphemes are words such as prepositions because they are free but do not convey meaning except in combination with other words. The English suffixes "-s" and "-ly" are called bound morphemes because they are meaningless unless affixed to a word. Bound morphemes that do

not form a new meaning but only change a quality of a word are known as inflections. Examples include a plural marker "-s" and the past participle marker "-ed". Affixes that change the meaning or function of a word are called derivational. For instance, the adjective "poor" combined with the suffix "-ly" derives the adverb "poorly."

CHAPTER III

FRENCH SYNTAX

Introduction

This chapter describes French syntax as it pertains to the project described in Chapter V. The important points described here include the conjugation of verbs in the indicative mood, the use of modals, types of noun groups that can be associated with a given verb, and a simple transformational model of French sentences. These grammar details will be of interest in devising a network grammar for French verb constructs. The network grammar will be useful in determining the deep structure of a sentence.

Conjugation of Verbs

Conjugations are variations from the base form of the verb that indicate such attributes as tense, number, and person. In this paper, the infinitive is used as a base reference form for verbs, as is customary in dictionaries. There are eleven tenses of the indicative mood in French. Five are simple tenses:

- Present
- Imperfect
- Simple Past
- Future
- Conditional

Three types of conjugations occur in the simple tenses--regular, stem changing, and irregular. Different verbs may require different kinds of conjugation for the same attributes, though usually large numbers of verbs have similar conjugations. The regular conjugation of a verb consists of changing the inflection of the base form to one that indicates the tense, number, and person:

parler + -ez -> parlez

The base form of the verb (parler) combines with the inflectional ending having the attributes indicating present tense, second person, and plural number (-ez). The result is a verb phrase having the same attributes. Some verbs undergo a stem change in certain tenses while still taking an inflection that denotes tense, number, and person:

aller + -ez(future) -> ir- + -ez -> irez.

Many verbs have a third type of conjugation which is irregular. The conjugated verb has a unique form for each combination of person and number without any regular pattern of inflections.

In addition to the five simple tenses, there are five compound tenses. They are

- Present perfect
- Pluperfect
- Passed anterior
- Future perfect
- Conditional past

Compound tenses are composed of an auxiliary and the verb. The auxiliary is conjugated as one of the five simple tenses and the verb has a past participle inflection. The tense is

that of the auxiliary with an added sense of completeness. For example, the present becomes the present perfect and the imperfect becomes the pluperfect. Most verbs require "avoir" as an auxiliary, but a few use "être". A verb that is conjugated with "être" as the auxiliary must agree with the subject in gender and number. One such verb "descendre" has the past particple "descendu", but in a sentence with the feminine subject "elle" (she) the participle gains the feminine ending "-e":

"elle est descendue." (She descended.)

Note that some of these rules are only apparent in spelling. Like English, the orthography of French was established long ago while the pronunciation has continued to change. The written form of the language is analyzed in this paper, but the reader should keep in mind that many of the same or similar rules apply to the spoken form.

One final tense consists of the present perfect of the auxiliary and the past participle of a verb, e.g. "j'ai eu parlé." This tense is interpreted much the same as the past perfect.

Modals

French makes use of a few verbs that can be classified as modals. A modal can affect the meaning of a sentence in two ways: aspect and modality. The aspectual modals affect the interpretation of the tense of the verb. The modal verb "aller", conjugated as "vais", indicates near future in the

phrase "je vais parler." (I am going to speak.) The second type of modal affects the mode of the sentence. In the phrase "je peux parler", the modal "pouvoir", conjugated as "peux", indicates the ability to do something rather than the actual act. When a modal occurs, it carries the conjugation while the main verb becomes an infinitive. An infinitive verb is not marked with person, number, or gender, but it can still have noun groups associated with it. If an infinitive is marked with the past tense, the appropriate auxiliary is included in the infinitive form and the main verb appears as a past participle.

Objects and Pronouns

Verbs can have various different types of noun groups associated with them. They can have direct objects, as in the phrase "porter le chapeau" meaning "to wear the hat". Direct objects are assigned the accusative case. They can have indirect objects, as in "avoir parlé à quelqu'un" which means "to have spoken to someone." Indirect objects are given the dative case. A third type of object is illustrated by the phrase "parler de quelqu'un" -- "to speak about someone". The genative case is assigned in this instance. These three cases are all that will be considered in this paper.

When a noun is replaced by a pronoun, it often changes position in the phrase. For instance, since "porter le chapeau" means "to wear the hat", "le porter" means "to wear

it" and "lui avoir parlé" means "to have spoken to him or her". In the case of a compound tense with a pronoun in the accusative case, the participle must agree in gender and number with the pronoun. In the phrase "les avoir portés" (to have carried them) the past participle ends with the plural marker "-s" in order to agree with the plural object "les".

Transformational Grammar

So far, this chapter has presented various aspects of French grammar without much regard for the structure of the phrase as a whole. The relationship between the constituents is depicted by this structure which can be described by a transformational grammar. It will be assumed that the PS rules derived from Dubois (Dubois and Dubois, 1970) and shown in Figure 5 generate the deep structures of the subset of French that is of interest for the project.

(1) S -> SN SV
(2) SV -> Aux GV
(3) GV -> V (Acc) (Gen) (Dat)
(4) Aux -> Tps (Parf) (M) (Parf)
(5) Tps -> (future) { pres | Past } Per Num
(6) Past -> imperfect | simple
(6) Parf -> { avoir | etre } PP
(7) M -> { Mod | Asp } Inf

Figure 5. Some French PS Rules

The symbol S stands for a valid deep structure of an affirmative sentence. SN is the subject of the phrase. SV indicates the verb phrase. Aux is an auxiliary and GV contains the main verb. Tps determines the tense, number, and person. Parf indicates a sense of completeness (perfect). M designates a modal. Past can be either imperfect or simple past. PP indicates the past participle affix and Inf indicates an affix for the infinitive mood and movement rules cause them to combine with a verb. The parentheses indicate an optional element while the curly brackets allow a choice of items separated by the vertical bar. These rules contain sufficient information to produce all the tenses that are of interest, but do not indicate any transformations.

The process of generating the phrase "Elle a lu le livre." (she has read the book.) can be illustrated with these rules. The process begins with the symbol S:

S => SN SV.

Applying rule (2) we derive

=> SN AUX GV.

Using rule (3) and choosing the optional Parf, the phrase becomes

=> SN Tps Parf GV.

We will assume that the main verb has "avoir" as an auxiliary. Thus, rule (6) gives

=> SN Tps avoir PP GV.

The constituent GV expands into the verb and object:

=> SN Tps avoir PP V ACC.

At this point, the tense, number, and person come into play:

=> SN present Per No avoir PP V ACC.

The parts of the constituent Tps form an affix for "avoir" which translates to the conjugation of "avoir":

=> SN a PP V ACC.

The subject SN is "elle", the verb V is "lire":

=> elle a PP lire ACC.

The past participle affix PP combines with the verb "lire" producing the form "lu":

=> elle a lu ACC.

The accusative object is "le livre" (the book), which gives the final form:

=> elle a lu le livre.

The surface structure can be represented by a tree, as shown in Figure 6.

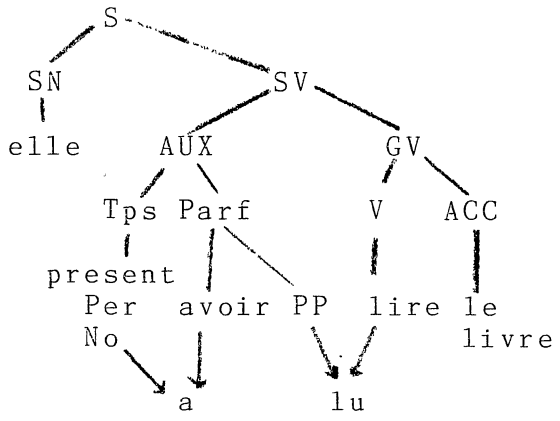


Figure 6. Phrase Structure of a French Sentence

If the sentence is going to have the pronoun "le" instead of the noun object "le livre", an additional transformation rule is necessary that forces movement of the accusative constituent. The rule might appear as:

Auxiliary Verb Pro => Pro Auxiliary Verb
1 2 3 3 1 2

The resulting phrase would be "elle l'a lu." (she has read it.)

CHAPTER IV

PARSING AND NETWORK GRAMMARS

Parsing

In order for computers to react in a meaningful way to natural human language, an algorithmic process is necessary that can deduce the underlying structure of a phrase. We will assume that a phrase P in a language is generated by a grammar G that describes the language and a structure S that encodes the meaning of the phrase:

P <= Generate(G,S).

Parsing is an inverse process to rediscover the structure S:

S <= Parse(G,P).

A parser takes as input a string of symbols such as the phrase P and outputs any valid structural representation of that string.

A model of how a language is generated should provide a basis for developing a parser. The generative-transformational model of natural language describes a phrase as being generated from a series of phrase structure rules and transformation rules. An ideal method would be to apply the transformational rules in reverse to the surface structure, thus reversing the generation process until the initial structure is found. This process is known as transforma-

tional parsing

Difficult problems occur in trying to find this reverse process (King, 1983). One such problem is that transformation rules often involve the deletion of a phrase constituent but the transformation rule itself may not give any indication of what the deleted value is.

An additional problem is that it is necessary to find the surface structure before the reverse transformations can be applied; another process would be necessary to do this. King describes a system called MITRE that does succeed in a limited domain using transformational parsing.

Augmented Transition Networks

In 1970, Woods published a paper titled "Transition Network Grammars for Natural Language Analysis." (Woods, 1970) In this paper, he described transition network grammars and their use in parsing phrases. A transition network, also known as a finite state automaton, can be described as a directed graph. The graph consists of arcs connecting vertices to one another. Each vertex represents a state. The arcs represent transitions from one state to another. Each arc from a given vertex of the graph has a label. This label is used to match some input symbol, such as a word, and change the current state. One vertex of the graph is a starting state. A path beginning with the starting state is traced through the network wherever arc labels can be matched with input symbols. Whenever the path reaches a vertex

marked as an acceptance state, a correct parse has been found. The path found is representational of the phrase being analyzed. Figure 3 illustrates a transition network.

Finite state automata are not computationally powerful enough to handle the complexities of natural language. If the symbols that label the arcs of the network are optionally allowed to represent any transition network, the network becomes a recursive transition network. Woods showed that recursive transition networks have the computational power of push-down automata. A push-down automaton can recognize exactly that set of formal languages representable by context-free grammars.

Context-free grammars provide a framework for the structure of natural language but lack the ability to represent context-sensitive attributes such as subject-verb agreement. They also lack the ability to specify arbitrary structural rearrangements. In order to strengthen this model, it needs to be augmented with arbitrary tests and structure-building actions in conjunction with named registers used to store partial results. The resulting construct is known as an augmented transition network (ATN). Woods demonstrated that if the structure building actions have the power of a Turing machine, then so will the ATN. Having such computational power indicates that the ATN model can be used in conjunction with any linguistic model of language, though its basic format is closest to those that make use of some form of phrase structure rules.

Publications that discuss aspects of ATN's include descriptions of numerous data base front ends, the use of ATN's in describing languages not yet well understood (Grimes, 1975), methods for developing interpreters, compilers, and editors for ATN's (Bole, 1983), and analysis of strengths, weaknesses, and extensions to the ATN formalism (King, 1983).

Winograd, in his book Language as a Cognitive Process (Winograd, 1983), gives detailed analysis of various parsing methods and provides an example ATN grammar of English. The networks in Winnograd's model are converted to procedures in a computer programming language and then compiled into machine code for fast, efficient analysis. Thus, they are called procedural grammars. The parsing procedes as a top-down, depth-first process. Because the complexity of natural language does not usually allow deterministic parsing, these processes must be capable of backtracking. Additional accounting must be done to find the right points in the networks to return to after trying an unsuccessful path.

The transition network should reflect the surface structure of the phrase. If the desired result of the parse is to find a deep structure, at various times in the parsing process it may be necessary to rearange structural components such as in a reverse transformations. These actions are accomplished by the structure building component of the grammar.

In the network grammar of English phrases used as an il-

lustration in Winograd's book, the networks only roughly reflect the structure of the phrase being parsed. The structure-building rules are used to arrange and rearrange register values. As a result, the structure-building actions bear much of work for the parsing process. The effect of placing too much of the logistics within the structure-building rules and not enough within the network diagram can make a grammar less comprehendable and less modifiable. Experience seems to indicate that large ATN models can become complex and thus difficult to extend (Johnson: 1983). This problem seems to indicate some degree of discipline is needed when specifying a network grammar. Some possibilities include restricting the forms that networks can take and limiting the types of tests and actions that can be accomplished by the grammar.

Augmented Transition Trees

Methods for interpreting and compiling ATN's are described in the first edition of the book LISP (Winston and Hart,1981), but in the second edition (Winston and Hart, 1984), Winston shifts the focus of the discussion to augmented transition trees (ATT'S). ATT's have been used with some success in processing data base queries (Hendrix et al., 1978).

An ATT is very similar to an ATN but with the added restriction that the network be limited to a subclass of directed graphs known as a tree. This restriction means that

no loops can occur within a transition network and that a given vertex of the graph can be reached from only one other vertex. Transition trees can be derived directly from a set of phrase structure rules that define the surface structure. Each symbol on the left-hand side of a phrase structure rule is the name of a transition tree. If more than one rule has the same left-hand symbol, the equivalent information as these rules is incorporated in a transition tree. Figure 7 shows some phrase structure rules and Figure 8 shows the equivalent transition tree.

```
A -> B C D E
A -> B C A
```

Figure 7. Some Phrase Structure Rules

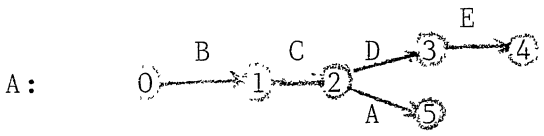


Figure 8. A Transition Tree

The different phrase structure rules share the same path from the root as far as the right-hand sides of the rules

are the same. Acceptance states occur after the rightmost symbol in a phrase structure rule.

Transition trees are used in the project described in the next chapter. This choice was made in order to make constuction of the grammar easier and to put more of the structure building burden on the tree structure itself.

CHAPTER V

PROJECT DESCRIPTION

Introduction

This chapter contains a description of a project to develop a transition network grammar of French verb constructs. The three distinct parts of the project were (1) to devise a network grammar definition, (2) to devise a parser to utilize this type of grammar, and (3) to develop a working grammar for French verb constructs. The parser described here deals with words as the lowest unit. Morphological analysis is performed on each word to discover relevant dictionary entries and the inflections affecting it. The process of morphological analysis, although distinct from the parsing process, is very closely tied to it. The parser then uses the analysis to trace a path through the transition trees, using recursion when necessary, to the final acceptance state, at which point a correct parse is judged to be found. Along the course of the parse, various tests and structure-producing actions are encountered. The tests on attributes of constituents limit the acceptable paths to follow; this allows the grammar to account for context-sensitivity. In addition, the structure building actions along the same path provide a method for making structural

rearangements.

Morphological Analysis and the Dictionary

The purpose of the dictionary here is to enumerate syntactically relevant information about a word. It is not the intent of this system to discover the semantics of a phrase. The category of a word is found in the dictionary along with a series of attributes that distinguish the use of a word within its category. To alleviate the need of having a separate dictionary entry for each inflection of a word, morphological analysis is performed on each word of the phrase being parsed. All possible base forms that are found in the dictionary are produced from this process. Each dictionary entry contains three types of information: (1) the basic category that describes the word, (2) a list of possible inflections on that word, and (3) attributes that describe the syntactic use of the word. Examples of these attributes include case, number, and person. Inflections on the word cause attributes other than those found in the dictionary to be added to the definition.

Because French verbs also have irregular and stem changing forms, special dictionary entries are made to relate these forms to a common root form. Words that fall into more than one category have a dictionary entry for each one, as do words that have different sets of attributes within a category. Some examples include the French word "le" which is both a pronoun and an article and the French verb

"descendre" which is both transitive and intransitive.

A standard dictionary entry has the form:

STD <category> <normal inflection id's>
<stem-changing inflection id's>
<standard attributes>
<default attributes>

Entries for a stem changing verb have the form:

STEM <standard form>

and an entry for an irregular form of a word has the form:

IRREG <standard form> <irregular attributes>

A list of inflections to be used in conjunction with the dictionary has entries of the form:

<inflection> <base form suffix>
<inflection number id>
<inflection attributes>

If a standard dictionary entry is found without inflections, the attributes assigned to the named register are the union of the standard attributes and the default attributes. If an irregular entry is found, the attributes are the union of the standard attributes and the irregular attributes. If the word has a valid inflection, the attributes are the union of the standard attributes and the inflection attributes found in the inflection list.

The Transition Tree Definition

The purpose of this grammar is to describe the surface structure of phrases, specifically, French verb phrases, and to show how it might be converted to a deep-structure representation. In order to construct such a grammar, one needs

to have a formal method for specifying this information in the form of a transition tree grammar. An augmented transition tree is defined by four components: (1) its name, (2) the labeled arcs which define the phrase-structure tree, (3) the conditions upon these arcs, and (4) the structure building rules.

An arc of the tree has a label that corresponds to either the name of a category of words or to the name of a transition tree. After an arc is matched, a register named according to the arc's label is automatically set to indicate information about the structure of the parsed constituent. This information is represented as a list of named attributes, each of which may recursively have attributes. Thus, the attributes of the constituents form trees. If the consituent named by the label of the arc is a word, then the register's value is the set of attributes found in the dictionary.

The structure of an arc is defined as follows:

<start node> <label> <destination node> <conditions>

Conditions on an arc are tests that must be satisfied if a path is to be continued past the arc. A condition is a comparison of two values or the inquiry of the existence of an attribute. Conditions may be combined with the boolean operators "and," "or," and "not." The condition may involve a constant or the attributes of any register already set or, recursively, any attribute of an attribute.

This is the form of an arc condition:

(<optional "not"> <condition> <operand> <operand>)

Attributes of a register are referenced by the name of the attribute and the register name:

(<attribute name> <register name>)

and, recursively, deeper level attributes can be referenced:

(<attribute name> <attribute reference>).

Figure 9 shows a typical arc definition. The programming language LISP provides the notational syntax. The arc is a list four elements delimited by parentheses. The conditions are grouped as another list within parentheses.

```
( V1 VERB V2 ( (same (NUMB VERB) (NUMB SUB))
               (exists (TENSE VERB) ) ))
```

Figure 9. A Typical Arc Definition

Registers are allocated dynamically and referenced lexically. The structure building actions utilize registers to store values and make inquiries about them. Because of the way it is allocated, a register set on a certain path will be deactivated when that path is completed causing the register value to disappear. It also means that if several registers with the same name are active, only the most recently activated can be referenced.

The final component of an augmented transition tree is the structure-building component. In this model, the struc-

tural building actions can only occur in an accepting state of the transition tree. The actions can manipulate the values that have been assigned to registers and return the results as attributes of a register with the name the same as the current transition tree. Thus, the parse tree is built recursively, according to the description of the transition tree. When the path of a parse reaches an acceptance state within a given transition tree, the values of the registers set in that tree can be manipulated to provide information to higher levels of the tree and to construct a representation of the structure of the phrase being parsed. The form of an action is as follows:

<list of names> <list of actions to names>
< final condition >

The list of names binds names to the registers set within the current tree. This is necessary to provide access to registers set on the same path that have conflicting names. The actions are evaluated one at a time and the results are assigned to attributes of a register with the same name as the name of the current tree. The final condition is a condition on the newly derived values to insure that the actions are correct.

The Parser

The parser written for the project is implemented in the programming language LISP. It is a top-down, left-to-right

parser and produces all parses in a parallel fashion. The parse begins in a starting state, and for each interpretation of the first word in the phrase, a set of possible next states is found. The union of these sets forms the set of next states for the parser. This process is repeated for each additional word of the phrase and each element of the next-state set until the end of the sentence is reached, at which point all of the valid parses can be determined. Because the parsing process occurs in parallel, each individual parse must have its own stack to indicate the current position within the transition networks and its own set of registers.

A transition tree has two types of arcs: those that match word categories and those that match the name of a transition tree. When an arc that contains a category name is encountered, the dictionary value is assigned to a register. If the arc is labeled with the name of a transition tree, the current status of the parse is pushed onto a stack and current state is set to the root of the new tree. This process is repeated until an arc matching a word category is found. This recursive method of parsing does not allow for the grammar being used to be a left-recursive grammar; if a transition tree has a label on its first arc which is the same as the name of the tree, then an infinite sequence of recursion would occur.

Upon reaching an accepting state within a transition tree, a send action is encountered. The result of a send

action is to evaluate the structure building actions which may use the values of the registers set while searching the current tree. At this point, these registers are deactivated and a new register named according to the label on the tree is created with attributes indicating what actions have been performed. An inquiry is then made of the stack of current states in order to continue searching the previously suspended transition tree.

The parsing process halts when no more paths exists to follow or when the end of the sentence is encountered. A top level transition tree is defined as in Figure 10 which causes the register S to be set that contains the output of an acceptable parse. The symbol $EOS is a special symbol indicating the end of the sentence. Upon completion, the parser outputs a list of all such valid parses.

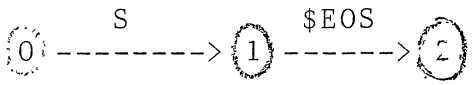


Figure 10. The Top-level Transition Tree

A Simple French Sentence Grammar

The first step in developing a grammar is to decide what elements of the language will be included. The grammar used here has been basically outlined in Chapter III. The major

difference is that elements such as tense are considered as attributes of constituents, rather than separate constituents by themselves. The categories of words used here include the verb, the pronoun, the auxiliary, the modal, and the words "à" and "de". The categories modal and auxiliary are considered different from verbs because of the difference in use syntactically. The verb phrase is the major point of interest, thus the categories used here are considered sufficient to be illustrative.

A network grammar must be descriptive of the surface structure of the phrase being described. The basic framework is a tree representation of each constituent indicating each possible surface structure for of that constituent. The number of permutations that a particular element might have can be very large, so careful planning is necessary to keep the grammar at a minimum size. A detailed grammar is shown in Appendix B.

A number of context sensitive rules are necessary within French grammar. Rules of this type used in the network grammar include subject-verb agreement, the association of direct objects, indirect objects, and genitive objects with verbs, and the agreement of accusative pronouns with a past participle.

The only rules corresponding to reverse transformations are those that move object pronouns from before the verb to after it and those that remove unimportant constituents such as the prepositions "à" and "de" which are replaced by

equivalent syntactic information.

As illustrations, Figures 11, 12, and 13 show resulting parses of French phrases. The notation is the same as that for lists in LISP. Each parse is a tree consisting of a label followed by a list of attributes. Each attributes may be a simple identifier or, recursively, another tree.

Figure 11 shows the verb "descendre" (to descend) conjugated with "etre" as the auxiliary in the sentence "nous sommes descendus." (we have descended.) Figure 12 illustrates a parse of the sentence "je vais lui prler de vous." (I am going to speak to him/her about you.) This sentence shows both a modal and a verb with dative object and a prepositional phrase. Figure 13 shows the structure of the phrase "ils les auront descendues." (They will have taken them down.) In this phrase, the rules of grammar requiring agreement between the accusative object and the past participle.

```
(S (NP (WORD nous) (NUMB plur) (PER 1))
   (VP (V (WORD descendre)
          (NUMB plur)
          (PER 1)
          (TENSE pres_perf)
          (GENDER masc))
       (PER 1)
       (NUMB plur)
       (TENSE pres_perf)))
```

Figure 11. The Sentence "nous sommes descendus."

```
(S (NP (WORD je) (NUMB sing) (PER 1))
   (VP (MODAL (WORD aller)
              (NUMB sing)
              (PER 1)
              (TENSE pres))
       (INFIN_PH (INFIN (WORD parler))
                 (IO (WORD lui) (PER 3) (NUMB sing))
                 (PP (PREP de)
                     (OBJ (WORD vous) (PER 2)
                          (NUMB plur)))
       (PER 1)
       (NUMB sing)))
```

Figure 12. The Sentence "je vais lui parler de vous."

```
(S (NP (WORD ils) (NUMB plur) (PER 3) (GENDER masc))
   (VP (V (WORD descendre)
          (NUMB plur)
          (PER 3)
          (TENSE future_perf)
          (accNUMB plur)
          (accGEND fem))
       (PER 3)
       (NUMB plur)
       (DO (WORD les) (PER 3) (NUMB plur))))
```

Figure 13. The Sentence "ils les auront descendues."

CHAPTER VI

CONCLUSION

Summary

This paper has included discussions of generalities of natural language syntax and specifics of French syntax. Also, known methods of processing natural language with transition networks were briefly examined, and the concept of augmented transition trees was introduced. A more specific design for these trees followed as an illustration of concepts discussed.

Conclusions

In conclusion, it should be reiterated that the closer the transition network models the surface structure, the more readable and modifiable it seems to be. The word categories used in the grammar should be chosen to be descriptive of the syntactic use of the words. Attributes can be assigned to words and phrases and used to provide context sensitive information for a parse. These attributes are also useful in describing the structure of a consitutent. Actions involving constituents and their attributes can be used to achieve reverse transformations helpful in arriving at a deep structure. The resulting information of a parse can

then be interpretted according to semantic rules or possibly be used in translation.

Further Recommendations

Possible directions that are suggested by this project include the design of more detailed network grammars in a fashion that more closely models tried and tested tranformational grammars. Extensions to ideas illustrated here might include using attributes to index nouns and pronouns or as a rating of validity of a consituent of a phrase; this might allow for processing of many phrases that are nearly syntactically correct.

Bibliography

Bole, Leonard. The Design of Interpretors, Compilers, and Editors for Augmented Transition Networks. New York: Springer Verlag. 1983.

Bolton, W. F. "Language: an Introductions." A Living Language: An Introduction. Random House, Inc. 1981.

Clark, Virginia P., Paul A. Eschholz, and Alfred F. Rosa, Editors. Language. New York: St. Martin's Press. 1981.

Dubois, Jean and Francoise Dubois-Charlier. Elements de linguistique francais: syntax. Paris: Larousse. 1970.

Grimes, Joe. Network Grammars. Norman, OK: Summer Institute of Linguistics. 1975.

Heatherington, Madeline E. "The Grammars of English." How Language Works. Cambridge, Mass.: Winthrop Publisheers, Inc. 1980.

Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. "Developing a Natural Language Interface to Complex Data." ACM Transactions on Database Systems. Vol 3. No. 2. pp. 105-147. June 1978.

Johnson, Roderick. "Parsing with Transition Networks." Parsing Natural Language. London: Academic Press. pp. 49-72. 1983.

King, M., Editor. Parsing Natural Language. London: Academic Press. 1983.

King, M. "Transformational Parsing." Parsing Natural Language. London: Academic Press. pp. 19-34. 1983.

Language Files: Materials for an Introduction to Languge. By the instructors of "Introduction to Language," Dept. of Linguistics, Ohio State University. Reynoldsburg, OH: Advocate Publishing Group, 1979.

Winograd, Terry. Language as a Cognitive Process. Reading, Mass.: Addison-Wesley. 1983.

Winston, Patrick and Berthold Hart. LISP. 1st, 2nd Ed. Reading, Mass.: Addison-Wesley. 1981, 1984.

Woods, W. A. "Transition network grammars for natural language analysis." Communications of the ACM. 1970. Vol. 13. pp. 591-606.

APPENDIX A

LISTING OF LISP PARSER CODE

This appendix contains a partial listing of the parser written in Franz LISP. The parser is a top-down parallel parser for augmented transition trees.

```
; sentence.l
;    takes as input a phrase ending in a '.'
;    and passes it a word at a time to set_trans
;    which finds the next set of configurations.
;    Output consists of a list of all valid deep
;    structures for the input sentence.

(defun sentence ()
    (prog (configs sent word)

          (setq configs (list (init_config)))

          (setq sent (getsent))

          loop

          (if sent
              then
              (setq word (pop sent))
              (setq configs (set_trans (disinf word configs))
              (go loop))

          (return (good configs))))

; good returns all configuration with null stacks
;    which means the parse was completed
;
(defun good (configs)
    (prog (config retlist)

          loop

          (if (null configs) (return retlist))

          (setq config (pop configs))

          (if (null (stack_of config))
               (push (tree_of config) retlist))
```

```
          (go loop)))
;
; settrans.l
;    performs a transition mapping for each
;    pair of elements from two sets
;
(defun set_trans ( A B )
     (prog (retlist A2)

     (setq A2 A)

     loop
          (if (null B) (return retlist))
          (setq retlist (union
                    (transition (car A2) (car B)) retlist))
          (setq A2 (cdr A2))

          (if (null A2) then (setq B (cdr B))(setq A2 A))

          (go loop)))

;
; trans.l
;         finds the transitions from a word-configuration
;         pair that are legal state changes according to
;         the transition tree.
;
(defun transition (word config)
     (prog (tree stack status node new_tree
          ret_list arcs arc next_stack)

          (setq tree (tree_of config))
          (setq stack (stack_of config))
          (setq status (pop stack))

;    check if we are looking at a new transition graph
;
          (if (null (cdr status))
           then
;      is it a non-terminal symbol?
                (if (nu (cat_of status))
                 then
;      is it a member of the 'first' set of the next state
                    (if member (cat_of word)
                                (first (cat_of status)))
                     then
                         (setq node (begin (cat_of status)))
                     else
                         (return nil))
                 else ; does our word match the path?
                    (if (equal (cat_of word)(cat_of status))
                     then
                         (setq new_tree (cons word tree))
                         (setq ret_list (resolve stack
```

```
                                                        new_tree)))
          else
               (setq node (node status)))

;      find all the arcs from this node.
     (setq arcs (getarcs node))

;      loop through the arcs, putting each next state
;      back on the stack for later processing.

     loop
     (if arcs
      then
          (setq arc (pop arcs))
          (setq next_stack stack)

          (push (list (cat_of status)
                      (node_of arc)
                      (test_of arc))
                next_stack)

;    recurse to find lower elements of the tree.
          (setq ret_list (append (transition word
                                             (list next_stack tree))
                                 ret_list))

          (go loop)
     )

     (return ret_list)))
;
; resolve.1    resolves send arcs and backs up
;              levels of the stack.
;

(defun resolve (stack tree)
     (prog (status tests new_stack
            new_tree ret_list send)

          (if (null stack) (return (list (list nil tree))))

          (setq status (pop stack))
          (setq tests (test_of status))

; if test on arc fails, return empty set
          (if (not (test tree tests)) (return nil))

; if a non-terminating node, set up path to next node.
          (if (getarcs (node_of status))
           then
               (setq new_stack stack))
               (push (list (cat_of status)
                           (node_of status)) new_stack)
               (push (list new_stack tree) ret_list)
```

```
        )

        (if (setq send (getsend (node_of status)))
         then
             (if (setq new_tree (action tree
                                        send
                                        (cat_of status)))
                 (setq ret_list (append (resolve stack
                                                 new_tree)
                                        ret_list)))

(return ret_list)))
```

APPENDIX B

SAMPLE GRAMMAR

This appendix contains a partial listing of the sample grammar constructed for simple French sentence. Each tree of the grammar is reprsented by a name and a list of arc definitions which were described in the text.

```
($start ( 0 S         1 nil)
        ( 1 $EOS      2 nil)
        ( 2 *SEND (s $) (s) nil))

(S      ( 0 NP        1 nil)
        ( 1 VP        2 ((agree (NUMB VP) (NUMB NP)) etc.))
        ( 2 *SEND (np vp) (sn sv '(Affirm.) '(Indic.)))

(VP     ( 0 V         1 nil)    ; a simple verb.
        ( 1 *SEND (v) (v (PER v) etc.

          .             .             .
          .             .             .
          .             .             .

        ( 0 ACC       20 nil)
        (20 V         21 ((exists (ACC V))); asks if verb is
        (21 *SEND (do v) (v do (PER v) (NUMB v) etc.) nil)

        (21 PP        22 (( ... does verb take prep? ..)))

          .             .             .
          .             .             .
          .             .             .
```

The section of the grammar shown above should give a general idea of how the grammar branches, makes tests on conditions, and how it relates to phrase structure rules.

VITA 2

Timothy R. Dugan

Candidate for the Degree of

Master of Science

Thesis: PARSING FRENCH VERBS USING AUGMENTED TRANSITION NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Sapulpa, Oklahoma, November 9, 1960, son of Earl and Shirley Dugan.

Education: Graduated from Kiefer High School, Kiefer, Oklahoma in May 1979; received Bachelor of Science degree in Computer Science from Oklahoma State University in May, 1983; completed requirements for Master of Science degree at Oklahoma State University in May, 1985.

Professional Experience: Teaching Assistant, Department of Mathematics, Oklahoma State University, August, 1983, to May, 1984. Programmer for TMS, Inc., May, 1984 to present.