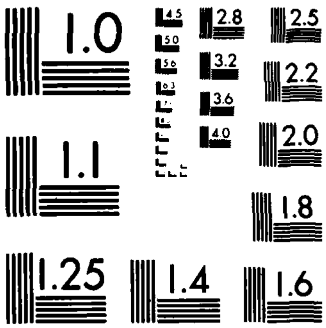
AD-A131 537 A THREE-STEP PROCEDURE FOR LANGUAGE GENERATION(U) 1/1
MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB B KATZ DEC 80 AI-M-599
UNCLASSIFIED N00014-80-C-0505 F/G 5/7 NL

END
FILMED



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AIM 599	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Three-Step Procedure for Language Generation		5. TYPE OF REPORT & PERIOD COVERED Memorandum
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Boris Katz		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0505
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE December 1980
		13. NUMBER OF PAGES 40
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Language Generation
Parsing
Transformations
Natural Language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper outlines a three-step plan for generating English text from any semantic representation by applying a set of syntactic transformations to a collection of kernel sentences. The paper focuses on describing a program which realizes the third step of this plan. Step One separates the given representation into groups and generates from each group a set of kernel sentences. Step Tow must decide, based upon both syntactic and thematic considerations, the set of tranformations that should be performed upon each set of kernels. The output of the first two steps provides the "TASK" for -

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Step Three. Each element of the TASK corresponds to the generation of one English sentence, and in turn may be defined as a triple consisting of: (a) a list of kernel phrase markers; (b) a list of transformations to be performed upon the list of kernels; (c) a "syntactic separator" to separate or connect generated snetences. Step Three takes as input the results of Step One and Step Two. The program which implements Step Three "reads" the TASK, executes the transformations indicated there, combines the altered kernels of each set into a sentence, performs a pronominalization process, and finally produces the appropriate English word string. This approach subdivides a hard problem into three more manageable and relatively independent pieces. It uses linguistically motivated theories at Step Two and Step Three. As implemented so far, Step Three is small and highly efficient. The system is flexible; all the transformations can be applied in any order. The system is general; it can be adapted easily to many domains.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

A.I.Memo No.599 December, 1980

A THREE-STEP PROCEDURE FOR LANGUAGE GENERATION

Boris Katz

ABSTRACT: This paper outlines a three-step plan for generating English text from any semantic representation by applying a set of syntactic transformations to a collection of kernel sentences. The paper focuses on describing a program which realizes the third step of this plan. Step One separates the given representation into groups and generates from each group a set of kernel sentences. Step Two must decide, based upon both syntactic and thematic considerations, the set of transformations that should be performed upon each set of kernels. The output of the first two steps provides the "TASK" for Step Three. Each element of the TASK corresponds to the generation of one English sentence, and in turn may be defined as a triple consisting of: (a) a list of kernel phrase markers; (b) a list of transformations to be performed upon the list of kernels; (c) a "syntactic separator" to separate or connect generated sentences. Step Three takes as input the results of Step One and Step Two. The program which implements Step Three "reads" the TASK, executes the transformations indicated there, combines the altered kernels of each set into a sentence, performs a pronominalization process, and finally produces the appropriate English word string. This approach subdivides a hard problem into three more manageable and relatively independent pieces. It uses linguistically motivated theories at Step Two and Step Three. As implemented so far, Step Three is small and highly efficient. The system is flexible; all the transformations can be applied in any order. The system is general; it can be adapted easily to many domains. Below is an actual example of English text generated by the program from the kernels and transformations of an appropriate input TASK:

"At the beginning of the story Lady Macbeth is bothered by the fact that Macbeth is not the king. Macbeth who was a nobleman is persuaded by her to murder the king with a knife because she wants Macbeth to be the king. She dies and Macbeth becomes unhappy. Macduff didn't see Macbeth murder the king. In the end Macbeth is killed by Macduff who was a good friend of the king. Were there some other stories written by Shakespeare?"

This research was done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505.

§0. Introduction

Suppose that a computer contains a semantic representation of a certain sequence of events or facts. For many reasons, it is useful to generate English text from such a representation. This paper outlines a three-step plan for generating English text from any semantic representation by applying a set of syntactic transformations to a collection of kernel sentences or phrase markers.[1] The paper focuses on describing a program called GEN which realizes the third step of this plan.

Step One of the Language Generation Procedure (LGP) separates the given semantic representation into groups and generates from each group a set of *kernel phrase markers*. A phrase marker can be used in one of several syntactic roles: a matrix clause K_0, an embedded clause K_1, or a relative clause K_2. The role of a given phrase marker must be specified in Step One.

Step Two of the LGP must decide, based upon both syntactic and thematic considerations, the set of transformations that should be performed upon each set of kernels.

The output of the first two steps provides the input for Step Three. We will sometimes refer to this input as the "TASK" for Step Three. Each element of the TASK corresponds to the generation of one English sentence, and in turn may be defined as a triple consisting of:

(a) a list of kernel phrase markers generated by Step One;

(b) a list of transformations, obtained from Step Two, to be performed upon the list of kernels;

(c) a "syntactic separator" (punctuation mark, conjunction, etc.) to separate or connect generated sentences.

Step Three takes as input the results of Step One and Step Two. The program GEN which implements Step Three of the generator "reads" the TASK, executes the transformations specified there, combines the altered kernels of each set into a sentence, performs a pronominalization process, and finally produces the appropriate English word string.[2]

---

1. The definition and the structure of *kernel phrase markers* (Semantic Frame Structure) is described in Section 1.
2. A comprehensive review of previous approaches to language generation can be found in [McDonald 1980].

For example, given the following kernels:[3]

K_0 = a young woman asked it (1)
K_1 = this man ate the cake

the program can apply different sets of transformations for altering the individual sentences or combining them. Among the possible outputs are:[4]

1. After	$0\text{-}NP_1\text{-}TO_1$:	A young woman asked this man to eat the cake.
2. After	QUESTION:	Did a young woman ask this man to eat the cake?
3. After	NOT_1:	Did a young woman ask this man not to eat the cake? (2)
4. After	PASSIVE:	Was this man asked by a young woman not to eat the cake?
5. After	N'T:	Wasn't this man asked by a young woman not to eat the cake?

Suppose that the program GEN takes as input the following TASK:

$$TASK1 = ((K_0\ K_1)\ (\mathit{0\text{-}np_1\text{-}to_1}\ \mathit{Question})\ (?))$$

According to this TASK, the program applies the transformation *0-np*$_1$*-to* to the embedded clause K_1 and the transformation *Question* to the matrix clause K_0. Then the altered kernels are combined and the resulting output is: "Did a young woman ask this man to eat the cake?" which is sentence #2 in (2). If, for instance, we want to obtain sentence #5 from the set (2) above, the input TASK should be:

$$TASK2 = ((K_0\ K_1)\ (\mathit{0\text{-}np_1\text{-}to_1}\ \mathit{Question}\ \mathit{Not_1}\ \mathit{Passive}\ \mathit{N't})\ (?))$$

The last step of the Language Generation Procedure uses three different classes of syntactic operations to "solve" the TASK provided by Step One and Step Two: the system of optional commutative *Transformations*, the set of *Transformational Filters* whose application is obligatory only if certain conditions are met, and the set of intrinsically ordered, obligatory *Adjustments*.

---

3. The word *it* is used here as a *joining point*.

4. For convenience, we append "1" to the name of a transformation which will be applied to an embedded clause, and "2" to a transformation that will be applied to a relative clause. Nothing is appended to the matrix clause. The significance of the names of transformations, such as $0\text{-}NP_1\text{-}TO$, will be explained below in Section 6.

Below is an actual example of English text generated by the program GEN from the kernels and transformations of an appropriate input TASK:[5]

> An old professor wanted a large English class to learn a story about Macbeth. He didn't notice a young man and a young lady eating a huge cake under the desk. They were in danger, weren't they? Fortunately the people who loved the story about Macbeth did not see them either. Hear it now! At the beginning of the story Lady Macbeth is bothered by the fact that Macbeth is not the king. Macbeth who was a nobleman is persuaded by her to murder the king with a knife because she wants Macbeth to be the king. She dies and Macbeth becomes unhappy. Macduff didn't see Macbeth murder the king. In the end Macbeth is killed by Macduff who was a good friend of the king. Were there some other stories written by Shakespeare? [6]

The pronouns found in this example were not present in the input TASK, but were created by the program.[7]

Among the advantages of this approach are:

- it nicely subdivides a hard problem into three more manageable pieces, such that each is relatively independent of the others
- it uses linguistically motivated theories at Step Two and Step Three of the LGP
- as implemented so far, Step Three is small and highly efficient
- the system is flexible; all the transformations can be applied in any order
- the system is very general; it can be adapted easily to many domains.

---

5. The following fragments show the structure of the TASK:

(K_0 = An old professor wanted it; K_1 = A large English class learns a story about Macbeth)	(*0-np$_1$-to$_1$*)	(.)
(K_0 = Macduff saw it; K_1 = Macbeth murdered the king)	(*N't 0-np$_1$-inf$_1$*)	(.)
(K_0 = In the end Macduff kills Macbeth; K_2 = Macduff was a good friend of the king)	(*Relative$_2$ Passive*)	(.)
(K_0 = Shakespeare wrote some other stories)	(*Question Passive There*)	(?)

6. The author apologizes for possible historical inaccuracy in the computer output.

7. The prononimalization procedure is described in the Appendix 1.

§1. Semantic Frame Structure

We will assume that the input semantic representation consists of a set of frames.[8] Step One of the Language Generation Procedure builds from the given representation a set of *kernel phrase markers* using three types of templates (noun-template, verb-template, adverb-template) and two operations (concatenation and conjunction) that assemble them. The structure of the templates is shown below:

noun-template (NT) = (prep* det* adj* noun)

verb-template (VT) = (tense aux1 aux2 aux3 verb)

adverb-template (AT) = (mod adverb)

Here *prep, det, adj, aux, mod* are, respectively, abbreviations for preposition, determiner, adjective, auxiliary and modifier. The superscript * indicates that a string of one or more symbols or their conjunction is allowed. *Tense* also carries the grammatical features of NP_1 -- its number, person, and gender.

Each template is implemented as a list of pairs, an *association list*. The first element of the pair is the name of the constituent, the second is its value. All constituents are optional. If any constituent is absent, its value in the template is *nil*.

Given below are some sample templates:

NT = ((prep nil) (det **a**) (adj **(young pretty)**) (noun **lady**))	"a young pretty lady"
NT = ((prep **with**) (det **(all the)**) (adj **nice**) (noun **books**))	"with all the nice books"
NT = ((prep **(from out of)**) (det **the**) (adj nil) (noun **darkness**))	"from out of the darkness"
VT = ((tense **past**) (aux1 **can**) (aux2 **have**) (aux3 nil) (verb **notice**))	"could have noticed"
AT = ((mod **very**) (adverb **well**))	"very well"

The following two operations are allowed:

8. The term "frame" was introduced in [Minsky 1975]. The Frame Representation Language (FRL) developed in [Roberts and Goldstein 1977] can be used as an example of such representation. In this paper, however, for testing purposes we will consider a set of kernel phrases as the input. A parser has been designed and implemented in order to process the kernel phrases. The description of the parser can be found in the Appendix 2.

1. Concatenation: CONC (X1 X2) ----> X1 X2

2. Conjunction: CONJ (X1 X2) ----> X1 *conj* X2

where X1 and X2 are two templates of the same type, and *conj* is a conjunction such as, *and, or,* etc.

The structure of the frames can be defined in terms of the templates and operations, as follows:

> *Noun-frame* (NF) is any appropriate[9] sequence of applications of the operations CONC and CONJ applied to the noun-template NT.
> *Verb-frame* (VF) and *adverb-frame* (AF) are defined in the same manner, as a sequence of applications of CONC and CONJ applied to VT and AT, respectively.[10]

The noun-frame will be called *prepositionless* if it does not begin with a preposition.

For example:

NF = ((prep nil) (det nil) (adj nil) (noun **Ivan**) (conj **and**)
(prep nil) (det nil) (adj nil) (noun **Maria**)) = "Ivan and Maria".

Each of the prepositionless noun-frames in a sentence has a particular semantic role associated with it: *agent, goal,* or *theme.*

> *Agent* is a thing that causes the action to occur.
> *Goal* is the recipient or the beneficiary of the action.
> *Theme* is a thing that undergoes a state of change.

As an example, in the sentence "Ivan gave Maria all his money", Ivan is the *agent* who causes the action to happen; Maria is the *goal,* as the beneficiary of his action; and all his money is the *theme* that transfers from Ivan to Maria.

To construct a *kernel phrase marker* we have to put together a subset of the following set of frames:

$$NF^{initial},\ NF^{agent},\ NF^{goal},\ NF^{theme},\ NF^{final},\ AF^{initial},\ AF^{medial},\ AF^{final},\ VF \qquad (3)$$

Here NF^{agent}, NF^{goal}, and NF^{theme} are noun frames that play, respectively, the roles of *agent, goal,*

---

9. The present paper will not discuss restrictions on the usage of these two operations. This issue properly belongs to Step One of the Language Generation Procedure.

10. In the implementation we have assumed that the verb-frame VF = tense aux1 aux2 aux3 verb.

or *theme* in a sentence;[11] $NF^{initial}$ and NF^{final} are noun frames that will be transformed later into the initial and final prepositional phrases; $AF^{initial}$, AF^{medial}, and AF^{final} are adverb frames that will be transformed into adverbs in initial, medial, and final positions. All these elements are optional.

In the actual implementation kernel phrase markers have the structure of an association list. We will call it a <u>Semantic Frame Structure</u> (SFS). The English word string corresponding to the Semantic Frame Structure of a kernel phrase marker we will call a *kernel sentence.*

For instance, the Semantic Frame Structure for the kernel sentence "Yesterday the young man bought Maria a beautiful present" is:

((AF^initial ((mod nil) (adverb **yesterday**)))
(NF^agent ((prep nil) (det **the**) (adj **young**) (noun **man**)))
(VF ((tense **past**) (aux1 nil) (aux2 nil) (aux3 nil) (verb **buy**)))
(NF^goal ((prep nil) (det nil) (adj nil) (noun **Maria**)))
(NF^theme ((prep nil) (det **a**) (adj **beautiful**) (noun **present**))))

A set of such Frame Structures together with a set of transformations to be performed upon them is the input to the program GEN. The program does not require any order of constituents in the SFS, but the following phrase structure rules are assumed for kernel sentences:[12]

$$KF \dashrightarrow SF\ \ VF\ \ CF$$

$$SF \dashrightarrow NF^{initial}\ \ NF^{agent} \tag{4}$$

$$CF \dashrightarrow NF^{goal}\ \ NF^{theme}\ \ NF^{final}$$

where KF stands for kernel frame, SF - subject frame, VF - verb frame, CF - complement frame.

Given these phrase structure rules, the general form of a kernel phrase marker is:

$$NF^{initial}\ \ NF^{agent}\ \ VF\ \ NF^{goal}\ \ NF^{theme}\ \ NF^{final} \tag{5}$$

As an example, here is the Semantic Frame Structure for the sentence "that young man with black eyes could have been teaching our class in Boston":

---

11. In Section 7 we will describe how the semantic roles associated with the noun-frames are used to define the transformation *Dative Movement* and its interaction with *Passive.*

12. Adverb frames $AF^{initial}$, AF^{medial}, and AF^{final} have been, for simplicity, left out of the phrase structure rules.

((NF^{agent} ((prep nil) (det **that**) (adj **young**) (noun **man**)
(prep **with**) (det nil) (adj **black**) (noun **eyes**)))

(VF ((tense **past**) (aux1 **can**) (aux2 **have**) (aux3 **be**) (verb **teach**))) (6)

(NF^{theme} ((prep nil) (det **our**) (adj nil) (noun **class**)))

(NF^{final} ((prep **in**) (det nil) (adj nil) (noun **Boston**))))

§2. Transformational Structure

The Semantic Frame Structure provides a semantic description of a kernel sentence which includes constituents such as NF^{agent}, NF^{goal}, and NF^{theme}. Because these semantic constituents represent in some sense a part of the underlying "meaning" of the sentence, their values do not change after applying transformations. However, the SFS does not give a syntactic representation complete enough to allow transformations to be directly applied. We must first construct an augmented representation from the Semantic Frame Structure, the Transformational Structure (TS), which gives a structural encoding of the information in the sentence. This structure will serve as the domain of application for all transformations. The Transformational Structure consists of syntactic constituents such as noun phrases (NP), prepositional phrases (PP), auxiliary system, several *dummy* elements, etc. The process of building the Transformational Structure from the Semantic Frame Structure is described as follows:

(a) The noun phrases and the prepositional phrases of the TS are derived from the noun frames of the SFS. Each noun phrase indicates a certain fixed position in the Transformational Structure: NP_1-position, $NP_{1.5}$-position, or NP_2-position.[13] Each noun phrase is associated with one of the prepositionless noun frames NF^{agent}, NF^{goal}, or NF^{theme} in the Semantic Frame Structure and derives its value from the noun frame. The value of a noun phrase is a "two-tuple"; it consists of a *semantic* value, which is taken from the name of the noun frame and which indicates the role that the noun phrase plays in the sentence: *agent*, *goal*, or *theme*, and a *lexical* value, which is the actual word string in the noun phrase. Transformations (for example, *Passive* or *Dative Movement*) may interchange the values of noun phrases; therefore, after such a transformation has been applied, the affected noun phrase receives not only the new word string as its lexical value, but also the new

13. Later on by a reference to a noun phrase NP_1, $NP_{1.5}$, or NP_2, we will mean the noun phrase which is located in the corresponding position in the Transformational Structure. For instance, reference to NP_1 calls for a noun phrase in the NP_1-position in the TS.

semantic role as its semantic value. In Section 7 we will describe how this important feature is used to define the transformations of *Dative Movement* and *Passive.*

We will assume that the noun phrases NP_1, $NP_{1.5}$, and NP_2 initially obtain their values from the frames NF^{agent}, NF^{goal}, and NF^{theme}, respectively. If only two noun frames NF^{agent} and NF^{theme} are present in the input Semantic Frame Structure, the corresponding Transformational Structure will have only two NP-positions: NP_1 and NP_2.[14] If there is only one noun frame NF^{agent} in the SFS, there will be only one noun phrase NP_1 in the TS. The Semantic Frame Structure in the example (6) will be transformed to:[15]

((NP$_1$ **(That young man with black eyes))**
(TENSE **past**) (AUX1 **can**) (AUX2 **have**) (AUX3 **be**) (VERB **teach**) (7a)
(NP$_2$ **(our class)**) (PP^{final} **(in Boston)**))

(b) A special procedure, "affix stripping", is accomplished on the auxiliary elements of the verb-frame to separate each auxiliary verb from its affix.[16] The affixes of the auxiliaries Modal, BE, and HAVE are, respectively, <u>*0*</u>, <u>*en*</u>, and <u>*ing*</u>. The notation reflects the fact that in an English sentence after a Modal comes an infinitive, after HAVE comes the past participle, and after BE comes the progressive form of the verb [Chomsky 1957]. If the sentence has no auxiliary verbs in it, the auxiliary *do* (without an affix) is inserted as a value of the AUX1.

((NP$_1$ **(That young man with black eyes))**
(TENSE **past**) (AUX1 **can**) (AFFIX1 0) (7b)
(AUX2 **have**) (AFFIX2 en) (AUX3 **be**) (AFFIX3 ing) (VERB **teach**)
(NP$_2$ **(our class)**) (PP^{final} **(in Boston)**))

(c) Several dummy elements are inserted in the TS which are left unspecified[17] until the appropriate transformation activates them and assigns the necessary values. Two elements NEG1 and NEG2 that will be used in the contracted and full forms of English negative are inserted after the first auxiliary verb. COMP is inserted in the beginning of the structure and INFL - before the first auxiliary verb. These will be used in constructing different embedded clauses. Thus the Semantic

---

14. This is the reason for having the obscure indices 1, 1.5, and 2 for the noun phrases.

15. In these diagrams we do not show the *semantic* values of the noun phrases.

16. The inverse operation, *Affix Hopping*, which is performed after all the transformations have been applied, attaches every affix to the immediately following verb.

17. In the implementation the values of unspecified elements NEG1, NEG2, COMP, and INFL are names of those elements.

Frame Structure (6) has now been transformed into Transformational Structure (7c) on which transformations can operate:

((COMP comp) (NP_1 **(That young man with black eyes))**
(TENSE **past**) (INFL infl) (AUX1 **can**) (NEG1 neg1) (NEG2 neg2) (AFFIX1 0) (7c)
(AUX2 **have**) (AFFIX2 en) (AUX3 **be**) (AFFIX3 ing) (VERB **teach**)
(NP_2 **(our class)**) (PP^{final} **(in Boston)**))

The Transformational Structure of a kernel sentence is a *flat list structure* which suggests a more simple and flexible way to define and perform the transformations. On the other hand, the sentence obtained after combining several kernel sentences is represented as an *embedded list structure* (tree structure). This allows the application of the *transformational filters* across the kernels (see Section 6), and also permits an easier interface with the results of other researchers, for example, sentence internal pronominalization (see [Lasnik 1976], [Reinhart 1976], [Sidner 1979]).

§3. Transformations and Adjustments

Let K be a surface word string that corresponds to the input Semantic Frame Structure. In Section 2 it has been described how the Semantic Frame Structure SFS(K) is mapped into the Transformational Structure TS(K). The Transformational Structure serves as the domain of application for all *Transformations* listed in the input TASK. However, after all the transformations have been applied, the resulting Transformation Structure $TS^*(K)$ cannot yet be the source for the correct English word string. Additional "clean-up" operations on the Transformational Structure, *Adjustments*, must be employed in order to obtain the grammatical output -- the altered kernel K^*. Let us consider this process in detail:

(a) The Semantic Frame Structure of the kernel sentence K is mapped {M} into a Transformational Structure TS(K).

(b) A set of transformations {T} is applied. Each transformation operates on the Transformational Structure, alters it and passes the altered structure to the next transformation.

(c) A set of special "clean-up" adjustments {A} is activated in order to get the terminal English word string K^*. The adjustments operate on the altered Transformational Structure $TS^*(K)$.

The following diagram (8) illustrates this process:

$$SFS(K) \xrightarrow{\{M\}} TS(K) \xrightarrow{\{T\}} TS^*(K) \xrightarrow{\{A\}} K^* \qquad (8)$$

Diagram (8) provides a way to define *Transformations* and *Adjustments* which shows an interesting and important distinction between them based on comparing the input kernel sentence K with the output word string K^*.

Definitions:

Assume that the set of transformations {T} is empty. A set of operations {A} on the Transformational Structure TS(K) of an input kernel sentence K will be called *Adjustments* if in diagram (8) the following equality holds: $\mathbf{K} = \mathbf{K}^*$.

An operation T on the Transformational Structure TS(K) of an input kernel sentence K will be called a *Transformation* if in diagram (8) the following inequality holds: $\mathbf{K} \neq \mathbf{K}^*$. In this paper we will use the notation T(K) to refer to the English word string that corresponds to the result of the action of the transformation T on TS(K).

According to these definitions, such operations as *Passive, Question, There Insertion,*[18] etc. are *Transformations* because they alter the original sentence. For example, as the result of successive application of these transformations the sentence "Ivan ate a carrot" will be transformed into "A carrot was eaten by Ivan", "Was a carrot eaten by Ivan" and then to "Was there a carrot eaten by Ivan." The transformations are part of a planning vocabulary, that is, each of them must be listed in the input TASK.

On the other hand, the obligatory operations like *Affix Hopping, Do Deletion, N't Hopping,* etc. together constitute the clean-up *Adjustments*[19] of the Transformational Structure whose goal is to obtain the necessary English output after all the transformations have been applied. They never appear in the input TASK. If the set of transformations {T} is empty, the original kernel sentence K will not be altered.

Adjustments are meaning-preserving, purely syntactical operations on the Transformational Structure. *Transformations,* on the contrary, can affect the meaning (or emphasis) of a kernel sentence. In fact, the meaning of a sentence obtained after combining several kernels altered by transformations is built with the help of these transformations. In that sense, one can consider *Transformations* as semantic (meaning-altering) operations on the Transformational Structure.

18. The description of these transformations will be given in Section 7.

19. Section 5 examines the *Adjustments* and the *Separation* and *Ordering Constraints* imposed upon them.

§4. Commutativity

Definitions:

Domain of Definition $\Omega(T)$ of a transformation T is the set of all sentences K such that T(K) is a grammatical sentence. The fact that the sentence K belongs to the domain of definition of the transformation T is expressed in the following way: $K \subset \Omega(T)$.

Two transformations T_1 and T_2 are *commutative* with respect to a set of kernel phrase markers $\aleph$ if for each phrase marker $K \subset \aleph$ the following relations hold:

$$K \subset \Omega(T_1),\ K \subset \Omega(T_2),\ T_1(K) \subset \Omega(T_2),\ T_2(K) \subset \Omega(T_1) \qquad (9)$$

and

$$T_1(T_2(K)) = T_2(T_1(K)) \qquad (10)$$

In other words, the transformations T_1 and T_2 are *commutative* if the equality (10) holds for the corresponding *domains of definition* of T_1 and T_2.

As an example, let us consider again the sentence K = "Ivan ate a carrot." The commutativity of the transformations T_P = *Passive* and T_Q = *Question* can be checked for this sentence because all the four outputs are grammatical and $T_P(T_Q(K)) = T_Q(T_P(K))$, i.e. we have the following commutative diagram (11):

Question

K = Ivan ate a carrot ---------------> $T_Q(K)$ = Did Ivan eat a carrot

Passive | | *Passive* (11)

Question

$T_P(K)$ = A carrot was eaten by Ivan ---------------> $T_P(T_Q(K))$ = Was a carrot eaten by Ivan

Considering another pair of transformations: T_P = *Passive* and T_{TI} = *There Insertion* shows that the commutativity of the transformations T_P and T_{TI} cannot be checked for the sentence K = "Ivan ate a carrot." The transformations can be applied in the order: 1. *Passive*, 2. *There Insertion*, resulting in:

$T_p(K)$ = A carrot was eaten by Ivan.

$T_{TI}(T_p(K))$ = There was a carrot eaten by Ivan.

But it is not possible to apply the transformations T_p and T_{TI} in the reverse order because the sentence K does not contain the verb *be*, which is necessary for *There Insertion* to work. This does not mean, however, that the two transformations: *Passive* and *There Insertion* are not, or cannot be made, commutative. The previous example does not contradict our definition of commutativity because the sentence K in this example does not belong to the *domain of definition* of the *There Insertion* transformation.

A similar problem arises with the sentence: K = "A boy was eating the cake." Here it is possible to apply either T_p = *Passive*:

$T_p(K)$ = The cake was being eaten by a boy

or T_{TI} = *There Insertion*:

$T_{TI}(K)$ = There was a boy eating the cake

But we cannot apply *There Insertion* after *Passive*, nor can we apply *Passive* after *There Insertion*. In both cases the result will be unacceptable[20] because the noun phrase *the cake* is definite, but *There Insertion* is restricted to sentences with an indefinite first noun phrase:

* There was the cake being eaten by a boy

The following diagram (12) illustrates this example:

```
                                          There Insertion
K = A boy was eating the cake          -------------------->  T_TI(K) = There was a boy eating the cake
                |                                                        |
                |                                                        |
                |                                                        |
         Passive|                                                        | Passive          (12)
                |                                                        |
                |                                                        |
                v                         There Insertion                v
T_p(K) = The cake was being eaten by a boy  ---------------->  * There was the cake being eaten by a boy
```

20. The asterisk * before a sentence indicates that the sentence is "unacceptable", or ungrammatical.

Again, as in the previous example, the fact that we can apply both T_p and T_{11} to the sentence K, but cannot apply T_{11} to $T_p(K)$ or T_p to $T_{11}(K)$, does not mean that there is no commutativity in the system. It means only that in this case the sentence K is irrelevant to the question of commutativity because K does not satisfy the conditions (9) specified in the definition of commutativity; K does not belong to the set of kernel phrase markers $\aleph$ with respect to which commutativity was defined.

One of the requirements that we impose upon the system is the possibility to apply the transformations in any order (equality (10)). Once this is done, the domain membership (conditions (9)) becomes crucial: all the sentences obtained at each step of the transformation derivation must be grammatical.

Definition:

N transformations T_1 T_2 ... T_N are *commutative* with respect to a set of kernel phrase markers $\aleph$ if for any i and j the two transformations T_i and T_j are *commutative* with respect to the set $\aleph$ and if for any m $\leq$ N the result of application of any composition of **m** different transformations T_{i_1} T_{i_2} ... T_{i_m} to a sentence K $\subset \aleph$: $T_{i_1}(T_{i_2} \ldots (T_{i_m}(K))\ldots)$ is a grammatical sentence.

In this paper we will build a system of *Commutative Transformations* that will permit us to "solve" the TASK given by Step One and Step Two of the Language Generation Procedure and to generate corresponding English text. The commutativity of the system gives more flexibility to the first two steps of the LGP. The decisions of how to separate the input semantic representation into groups of kernel phrase markers and which of the transformations to apply will not depend on the order of the transformations. It will be based only on the domain of definition of each particular transformation and on the meaning that this transformation adds to the meaning of the generated sentence.

§5. Adjustments

This section will examine the clean-up *Adjustments*, operations which further alter the Transformational Structure after all the transformations have been applied, in order to obtain the correct English output. We will start with an example. Consider the following kernel sentence K_0 = "Ivan was eating potatoes." The Transformational Structure $TS(K_0)$ for the kernel K_0 is shown in (13):

((COMP comp) (NP_1 **(Ivan)**)
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (AFFIX1 ing) (13)
(AUX2 nil) (AFFIX2 nil) (AUX3 nil) (AFFIX3 nil) (VERB **eat**)
(NP_2 **(potatoes)**))

Suppose that the program GEN gets as input the following TASK:

TASK = ((K_0) (*N't Question*) (.))

Here, *N't* is the notation for the contracted form of the negative transformation. It simply inserts *n't*, the contracted form of *not*, as the value of the element NEG1 of the TS(K_0). *Question* is the name of the question transformation. This transformation takes the three elements: TENSE, AUX1 and NEG1 in the Transformational Structure of a kernel sentence and moves them to the front of the structure.

$TS^*(K_0)$, the altered Transformation Structure of $TS(K_0)$ after undergoing these two transformations, is shown below in (14):

((TENSE past) (AUX1 **be**) (NEG1 **n't**) (COMP comp) (NP_1 **(Ivan)**)
(INFL infl) (NEG2 neg2) (AFFIX1 ing) (14)
(AUX2 nil) (AFFIX2 nil) (AUX3 nil) (AFFIX3 nil) (VERB **eat**)
(NP_2 **(potatoes)**))

Now that all the transformations have been applied to $TS(K_0)$ let us consider how certain *Adjustments* work on the resulting Transformational Structure (14).

(a) *Garbage Deletion* removes all the elements in the TS which have been left unspecified at this point. It also rejects all elements whose value is *nil*. The result is:

((TENSE past) (AUX1 **be**) (NEG1 **n't**) (NP_1 **(Ivan)**)
(AFFIX1 ing) (VERB **cat**) (NP_2 **(potatoes)**))

(b) *Do Deletion* deletes the auxiliary *do* when it immediately precedes a verb. But the Transformational Structure is designed in such a way that the auxiliary *do* is inserted in the TS as a value of the AUX1 only when the original kernel sentence does not contain any other auxiliary verbs. There is no *do* in the Structure (14) since the auxiliary *be* is present, therefore the adjustment *Do Deletion* is not applicable in this example.

(c) *Affix Hopping* takes every affix that is immediately followed by a verb (including TENSE, which as mentioned in Section 1 also carries the grammatical features of NP_1) and attaches it to the verb so that the affix becomes part of it:

((AUX1 **was**) (NEG1 **n't**) (NP_1 **(Ivan)**)
(VERB **eating**) (NP_2 **(potatoes)**))

(d) <u>*N't*</u> <u>*Hopping*</u> takes the element NEG1 of the Transformational Structure and attaches its value *n't* to the immediately preceding auxiliary verb.

((AUX1 **wasn't**) (NP_1 **(Ivan)**) (15)
(VERB **eating**) (NP_2 **(potatoes)**))

Now that all the adjustments have been performed, the word string composed of the values of all the constituents of the Transformational Structure (15) produces the correct kernel sentence K_0^* = "Wasn't Ivan eating potatoes":

Thus, we have examined the following four *Adjustments*:[21]

1. Garbage Deletion
2. Do Deletion (16)
3. Affix Hopping
4. N't Hopping

We are going to show now that in order to build a system of *Commutative transformations* the following two constraints must be imposed:

<u>*Separation*</u> <u>*Constraint*</u>: Any adjustment in the list (16) should be executed only after all the transformations from the TASK have been applied.

<u>*Ordering*</u> <u>*Constraint*</u>: The adjustments must be applied only in the order in which they are listed in (16).

First, in order to prove the necessity of the *Separation Constraint* we need to show that none of the adjustments can be performed before a transformation. This can be shown by contradiction: we will demonstrate for each of the adjustments in (16) that the violation of the *Separation Constraint* prevents the application of certain transformations or leads to an infraction of commutativity.

---

21. In the implemented system there are a few other adjustments, but for simplicity we will not discuss them here.

Finding one counterexample to every adjustment is sufficient for a proof because of the *Commutativity* of the system. For clarity, we will consider the arguments in connection with either the kernel sentence "Ivan ate the carrot" and its corresponding Transformational Structure (17):

((COMP comp) (NP_1 **(Ivan)**)
(TENSE past) (INFL infl) (AUX1 **do**) (NEG1 neg1) (NEG2 neg2) (VERB **eat**) (17)
(NP_2 **(the carrot)**))

or with the kernel sentence "Ivan was eating the carrot" and its Transformational Structure (18):

((COMP comp) (NP_1 **(Ivan)**)
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (AFFIX1 ing) (18)
(VERB **eat**)
(NP_2 **(the carrot)**))

1. *Garbage Deletion* cannot be performed before any transformation that uses the dummy elements in the Transformational Structure; for instance, *Negation*, or any transformation that constructs embedded clauses. This is because the elements required by such transformations would be deleted by the *Garbage Deletion* adjustment before these transformations could have been applied.

2. *Do Deletion* cannot be performed before the *Question* transformation. If it were, the auxiliary verb *do* which is necessary for the construction of the correct output of the *Question* transformation, "Did Ivan eat the carrot", will be prematurely deleted by the adjustment *Do Deletion.*

3. *Affix Hopping* cannot be performed before any transformation that inserts a new verb or a new affix into the Transformational Structure. Suppose that the transformation *Passive* is to be applied to the Transformational Structure (18) of the kernel sentence "Ivan was eating the carrot." Among other effects this transformation inserts the verb be with the affix en in front of the main verb eat into the Transformational Structure (18) in order to obtain the past participle of the verb.[22]

((COMP comp) (NP_1 **(The carrot)**)
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (AFFIX1 ing)
(BE-PASS **be**) (EN-PASS **en**) (VERB **eat**)
(BY-PASS **by**) (NP_2 **(Ivan)**))

If the adjustment *Affix Hopping* had been applied to the Transformational Structure (18) before the

22. The description of the *Passive* transformation can be found in Section 7.

Passive transformation, the affix ing would "hop" onto the wrong verb eat instead of the verb be. The affix en, inserted afterwards by the *Passive* transformation, could not "hop" on the verb eat. As a result, the system would not be able to output the expected sentence "The carrot was being eaten by Ivan."

4 The adjustment *N't Hopping* also cannot precede certain transformations. Let us consider the Transformational Structure TS* for the sentence "Ivan ate the carrot" after the two transformations *Question* and *N't* have been applied:

((TENSE past) (AUX1 **do**) (NEG1 **n't**) (COMP comp) (NP_1 **(Ivan)**)
(INFL infl) (NEG2 neg2) (VERB **eat**)
(NP_2 **(the carrot)**))

If the adjustment *N't Hopping* is now applied before the transformation *Passive*, the element n't will be prematurely attached to the auxiliary verb do. As a result, it will be impossible to execute the transformation *Passive* and transform the sentence "Didn't Ivan eat the carrot" into the sentence "Wasn't the carrot eaten by Ivan."

We have shown that the adjustments are separated from the transformations. Now, to demonstrate that the *Ordering Constraint* is needed, we will again examine the sentence "Ivan ate the carrot" and its Transformational Structure (17). We need to show that the adjustments must necessarily be ordered as stated in (16): 1. Garbage Deletion, 2. Do Deletion, 3. Affix Hopping, 4. N't Hopping.

1. *Garbage Deletion* differs from the other three adjustments in (16) because it does not employ the notion of adjacency required by all the other adjustments: *Do Deletion* deletes the auxiliary *do* when it immediately precedes a verb; *Affix Hopping* attaches the affix to the verb immediately to its right; *N't Hopping* attaches *n't* to the immediately preceding auxiliary verb.

Prior to the adjustment *Garbage Deletion*, the adjacency condition necessary for the application of the other adjustments does not hold because of the presence of unspecified elements in the Transformational Structure. *Garbage Deletion* should be executed first because it eliminates these unspecified elements allowing the other adjustments to work. For example, *Garbage Deletion* removes the unspecified elements NEG1 and NEG2 in the Transformational Structure shown below, making it possible for *Do Deletion* to delete the auxiliary *do* since *do* and the main verb are now adjacent:

((NP_1 **(Ivan)**)
(TENSE past) (AUX1 **do**) (VERB **eat**)
(NP_2 **(the carrot)**))

It should also be noted that the adjustments *Do Deletion*, *Affix Hopping*, and *N't Hopping* do not introduce any dummy or unspecified elements in the TS which would otherwise have to be removed by the *Garbage Deletion.*

2. *Do Deletion* must be performed before *Affix Hopping.* Otherwise, the value *past* of the TENSE would "hop" onto the verb *do* instead of *eat*, and after executing all the adjustments on the Transformational Structure (17) we would obtain the sentence "Ivan did eat the carrot" instead of "Ivan ate the carrot."[23]

3. The adjustment *Affix Hopping* cannot follow *N't Hopping.* To show this suppose that the transformation *N't* has been applied to the Transformational Structure (17). After the *Garbage Deletion* adjustment the TS will have the following form:

((NP_1 **(Ivan)**)
(TENSE past) (AUX1 **do**) (NEG1 **n't**) (VERB **eat**)
(NP_2 **(the carrot)**))

If now the adjustment *N't Hopping* is executed first, producing the new element *don't*, the *Affix Hopping* adjustment will not be able to attach the TENSE to the element *don't.* This observation completes the proof of the *Ordering Constraint.*

Thus two different classes of operations on the Transformational Structure have been defined in this paper so far: the optional commutative *Transformations* and the obligatory, intrinsically ordered *Adjustments.* The *Separation Constraint* and the *Ordering Constraint* explain their mutual relations.

§6. Connective Transformations

In this section we will examine the *Connective Transformations*, a particular class of English transformations which are used to construct sentences with *Embedded Clauses.* In order to form a

---

23. The sentence "Ivan did eat the carrot" is, of course, fully grammatical. It is the *emphatic* form of the sentence "Ivan ate the carrot" used to stress a certain semantic message. In designing the system we decided not to allow such sentences so that all the adjustments would be obligatory.

sentence of this type the system needs two kernel phrase markers as input: K_0 - which plays the role of *matrix clause* and K_1 - which is used as the basis for the *embedded clause*. The word it will be used as a *joining point*. The kernel sentence K_0 must contain the word it, in the role of either agent or theme. A *Connective Transformation* when applied to the kernel sentence K_1 is said to produce the altered kernel K_1^* in the form appropriate for an embedded clause. Other transformations (if any) change the kernel K_0 into K_0^*. Then, a special procedure combines the two altered kernel sentences by substituting the kernel K_1^* for the joining point it in the kernel K_0^*.

As an example of the procedure just described, consider the following kernel sentences:

K_0 = It bothers Maria (19)
K_1 = Ivan has ignored that letter

The embedded clause K_1 has the Transformational Structure below:

((COMP comp) (NP_1 **(Ivan)**)
(TENSE present) (INFL infl) (AUX1 **have**) (NEG1 negl) (NEG2 neg2) (AFFIX1 en) (20)
(VERB **ignore**) (NP_2 **(that letter)**))

Now suppose that one of the *Connective Transformations*, FOR-NP_1-TO_1, is to be applied. This transformation produces a *for-to*-complement clause by inserting FOR and TO as the values of the elements COMP and INFL, respectively, in the Transformational Structure (20). The result is:

((COMP **For**) (NP_1 **(Ivan)**)
(TENSE present) (INFL **to**) (AUX1 **have**) (NEG1 negl) (NEG2 neg2) (AFFIX1 en)
(VERB **ignore**) (NP_2 **(that letter)**))

Then the transformed kernel sentence K_1^* = "For Ivan to have ignored that letter" is substituted for the word it in the kernel K_0 = "It bothers Maria" producing the sentence: "For Ivan to have ignored that letter bothers Maria" as the result.

The *Connective Transformations* form a family of transformations defined by three parameters, each referring to a position in the Transformational Structure of the embedded clause: the complementizer COMP, the first noun phrase NP_1 and the complementizer INFL.

The parameter COMP indicates the affix which introduces the embedded clause. It can receive one of the following four values: {POSS, THAT, FOR, 0}. The affix POSS actually "hops" onto the following noun during the *Affix Hopping* adjustment to form a possessive noun phrase. THAT and

FOR are "independent" words and do not hop to the right like other affixes. 0 means that there is no overt affix in this position.

The parameter NP_1 takes on one of two values: {NP_1, 0} indicating whether or not the first noun phrase of the embedded clause is present in the resulting sentence.

The parameter INFL may also be considered an affix specifying how the verb in the embedded clause should be inflected. It can have one of four values: {ING, INF, TO, 0}. As in the case of POSS, the value ING is the only "real" affix which hops onto the following verb producing the progressive form. The value INF indicates that the following verb is in the infinitive form. TO is an independent word which produces the TO + infinitive form of the verb. 0 signals that no affix is present, so the verb remains inflected for person, number and tense.

A particular set of values for the three parameters COMP, NP_1, and INFL is sufficient to determine the form of the embedded clause completely (although not every possible triple produces a grammatical English clause). Hence the commutative transformation which generates the embedded clause is also uniquely determined. Therefore, we will use the values of these three parameters to form the names for the *Connective Transformations*. Each name is represented as a triple consisting of the current values of the parameters COMP, NP_1 and INFL with the following structure: COMP-NP_1-INFL. We have already seen some examples of such names: 0-NP_1-TO, THAT-NP_1-0, FOR-NP_1-TO, etc.

The adjustment *Affix Hopping* deletes the element TENSE in the Transformational Structure unless the value of the parameter INFL is 0. Consequently, most of the *Connective Transformations* construct tenseless clauses. The exceptions are the transformations THAT-NP_1-0 and 0-NP_1-0. To give examples with tensed embedded clauses let us consider now another kernel sentence for the matrix clause: K_0 = "Maria knows it." After combining the altered kernels, the program outputs the following sentences:

1. After THAT-NP_1-0_1: Maria knows that Ivan has ignored that letter

2. After 0-NP_1-0_1: Maria knows Ivan has ignored that letter

The transformations THAT-NP_1-0 and 0-NP_1-0 also differ from the transformations which produce tenseless clauses in their treatment of negative elements in the Transformational Structure.

A general description of the action of a *Connective Transformation* can be stated as follows:

(a) Current values of the parameters COMP and INFL from the name of the transformation are inserted in the Transformational Structure.

(b) If the value of the parameter NP_1 is 0, the element NP_1 is removed from the Structure.

(c) If the value of INFL is not 0, two negative elements, NEG1 and NEG2, are moved from their usual position after the first auxiliary verb and placed in the front of TENSE. We will call this phenomenon *Neg Jump.*

When the value of the parameter NP_1 is 0, the first noun phrase in the matrix clause must be coreferential with the first noun phrase in the embedded clause. As an example, consider the pair of kernel sentences:

K_0 = Ivan wanted it
K_1 = Ivan kissed Maria

Suppose that the transformation 0-0-TO applies. This transformation inserts the required values of COMP and INFL into the Transformational Structure for the kernel sentence K_1 and deletes the element NP_1 in the Structure:

((COMP 0) (NP_1 0) (NEG1 neg1) (NEG2 neg2)
(TENSE past) (INFL to) (AUX1 do)
(VERB kiss) (NP_2 (Maria)))

After all the adjustments have been performed and the altered kernels have been combined the program will output: "Ivan wanted to kiss Maria."

In the example above *Neg Jump* takes place only vacuously because the unspecified elements NEG1 and NEG2 are removed from the Transformational Structure. It is also difficult to see this phenomenon if the kernel sentence K_1 does not have any auxiliary verbs because after the *Garbage Deletion* adjustment nothing will be left in the TS for NEG to jump over. This phenomenon can be observed in a sentence in which the negative transformation *Not* also applies. The following TASK provides us with the appropriate examples:

TASK = ((K_0 K_1) (*That-np*$_1$*-0*$_1$ *Not*$_1$ *For-np*$_1$*-to*$_1$ *Poss-np*$_1$*-ing*$_1$) (.))

Here K_0 and K_1 are the kernel sentences from (19):

K_0 = It bothers Maria (19)
K_1 = Ivan has ignored that letter

Neg Jump can be observed by comparing either sentences #2 with #3 or sentences #. with #4 in (21):

1. After THAT-NP_1-0_1:	That Ivan <u>has</u> ignored that letter bothers Maria	
2. After NOT_1:	That Ivan <u>has</u> <u>not</u> ignored that letter bothers Maria	(21)
3. After FOR-NP_1-TO_1:	For Ivan <u>not</u> to <u>have</u> ignored that letter bothers Maria	
4. After POSS-NP_1-ING_1:	Ivan's <u>not</u> <u>having</u> ignored that letter bothers Maria.	

The system contains a family of ten different *Connective Transformations* used to construct sentences with various embedded clauses. An example of the application of each of these transformations is given in the table below:

TABLE 1

Transformation	*Matrix Clause*	*Embedded Clause*	*Sentence*
THAT-NP_1-0	It bothers Maria	Ivan ignored the letter	That Ivan ignored the letter bothers Maria
THAT-NP_1-INF	Maria suggests it	Ivan is silent	Maria suggests that Ivan be silent
0-NP_1-INF	Maria watched it	Ivan washed the dishes	Maria watched Ivan wash the dishes
0-NP_1-0	Maria knows it	Ivan has ignored that letter	Maria knows Ivan has ignored that letter
FOR-NP_1-TO	It amuses Maria	Ivan ignored the letter	For Ivan to ignore the letter amuses Maria
0-NP_1-TO	Maria asked it	Ivan ate the cake	Maria asked Ivan to eat the cake
0-0-TO	Ivan claims it	Ivan has written that letter	Ivan claims to have written that letter
POSS-NP_1-ING	It shocked Maria	Ivan ignored the letter	Ivan's ignoring the letter shocked Maria
0-NP_1-ING	Maria saw it	Ivan ate the cake	Maria saw Ivan eating the cake
0-0-ING	It amuses Maria	Maria watches movies	Watching movies amuses Maria

The main verb of the matrix clause determines which clause may be embedded under it and, therefore, the kind of *Connective Transformation* that may be applied in each particular case. Consequently, any computer implementation of the Language Generation Procedure must contain a list of permissible transformations for every verb that can appear in a matrix clause.

The system prevents the generation of the ill-formed sentences by employing *transformational filters*, a third class of operations on the Transformational Structure, which not only signal that a certain combination of the transformations will result in an ungrammatical sentence, but also, if possible, suggest additional rules in order to correct the output.[24] Although this paper will not consider such rules in detail, here are several illustrative examples:

(a) If the transformations THAT-NP_1-0_1 and PASSIVE have been applied to kernel sentences K_0 and K_1 from (19), the resulting sentence obtained after combining the altered kernels is ill-formed: * "Maria is bothered by that Ivan has ignored that letter." The transformational filter rejects the ungrammatical construction *prep-THAT-clause* and suggests that the words "the fact" be inserted in the Transformational Structure of the embedded clause. Then, after all the *Adjustments* have been applied, the system outputs the correct sentence: "Maria is bothered by the fact that Ivan has ignored that letter."[25]

(b) If the transformations 0-NP-TO_1 and PASSIVE are to be performed on the kernels K_0 = "Maria forced it" and K_1 = "Ivan peeled potatoes", the program has to combine the altered kernel K_0^* = "It was forced by Maria" with the altered kernel K_1^* = "Ivan to peel potatoes." The transformational filter rejects the resulting sentence: * "Ivan to peel potatoes was forced by Maria" as ungrammatical. Then, taking into account the semantic class of the main verb *force*, the filter proposes to "raise" the first noun phrase of the embedded clause *Ivan* to the position of the first noun phrase in the matrix clause *it*; the rest of the embedded clause moves to the end of the matrix clause. Now the resulting sentence is correct: "Ivan was forced by Maria to peel potatoes."

(c) If the embedded clause is very long, the transformational filter suggests applying the *Extraposition* rule, a rule which shifts the embedded clause to the end of the matrix clause leaving the word "it" in its original position in the matrix clause. Thus, the sentence #1 from (21) can be changed to "It bothers Maria that Ivan has ignored that letter."[26]

A family of ten transformations for generating embedded clauses was introduced in this section. This family is completely defined by the values of three parameters COMP, NP_1, and INFL and, therefore, can be considered as a single Connective Transformation whose surface manifestation has several different forms depending on the values of these parameters.

---

24. In contrast with the usual transformations, which operate within a kernel sentence, one can analyze *transformational filters* as *interkernel transformations* because they may operate across the kernels.

25. The noun "fact" is one of the so-called *factive* nouns which include also words like "idea", "report", etc.

26. We leave aside for now the question of defining "length". Notice, that the *Extraposition* rule may also apply to short embedded clauses: "It bothers Maria that Ivan left."

§7. Other Transformations

In this section we will add three new transformations to the system of *Commutative Transformations*: T_P = *Passive*, T_{DM} = *Dative Movement*, and T_{TI} = *There Insertion*. The definitions of these transformations refer to the noun phrase positions in the Transformational Structure: NP_1, $NP_{1.5}$, NP_2, and to their lexical and semantic values.

Definition:

Passive is a transformation that

(a) Permutes to the left the values of three noun phrases NP_1, $NP_{1.5}$, and NP_2

```
 --------------------------------->
 ↑                               ↓
NP1 <--------- NP1.5 <---------NP2
```

(b) Inserts the verb *be* after the affix of the last auxiliary verb or, if AUX1 = *do*, substitutes the verb *be* for the auxiliary *do*

(c) Inserts the affix *en* in front of the main verb

(d) Inserts the preposition *by* in front of NP_2[27]

Consider the kernel sentence K = "Ivan gave Maria the bottle" with the Transformational Structure (22):

((COMP comp) (NP_1 **(Ivan))**
(TENSE past) (INFL infl) (AUX1 **do**) (NEG1 neg1) (NEG2 neg2) (22)
(VERB **give**) ($NP_{1.5}$ **(Maria)**) (NP_2 **(the bottle)))**

The *Passive* transformation alters the kernel sentence K = "Ivan gave Maria the bottle" into the sentence $T_P(K)$ = "Maria was given the bottle by Ivan" by changing the Transformational Structure (22) into:

((COMP comp) (NP_1 **(Maria))**
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (23)
(EN-PASS en) (VERB **give**) ($NP_{1.5}$ **(the bottle)**) (BY-PASS **by**) (NP_2 **(Ivan)))**

27. In this definition, when referring to the words *be*, *en*, and *by* we mean elements (BE-PASS be), (EN-PASS en), and (BY-PASS by) in the Transformational Structure.

In the case when there are only two noun phrases in the input kernel sentence, to permute two NP's means the same as to interchange them. In this case our definition comes closer to the conventional definition of *Passive.*

The transformation *Dative Movement* changes the kernel sentence K = "Ivan gave Maria the bottle" into the sentence $T_{DM}(K)$ = "Ivan gave the bottle to Maria." A possible definition of *Dative Movement* could refer to the syntactic constituents $NP_{1.5}$ and NP_2 in the Transformational Structure:

(a) Interchange the values of $NP_{1.5}$ and NP_2
(b) Insert the word *to* in front of NP_2

This definition of *Dative Movement* when applied to the kernel K gives the desired result $T_2(K)$ = "Ivan gave the bottle to Maria." Suppose, however, that the *Passive* transformation is to be applied to the kernel K prior to *Dative Movement* producing the Transformational Structure (23) corresponding to the sentence $T_p(K)$ = "Maria was given the bottle by Ivan." If *Dative Movement* (as it is defined above) is now applied to $T_p(K)$, the Transformational Structure will be transformed into:

((COMP comp) (NP_1 (**Maria**))
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2)
(EN-PASS en) (VERB **give**) ($NP_{1.5}$ (**Ivan**)) (BY-PASS **by**) (DAT **to**) (NP_2 (**the bottle**)))

The resulting sentence $T_{DM}(T_p(K))$ is unacceptable: * "Maria was given Ivan by to the bottle."

In order to obtain the correct result we need a different definition of *Dative Movement* which refers not to NP-positions in the Transformational Structure, but rather to semantic values of the noun phrases which indicate the role that the noun phrase plays in the sentence: *agent, goal,* or *theme.*

In this paper several transformations which operate on the Transformational Structure of a kernel phrase marker have been examined: the *Question Transformation. Negation* (full and contracted), *Passive*, and the *Connective Transformation.* All of these transformations are defined in purely syntactical terms: they move, insert, or delete syntactic constituents in the Transformational Structure. So far, we have seen semantic notions used only in the "decision-making" procedures. For instance, the system needs the semantic class of the main verb in the matrix clause to determine which of the *Connective Transformations* may be applied and what rule should combine the altered kernel sentences.

Dative Movement is the only transformation in our system that is defined in semantical terms.

Definition:

Dative Movement is a transformation that

(a) Interchanges the noun phrase which plays the role of *goal* with the noun phrase which plays the role of *theme*

(b) Depending on the main verb, assigns one of two values {TO, FOR} to the element DAT which will later be inserted in front of the *goal* noun phrase in the TS by a special adjustment *DAT Insertion*

The result of this transformation is that the *goal* noun phrase will be in the position previously occupied by *theme* noun phrase and the *theme* NP will be in the position occupied by *goal* NP.

For instance, the *Dative Movement* transforms the Transformational Structure (22) of the kernel sentence K = "Ivan gave Maria the bottle" into:

((COMP comp) (NP_1 **(Ivan)**)
(TENSE past) (INFL infl) (AUX1 do) (NEG1 neg1) (NEG2 neg2) (24)
(VERB **give**) ($NP_{1.5}$ **(the bottle)**) (NP_2 **(Maria)**))

Then the adjustment *DAT Insertion* inserts the element (DAT to) into the Transformational Structure (24) resulting in (25):

((COMP comp) (NP_1 **(Ivan)**)
(TENSE past) (INFL infl) (AUX1 do) (NEG1 neg1) (NEG2 neg2) (25)
(VERB **give**) ($NP_{1.5}$ **(the bottle)**) (DAT **to**) (NP_2 **(Maria)**))

After all the other adjustments have been performed the system outputs the correct sentence "Ivan gave the bottle to Maria."[28]

Not only do we obtain the correct results by applying each of two transformations *Passive* and *Dative Movement* to the TS of the kernel sentence, but also, our definitions are so formulated that these transformations can be applied in any order. *Dative Movement* can be applied to the Transformational Structure altered by *Passive* and vice versa.

---

28 In the usual analysis (see, for instance, [Akmajian and Heny 1975]) *Dative Movement* transforms the sentence "Ivan gave the bottle to Maria" into the sentence "Ivan gave Maria the bottle."

Let us, for example, apply *Dative Movement* to the TS in (23) altered by the *Passive* transformation. The noun phrase "Maria" which plays the role of *goal* is interchanged with the noun phrase "the bottle" which plays the role of *theme*. Then, the *DAT Insertion* adjustment inserts the preposition *to* in front of the *goal* noun phrase "Maria" producing the TS (26):

((COMP comp) (NP_1 **(the bottle)**)
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (26)
(EN-PASS en) (VERB **give**) (DAT **to**) ($NP_{1.5}$ **(Maria)**) (BY-PASS **by**) (NP_2 **(Ivan)**))

The resulting sentence is $T_{DM}(T_p(K))$ = "The bottle was given to Maria by Ivan." The Transformational Structure (26) can also be obtained if the transformation *Passive* is applied to the TS (24) (that is, after *Dative Movement* has been applied).

The following diagram (27) illustrates the commutativity of these transformations:

Dative Movement

K = Ivan gave Maria the bottle --------------------> $T_{DM}(K)$ = Ivan gave the bottle to Maria

Passive | | *Passive* (27)

↓ *Dative Movement* ↓

$T_p(K)$ = Maria was given the bottle by Ivan --------------------> The bottle was given to Maria by Ivan

The final transformation to be considered is *There Insertion*, defined below.

Definition:

There Insertion is a transformation that

(a) Substitutes the word *there*[29] for the the NP_1-position in the TS

29. More precisely, the element (TH there) of the Transformational Structure.

(b) Moves the NP_1-position in front of the affix associated with the leftmost occurrence of the verb *be* in the Transformational Structure

As an example, consider the kernel sentence K = "A boy was eating a cake" with the Transformational Structure (28):

((COMP comp) (NP_1 (a **boy**))
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (AFFIX1 ing) (28)
(VERB **eat**) (NP_2 (a **cake**)))

There is only one verb *be* in the TS above, therefore, AFFIX1 is the required affix. After applying the transformation *There Insertion* the Transformational Structure (28) will be transformed to (29):

((COMP comp) (TH **there**)
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (29)
(NP_1 (a **boy**)) (AFFIX1 ing) (VERB **eat**) (NP_2 (a **cake**)))

The corresponding English output is "There was a boy eating a cake."

Suppose, however, that *Passive* was applied to the Transformational Structure (28) causing another verb *be* to be inserted after AFFIX1:

((COMP comp) (NP_1 (a **cake**))
(TENSE past) (INFL infl) (AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (AFFIX1 ing) (30)
(BE-PASS **be**) (EN-PASS en) (VERB **eat**) (BY-PASS **by**) (NP_2 (a **boy**)))

Here the leftmost occurrence of *be* in the Structure (30) is the one referred to in the definition of the transformation *There Insertion* and the position of the first noun phrase is moved in front of its affix:

((COMP comp) (TH **there**) (TENSE past) (INFL infl)
(AUX1 **be**) (NEG1 neg1) (NEG2 neg2) (NP_1 (a **cake**)) (AFFIX1 ing) (31)
(BE-PASS **be**) (EN-PASS en) (VERB **eat**) (BY-PASS **by**) (NP_2 (a **boy**)))

The Transformational Structure (31) produces the sentence: "There was a cake being eaten by a boy."

Consider now the question of the commutativity of the three transformations introduced in this section. It follows from the definitions that *There Insertion* does not change the original disposition of the noun phrases NP_1, $NP_{1.5}$, and NP_2 in the Transformational Structure. Therefore, since *There Insertion* does not interfere with the action of *Passive* and *Dative Movement*, it does not prevent these transformations from permuting or interchanging the noun phrases. It is thus possible to apply the *Passive* transformation and *Dative Movement* after *There Insertion*. On the other hand, *Dative Movement* and *Passive* may freely interchange the noun phrases in the TS because it makes no difference to the transformation *There Insertion* which of the noun phrases is located in the NP_1-position.[30] *There Insertion*, therefore, can be applied after *Passive* and *Dative Movement*. Since we have shown that *There Insertion* may precede or follow *Passive* and *Dative Movement*, and that the transformations *Passive* and *Dative Movement* are commutative, it follows that all three transformations introduced in this section are commutative.

The application of *There Insertion* is restricted to sentences with an indefinite noun phrase in the NP_1-position, but the transformations *Passive* and *Dative Movement* interchange the noun phrases in the Transformational Structure. We, therefore, need a sentence with three indefinite noun phrases to provide an appropriate example for showing the commutativity of the transformations. In this case the result of application of *There Insertion* would be grammatical regardless of the order of the transformations. Suppose that K = "A man was reading a boy some interesting stories." If the three transformations are applied in the order: 1. *There Insertion*, 2. *Passive*, 3. *Dative Movement*, the result is:

1. After	*There Insertion*:	There was a man reading a boy some interesting stories.
2. After	*Passive*:	There was a boy being read some interesting stories by a man.
3. After	*Dative Movement*:	There were some interesting stories being read to a boy by a man.

The reverse order of the transformations: 1. *Dative Movement*, 2. *Passive*, 3. *There Insertion* produces the following sentences:

1. After	*Dative Movement*:	A man was reading some interesting stories to a boy.
2. After	*Passive*:	Some interesting stories were being read to a boy by a man.
3. After	*There Insertion*:	There were some interesting stories being read to a boy by a man.[31]

30. *There Insertion* merely moves the NP_1-position in the Transformational Structure to the right.

31. We will not give here examples of all possible permutations of these transformations because the commutativity of *Passive* and *Dative Movement* has been already shown before.

This section has introduced three additional transformations that refer to noun phrase positions and their values.[32] The *Passive* transformation refers to the positions NP_1, $NP_{1.5}$, and NP_2 permuting their values. *Dative Movement* refers to the semantic values *goal* and *theme* and interchanges the corresponding NP-positions. *There Insertion* moves the NP_1-position in the Transformational Structure.

Step Three of the Language Generation Procedure uses the constructed system of optional commutative *Transformations*, the set of conditionally obligatory *Transformational Filters*, and the set of ordered obligatory *Adjustments* to "solve" the TASK provided by Step One and Step Two and to produce the corresponding English text.

32. The system contains other transformations, such as the *Relative* transformation, *Imperative* and *Tag Question*, but they will not be discussed here.

APPENDIX 1: **Pronominalization**

The program GEN which implements Step Three of the Language Generation Procedure "reads" the input TASK supplied by Step One and Step Two of the LGP and generates the appropriate word string. Each element of the TASK consists of a list of kernel phrase markers, a list of transformations, and a syntactic separator. There is a one-to-one correspondence between each TASK element and each English sentence in the output. Every noun phrase in a generated sentence derives its lexical value from one of the prepositionless noun frames, NF^{agent}, NF^{goal}, or NF^{theme}, of the Semantic Frame Structure. Transformations may interchange the values of noun phrases, but the actual word string is never altered; it can only be transferred into another NP-position in the Transformational Structure. Therefore, if different phrase markers of the TASK refer to the same noun frame, each instance of the corresponding noun phrase would have the same lexical value.

This repetition is awkward and in order to make the text more fluent, the system requires a *pronominalization* procedure which substitutes the repeated noun phrase with an appropriate pronoun. The choice of the pronoun is determined by two parameters, the *number* and *gender* of the corresponding noun phrase. The values of these two parameters can be computed from the number and gender of the nouns which are heads of the noun phrases and the conjunctions that connect them.[33]

The pronominalization procedure is activated every time the system "reads" the next element of the TASK and generates the corresponding English sentence. In order to decide which noun phrase should be pronominalized, two subsequent sentences, referred to by the names *current* and *previous*, are examined.

<u>Definitions.</u>

Current-NP-list is a list of all noun phrases in the last generated (*current*) sentence.

Previous-NP-list is a list of noun phrases in the *previous* sentence.

Previous-pronouns-list (PPL) is a list of pronouns corresponding to each element of the *previous-NP-list.*[34]

---

33. In this paper we will not describe the rules that the system uses to calculate the number and gender of a given noun phrase. For instance, the number of the noun phrase "Ivan and Maria" is "plural", the gender is "indifferent." Or, in the case of another noun phrase, "the man with a red tie", the number is "singular", the gender is "masculine". Similar rules can be found in [Katz 1978].

34. All the pronouns contained in the *previous-pronouns-list* are in the nominative case. But after the decision to pronominalize a noun phrase has been made, the appropriate lexical form of a pronoun (nominative, objective, or possessive) is chosen depending on the case of the corresponding noun phrase.

Suppose, for example, that the following two sentences were generated by the system:

"Maria suspects that Tania is writing a letter to Ivan. Tania loves Ivan." (32)

In this example we have:

current-NP-list = ((Tania) (Ivan))
previous-NP-list = ((Maria) (Tania) (a letter) (Ivan))
previous-pronouns-list = ((she) (she) (it) (he))

Every noun phrase in the *current-NP-list* which is also present in the *previous-NP-list* is a possible candidate for pronominalization. But the decision to pronominalize each noun phrase is made only after comparing the pronoun that corresponds to the noun phrase under consideration with all the other elements of the PPL.

> *Pronominalization Rule:* A repetitive noun phrase in the *current* sentence is replaced by its pronoun only if the pronoun is *unique* in the *previous-pronouns-list* (that is, no other noun phrases in the *previous* sentence has the same pronoun).

In the example (32) the program will not substitute the pronoun "she" for the noun phrase "Tania" because this pronoun is not unique in the PPL. The *Pronominalization Rule* helps the system avoid ambiguity that arises when this substitution is made: "Maria suspects that Tania is writing a letter to Ivan. She loves Ivan."

On the other hand, since there is only one occurrence of "he" in the PPL, the noun phrase "Ivan" can be pronominalized, resulting in: "Maria suspects that Tania is writing a letter to Ivan. Tania loves him."[35]

The *Pronominalization Rule,* however, is a necessary but not a sufficient condition for the pronominalization procedure. In some cases complementary heuristic rules must be employed in order to make the final decision.

35. It should be noted that more than one pronoun is possible in a sentence as long as the *Pronominalization Rule* is not violated. The example is: "Many people suspect that Tania is writing a letter to Ivan. She loves him."

APPENDIX 2: The Parser

This section describes the *parser*, a program that processes kernel sentences and builds the corresponding Semantic Frame Structures using the three types of templates defined in Section 1 of this paper:

noun-template (NT) = (prep* det* adj* noun)

verb-template (VT) = (tense aux1 aux2 aux3 verb) (34)

adverb-template (AT) = (mod adverb)

Each word in the kernel sentence is always associated with a unique part of speech,[36] and, therefore, with a unique position inside one of the templates NT, VT, or AT. We will assume that a kernel sentence can be represented by the following sequence of frames:[37]

$$NF^{initial}\ NF^{agent}\ VF\ NF^{goal}\ NF^{theme}\ NF^{final} \quad (35)$$

Each constituent in (35) can be obtained by applying the two operations *Concatenation* (CONC) and *Conjunction* (CONJ)[38] to the templates NT, VT, or AT in (34). All constituents are optional: for example, the presence of the frames NF^{goal} and NF^{theme} depends on the type of main verb in the kernel sentence (transitive, intransitive, or double-transitive).

The parser analyses every word in the input kernel sentence, scanning it from left to right and mapping the appropriate pieces of the word string onto the corresponding templates. The parser then decides which of the templates should be *concatenated* or *conjoined* in order to form the necessary noun-frame, verb-frame, or adverb-frame. Because each word in the sentence is associated with a unique template, the parser starts to create the appropriate type of template ("opens" the template) after examining the very first word in the sentence. The template must be filled out from left to right. If any element in the template is left unspecified, the parser inserts *nil* as the value of this element. When all the elements of the template are filled out, the parser "closes" the template. Then, depending on the next word and its position in the sentence, the parser has two choices:

---

36. The question of *lexical ambiguity*, that is the case when one word can serve as various parts of speech, is not discussed in this paper.

37. For simplicity, the adverb frames $AF^{initial}$, AF^{medial}, and AF^{final} are not shown here.

38. These operations are defined in Section 1.

(a) it either continues the construction of the frame, starting to create another instance of a template of the same type, or

(b) it "closes" the current frame and begins to fill out the elements of another template, thereby starting a new frame.

Here is an example to clarify the procedure. Suppose that the parser takes the following sentence as input:

"In the evening a young tall man with blue eyes gave Maria a beautiful book and a rose." (36)

Triggered by the preposition "in", the parser begins the construction of the noun-frame $NF^{initial}$ by filling out the noun-template as follows. ((prep **in**) (det **the**) (adj nil) (noun **evening**)). The next word in the sentence is the determiner "a", which indicates the absence of a preposition in the next noun-template, and, therefore, suggests that the frame ($NF^{initial}$ ((prep **in**) (det **the**) (adj nil) (noun **evening**))) should be closed and that the construction of a new prepositionless noun-frame NF^{agent} should begin. This frame consists of the concatenation of two noun-templates:

(NF^{agent} ((prep nil) (det **a**) (adj **(young tall)**) (noun **man**)
(prep **with**) (det nil) (adj **blue**) (noun **eyes**)))

There is no auxiliary verbs in the sentence (36), and hence the parser builds from the verb "gave" the following verb-frame (VF ((tense **past**) (aux1 nil) (aux2 nil) (aux3 nil) (verb **give**))) with the unspecified auxiliary elements. Then, the word "Maria" forms another noun-frame (NF^{goal} ((prep nil) (det nil) (adj nil) (noun **Maria**))). Finally, the parser uses the operation *Conjunction* to construct the last prepositionless noun-frame NF^{theme} from two noun-templates:

(NF^{theme} ((prep nil) (det **a**) (adj **beautiful**) (noun **book**) (conj **and**)
(prep nil) (det **a**) (adj nil) (noun **rose**)))

The Semantic Frame Structure (37) below is the output of the parser after processing the sentence (36):

((NF^{initial} ((prep **in**) (det **the**) (adj nil) (noun **evening**)))

(NF^{agent} ((prep nil) (det **a**) (adj **(young tall)**) (noun **man**)
(prep **with**) (det nil) (adj **blue**) (noun **eyes**)))

(VF ((tense **past**) (aux1 nil) (aux2 nil) (aux3 nil) (verb **give**))) (37)

(NF^{goal} ((prep nil) (det nil) (adj nil) (noun **Maria**)))

(NF^{theme} ((prep nil) (det **a**) (adj **beautiful**) (noun **book**) (conj **and**)
(prep nil) (det **a**) (adj nil) (noun **rose**))))

Let us consider now the problem of *closure* that arises in two different contexts in the parsing process: the closure of a template and the closure of a frame.

A template is closed after a certain word if:

(a) The next word belongs to a template of another type.
(b) The next word belongs to the same type of template, but corresponds to a template element located to the left of the template element filled last.[39]

Suppose now that a template has just been closed. The closure of a template implies the closure of a frame if:

(a) The next word belongs to a template of another type. (The parser will then start to create a frame of the new type).
(b) The next word is an element of a noun-template[40] but it is not a preposition. (The parser starts to construct a new prepositionless noun-frame).

If these conditions are not satisfied, the parser continues the construction of the corresponding frame using, depending on the next word, one of the operations *Concatenation* or *Conjunction*. The rules above allow the parser to open and close all the frames in the kernel phrase marker (35), except the last one, NF^{final}. The system does not have a syntactic rule which can determine, when the prepositionless noun-frame NF^{theme} ends and NF^{final} begins. What, for example, should happen in the sentence: "Ivan saw Maria with the binoculars"? One possible reading of this

39. Remember that a template must be filled out from left to right.

40. We assume here that the previous template was also a noun-template; otherwise, the condition *a.* holds.

sentence, where the parser closes the frame NF^{theme} and opens NF^{final} as soon as the preposition *with* has been encountered, is represented below in the Semantic Frame Structure (38). It suggests that Ivan was watching Maria through his binoculars:

((NF^agent ((prep nil) (det nil) (adj nil) (noun **Ivan**)))
(VF ((tense **past**) (aux1 nil) (aux2 nil) (aux3 nil) (verb **see**)))
(NF^theme ((prep nil) (det nil) (adj nil) (noun **Maria**)))
(NF^final ((prep **with**) (det **the**) (adj nil) (noun **binoculars**)))) (38)

Another reading of the sentence "Ivan saw Maria with the binoculars" is represented by the Semantic Frame Structure (39). In this case the parser uses the operation *Concatenation* to continue the construction of the noun-frame NF^{theme} after finding the preposition *with*. This reading suggests that Maria had the binoculars at the time when Ivan saw her:

((NF^agent ((prep nil) (det nil) (adj nil) (noun **Ivan**)))
(VF ((tense **past**) (aux1 nil) (aux2 nil) (aux3 nil) (verb **see**))) (39)
(NF^theme ((prep nil) (det nil) (adj nil) (noun **Maria**)
(prep **with**) (det **the**) (adj nil) (noun **binoculars**))))

The problem of attaching the last prepositional phrase (PP^{final} Attachment) requires the use of semantic/syntactic interaction for its resolution [Marcus 1979] and will not be discussed here.[41]

The parser described here is restricted to processing the kernel sentences of the general form (35). However, more complicated sentences with several clauses can also be analyzed by the parser if a mechanism that splits the sentence into kernels is provided. As an example, consider the sentences with embedded clauses generated by the Language Generation Procedure with the help of the *Connective Transformations* (see Table 1 in Section 6). The use of simple heuristic procedures (for example, counting the number of verbs and noun phrases, or searching for certain values of the complementizers COMP and INFL) appears to be sufficient for reconstructing the kernels (matrix clause and embedded clause) which form every complex sentence in Table 1. For instance, the

41. At this point, the parser makes the decision based on a list of prepositions which "usually" begin a new noun-frame (i.e. *through*).

sentence "Ivan claims to have written that letter" consists of two clauses: "Ivan claims it" and "Ivan has written that letter", which have been combined by the transformation 0-0-TO. The possibility to reconstruct the kernels allows the parser to process any sentence generated by the *Connective Transformations.*

Suppose that the input to the parser consists of several connected sentences (a coherent text), and that a pronominalization procedure had been employed by the writer in order to make the text more fluent. If this procedure had been accomplished obeying the *Pronominalization Rule* stated in Appendix 1, the parser can easily *resolve the anaphor*, i.e. restore the noun phrases which were replaced by the pronouns.

This parser has been used as a front end for Winston's learning system [Winston 1980]. It translates English descriptions of situations into descriptions in the *extensible-relation* representation. This representation was suggested by Winston and is implemented in his learning system using a version of Frame Representation Language [Roberts and Goldstein 1977]. In FRL, an *agent-act-theme* combination is expressed as a *frame*, a *slot* in the frame, and a *value* in the slot. In the extensible-relation representation, a supplementary description node for an agent-act-theme combination is expressed as a *comment frame* attached to the frame-slot-value combination.

Suppose, for example, that the following English text is the input to the parser:

> In the beginning of the story Duncan was a king. Macbeth was a happy noble. He married Lady-Macbeth. She was a greedy and ambitious woman. She wanted Macbeth to be king. He also desired to be the king. Lady-Macbeth persuaded him to murder Duncan. Soon Lady-Macbeth decided to kill herself. Macduff was a loyal noble. He became angry. Macbeth's murder of Duncan caused him to kill Macbeth.

Below follow the frames which were generated after parsing the input text. Here, AKO stands for A-KIND-OF, and HP -- for HAS-PROPERTY relations.

```
(DUNCAN        (ako (KING)))

(MACBETH       (ako (NOBLE) (KING (ako-1)))
               (hp (HAPPY))
               (marry (LADY-MACBETH))
               (desire (ako-1))
               (murder (DUNCAN (murder-1))))
```

```
(LADY-MACBETH  (ako (WOMAN))
               (hp (GREEDY) (AMBITIOUS))
               (want (ako-1))
               (persuade (murder-1))
               (kill (LADY-MACBETH (kill-1)))
               (decide (kill-1)))

(MACDUFF       (ako (NOBLE))
               (hp (LOYAL) (ANGRY))
               (kill (MACBETH (kill-2))))

(AKO-1         (frame (MACBETH))
               (slot (AKO))
               (value (KING)))

(MURDER-1      (frame (MACBETH))
               (slot (MURDER))
               (value (DUNCAN))
               (cause (KILL-2)))

(KILL-1        (frame (LADY-MACBETH))
               (slot (KILL))
               (value (LADY-MACBETH)))

(KILL-2        (frame (MACDUFF))
               (slot (KILL))
               (value (MACBETH)))
```

ACKNOWLEDGMENTS

I am deeply indebted to Beth Levin and Mitch Marcus for numerous discussions about this work and friendly help. I also wish to thank Bob Berwick, Mike Brady, Jane Grimshaw, Bill Martin, Dave McDonald, Candy Sidner, and Patrick Winston who read the draft of the paper and made many valuable suggestions.

REFERENCES

Akmajian, A. and Heny, F., *An Introduction to the Principles of Transformational Syntax*, MIT Press, 1975.

Chomsky, N., *A Theory of Syntactic Structures*, Mouton & Co., 1957.

Katz, B., *A Verse-Writing Program*, Avtomatika i Telemekhanika 2, 151-156, 1978.

Lasnik, H., *Remarks on Co-reference*, Linguistic Analysis 2, 1-22, 1976.

Marcus, M. P., *A Theory of Syntactic Recognition for Natural Language*, MIT Press, 1979.

McDonald, D. D., *Natural Language Production as a Process of Decision-making Under Constraints*, MIT PhD Thesis, 1980.

Minsky, M., *A Framework for Representing Knowledge*, in P. H. Winston (ed.), *The Psychology of Computer Vision*, McGrow-Hill, 1975.

Reinhart, T., *The Syntactic Domain of Anaphora*, MIT PhD Thesis, 1976.

Roberts, R. B. and Goldstein, I. P., *The FRL Reference Manual*, MIT Artificial Intelligence Laboratory Memo 409, 1977.

Sidner, C. L., *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse*, MIT Artificial Intelligence Laboratory Technical Report 537, 1979.

Winston, P. H., *Learning and Reasoning by Analogy*, Communications of the ACM 23(12), 689-703, 1980.

END

FILMED

9-83

DTIC