AN AUGMENTED TRANSITION NETWORK INTERPRETER

by

Rowland R. Johnson

Memorandum No. ERL-M511

7 March 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN AUGMENTED TRANSITION NETWORK INTERPRETER[†]

Rowland R. Johnson

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California at Berkeley

March 1975

Abstract

This paper discusses the Augmented Transition Network model which is a formalism for natural language analysis. It also explains the use of a system that implements ATNs by interpreting programs written in a language called ATNL.

We consider Transformational Grammars and the difficulties encountered when using them for sentential analysis. The aspects of the ATN model which are designed to overcome these difficulties are discussed. Next the syntax and semantics of ATNL are discussed in detail and several examples are given. Finally we discuss an ATNL program that is the implementation of an ATN.

[†]Research sponsored by Naval Electronic Systems Command Contract N00039-75-C-0034.

I. Introduction

The report herein describes the use and operation of a system which will implement an Augmented Transition Network. A programming language, called ATNL, is presented that can be used to express an ATN. The main purpose of the ATN formalism is to provide a framework that is both suggestive and convenient for developing sentential analysis procedures. Typically a user, such as a linguist, conceives a procedure for sentential analysis based on the ATN formalism. This procedure is then coded, using ATNL, and presented to the system which iterates on 1) reading a sentence; 2) analyzing it by interpreting the ATNL program; and 3) printing the results. Thus, the system, which is written in UTEX LISP1.9, consists mainly of an interpreter for ATNL programs.

Section II presents a general theory of ATN's which is covered in Woods [1]. Transformational grammars, as they relate to ATNs, are discussed. This section is intended mainly for the user who is unfamiliar with ATNs. Section III is a detailed account of the syntax and semantics of ATNL. Several examples of ATNL statements are given to illustrate these aspects. Finally in Section IV an ATNL program is discussed. This program is an implementation of the ATN in Woods [1].

II. Theory of ATNs

Sentential analysis is done for a variety of reasons, such as question answering or automatic translation. In general a representation of the meaning of the sentence must somehow be obtained. In addition this representation should be in a canonical form so as to facilitate processing by other components of the system. An ATN does just this, it produces from a sentence the canonical representation of the meaning of that sentence.

The ATN formalism is based on the Transformational Grammar model of Natural Language. The TG model was introduced, by Chomsky [2], as a formalism for describing the generation of sentences. The inception of the TG model represented a large step in the development of linguistic concepts. As an example TGs are able to account for the systematic relationship that exists between the active and passive forms of the same sentence. Theoretically, TGs have the power of a Turing machine. However for all the power and sophistication the TG model had it could not feasibly be used to systematically analyze sentences. The ATN formalism allows one to incorporate TG principles in a procedure for sentential analysis.

Basically a TG $\mathcal{G}$ consists of a base component G and a transformational component τ. Generation of a sentence starts with G producing the so-called "deep structure" which is the canonical representation of the meaning of the sentence. Next τ transforms the deep structure into the "surface structure," part of which is the generated sentence. In general several surface structures can be derived from the same deep structure. The linguistic counterpart of this is the paraphrase concept. For example "The cat ate the mouse" and "The mouse was eaten by the cat" are paraphrases of each other. In the TG model the surface structures of these sentences are derived from the same deep structure. The difference is in the way that τ produces the surface structures.

Specifically a TG $\mathcal{G}$ is a tuple (G,τ) where G is a context free grammar and $\tau = \{\tau_1,\tau_2,\ldots,\tau_k\}$ is a set of transformations. Each τ_i can be considered to be a partial function whose domain and range are sets of trees.

We make the following notational conventions:

$\tau_i(t) \triangleq$ the tree which results from applying transformation τ_i to the tree t, if τ_i can be applied to t.

$T(G) \triangleq$ set of trees produced by G; i.e. the set of deep structures in $\mathcal{G}$.

$\tau_{i_1} \cdot \tau_{i_2} \cdot \tau_{i_3} \cdots \tau_{i_N}(t) \triangleq \tau_{i_N}(\cdots(\tau_{i_2}(\tau_{i_1}(t)))\cdots)$

$F(t) \triangleq$ string composed of terminals of tree t read from left to right; i.e. the frontier of t.

Formally a sentence is generated by $\mathcal{G}$ as follows:

1) the deep structure $t_0 \in T(G)$ is produced by G.

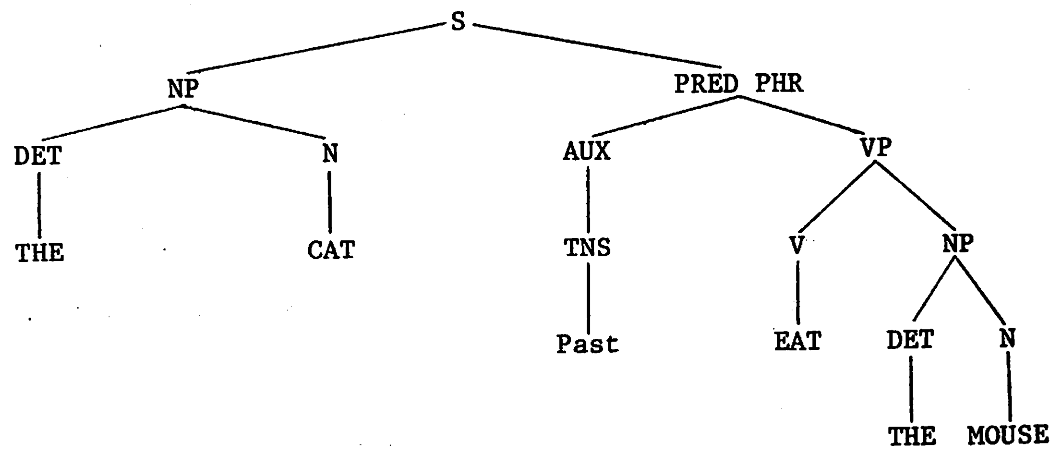
2) the surface structure t_N is obtained by applying the sequence $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_N}$ to t_0; i.e. $t_N = \tau_{i_1} \cdot \tau_{i_2}, \cdots \tau_{i_N}(t_0)$

3) the sentence is just $F(t_N)$

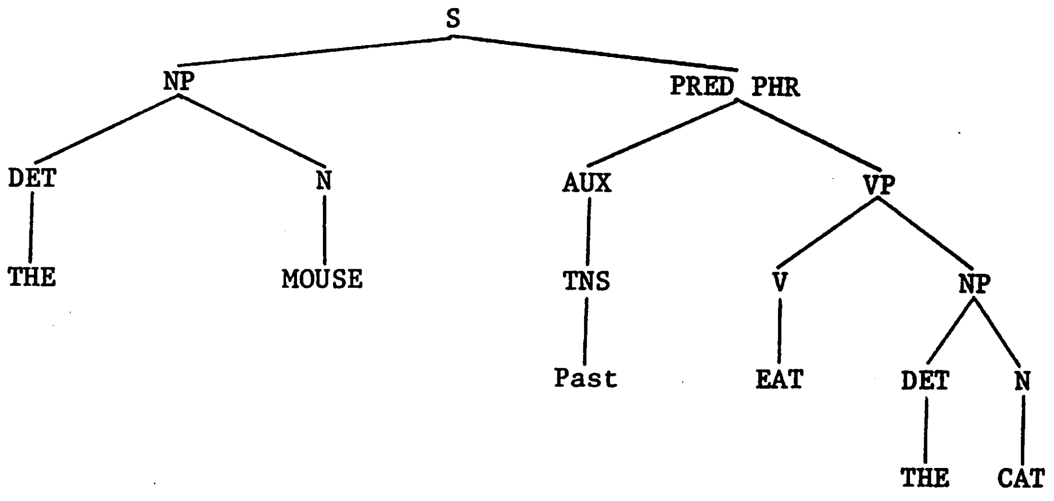
Example 1 Passive Transformation

There are many types of TGs and the example presented here should not be regarded as "the" way to do a passive transformation. Rather this example attempts to convey the general idea of a transformation.

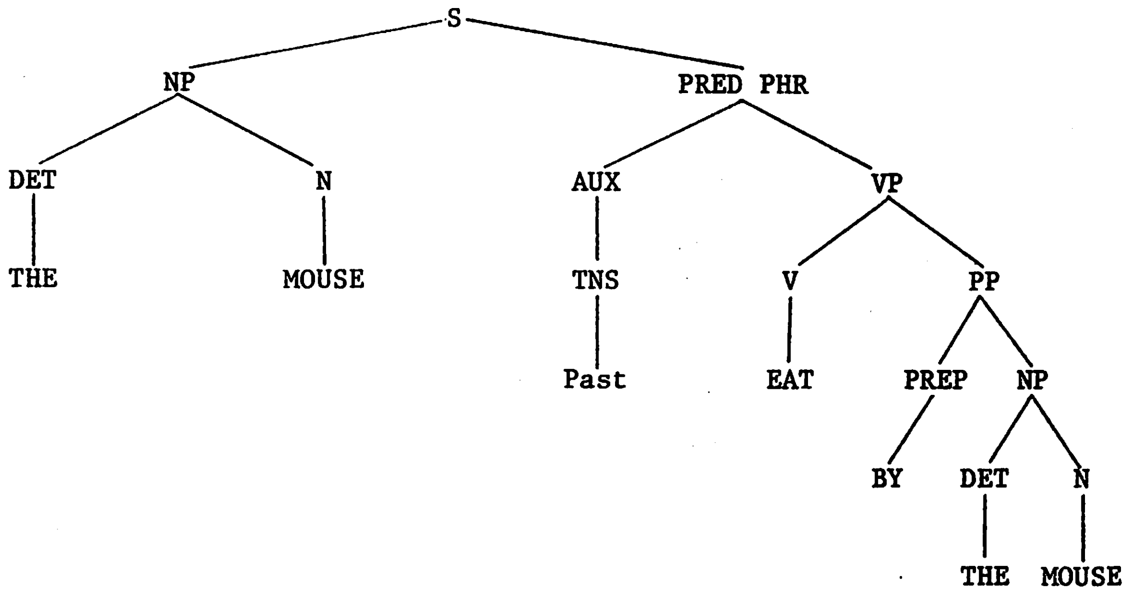
We start with a deep structure that is a representation of the concept "cat ate mouse".



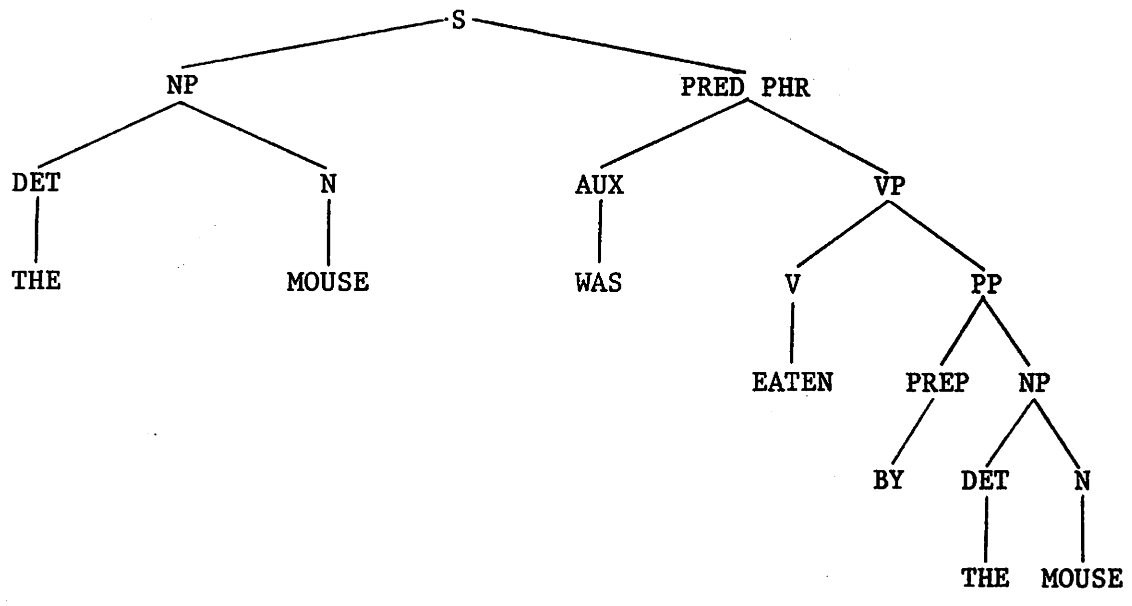
First the subject and object are interchanged



Next "BY" is inserted in front of the original subject



Finally the verb is changed to the passive form

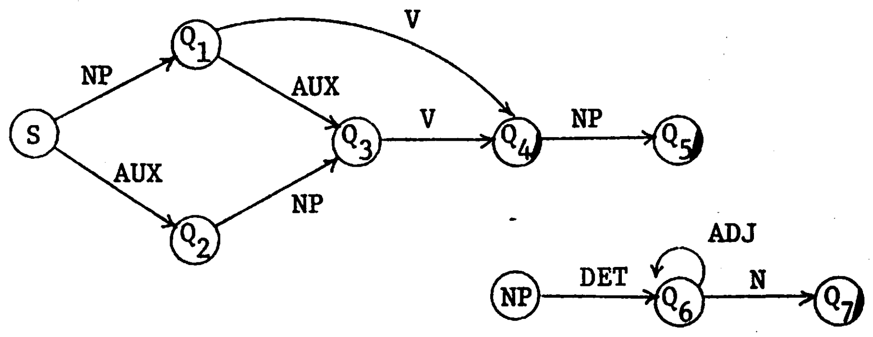


As we can see "The mouse was eaten by the cat" is just the terminal nodes of the tree.

As noted above the TG model does not provide for an analysis procedure. Stated formally, given a sentence S we want to find a $t_0 \varepsilon T(G)$ such that $S = F(\tau_{i_1} \cdot \tau_{i_2} \cdots \tau_{i_N}(t_0))$. There were several attempts at sentential analysis prior to the ATN model that were based on the TG model. Generally they fall into two classes, the first of which is loosely termed "analysis by synthesis." This method attempts to guess a deep structure and then apply sequences of transformations until one is found that yields the sentence in question. Although many heuristics could be employed the resulting procedures were grossly inefficient and tended to be ad hoc. The second method attempted to determine an inverse sequence of transformations which would result in the deep structure. That is, for a given sentence S ; 1) find a surface structure t_N such that $S = F(t_N)$ and 2) find an inverse sequence $\tau'_{i_N}, \tau'_{i_{N-1}}, \ldots, \tau'_{i_1}$ such that $\tau'_{i_N} \cdot \tau'_{i_{N-1}} \cdots \tau'_{i_1}(t_0) = t_0 \varepsilon T(G)$. Step 1) is not a trivial matter since the surface structures are themselves characterized by the generation process. Even if a practical solution to step 1) were developed, step 2) presents a more serious obstacle. The problem here is that for a given tree usually only one transformation will apply. However there may be several inverse transformations that can be applied. In general at each step of the search for this sequence we must consider all of the inverse transformations that can be applied. Thus in determining the inverse sequence the amount of resources that are consumed tends to grow exponentially in the number of inverse transformations required.

ATNs are an extension of <u>R</u>ecursive <u>T</u>ransition <u>N</u>etworks which are similar to a Top Down Parser for a context free grammar. It will be convenient and more lucid to present RTNs and then extend them to ATNs. Basically an RTN is a directed graph in which the nodes are called states. A connected set of states along with the arcs incident upon them is called a sub-network. The organization of these sub-networks account for the phrase structure of a language. The easiest way to understand the notion of an RTN and how one operates is by way of an example.

Example 2



"THE RED BARN COLLAPSED". Starting in state S, Q_1 is PUSH-ed onto the PDS and control goes to state NP. "THE" is a determiner and control goes to Q_6. "RED" is an adjective and control goes to Q_6. "BARN" is a noun and control goes to Q_7. Q_7 is a final state so Q_1

is POP-ed off the PDS and control goes to Q_1, etc.

We make the convention that an RTN always has a state S which is the unique start state. There are two types of arcs; lexical category arcs such as aux or det, and recursion arcs such as NP. The lexical category arcs are used to test for words which belong to the specified lexical category. Recursion arcs must be labeled with state names and are used to recursively call sub-networks. As we will see the arc labels are actually predicates that are used to recognize constituents of the sentence. Finally some states are called final states, such as Q_4, Q_5 and Q_7.

The operation of an RTN makes use of a <u>P</u>ush <u>D</u>own Store and a register, called *, that contains individual words of the sentence. The task is to find a path through the network subject to the tests imposed by the arcs. The RTN is started at state S with * containing the first word of the sentence. In general we are in some state Q with several arcs emanating from Q. The arcs are considered one at a time according to the following schema:

1) If the arc label is a lexical category and * contains a word of that category then traverse the arc and place the next word of the sentence in *. The RTN is then said to be in the state at the end of the arc.
2) If the arc is a recursion arc then place the state at the end of the arc on the PDS and transfer control to the state specified by the arc.
3) If Q is a final state and all arcs have been tried then remove the state on top of the PDS and transfer control to it.

As we can see 2) and 3) result in a sequence of actions that are much like invoking a subroutine in an ALGOL-like language. In fact this sub-network calling may be recursive since any state name within the RTN may be used to label a recursion arc.

Each recursive call sets up a new lower level of computation corresponding to a search for a path through the called sub-network. The search for this path may in turn encounter more recursion arcs which causes recursive calls to other sub-networks. This lower level computation terminates by either finding such a path or determining one does not exist. In the former case the recursion arc which called the sub-network is traversed and in the latter it is not. Thus the traversal of a recursion arc occurs when the corresponding embedded phrase is recognized.

Now to find a complete path through the net we must, in general, try many sequences of arc traversals. This involves backtracking which is the process that occurs when all arcs from a non-final state have been tried and none can be traversed. There are two cases; either this state had control transferred to it by a recursive call or by an arc incident upon it. In either case control is transferred to the state that previously had control. This state in turn tries its next arc.

Finally, the RTN terminates when a final state in the S sub-network is reached and * is empty. That is, the string of lexical categories described by the path matches the string of lexical categories obtained from the words in the sentence. In effect the RTN has recognized the S phrase which is just the sentence itself.

An ATN is an RTN with a register space provided for each level of computation. In addition the arcs may now have actions that test and manipulate this register space. These actions may also directly or indirectly control the order in which the arcs are traversed. In general the registers contain linguistic structures that are the results of structure building actions on previously traversed arcs.

The advantages of this extension can be fully appreciated only by inspection of an example, such as the one in the appendix. However, the main advantage is that the deep structure can be created, modified, and kept in the register space while doing the surface structure parsing. Because of this an inverse transformational component is not required. Decisions can be made about the sentence structure which are easily changed. As an example consider the sentence "The mouse was eaten by the cat." Initially we may decide that "The mouse," since it is at the beginning of the sentence, is the subject. However the verb phrase "was eaten by" tells us this is a passive sentence. Accordingly we now know that what we decided was the subject is actually the object. Also we know that what follows the verb will be the subject, instead of the object.

Another advantage of the ATN formalism is that it is more natural and convenient to use. Part of this is because the operation of an ATN is closer to that of a human understanding a sentence. Also the structure building actions on the arcs can be quite powerful and sophisticated as we will see in the next section.

III. Syntax and Semantics of ATNL

The deck setup for using the system may be found in the appendix. There are two things the user must supply in order to use the system. They are the DICTIONARY and the ATNL program itself. As we will see these are both just a set of S-expressions. There are two objectives of this section. One is to define the syntax of these S-expressions and the other is to explain the semantics of ATNL.

DICTIONARY

The DICTIONARY provides information about words in sentences to be analyzed. It should be noted that the DICTIONARY can contain more than just syntactic information, such as semantic information.

The syntax for the DICTIONARY is given in Fig. 1. As is shown,

```
<DICTIONARY> ::= <DICTENTRY>+
<DICTENTRY> ::= (<Word>(<CATEGORYLIST>)<FEATURELIST>)
<CATEGORYLIST> ::= <Wordcategory>+
<FEATURELIST> ::= {<UNARYFEATURE>|<BINARYFEATURE>}+
<UNARYFEATURE> ::= <FEATURENAME>
<BINARYFEATURE> ::= (<FEATURENAME><FEATUREVALUE>)
<FEATURENAME> ::= lisp atom
<FEATUREVALUE> ::= lisp list
<Wordcategory> ::= lisp atom
<Word> ::= lisp atom
```

Figure 1. Syntax for the DICTIONARY

two things are specified for each word in the DICTIONARY. <CATEGORYLIST> specifies a set of syntactic categories to which the word can belong, such as V, ADJ, N, etc. <FEATURELIST> specifies a set of features that the word can have. As an example consider the dictionary entry for the word BELIEVED.

```
(BELIEVED (V) PPRT (TENSE PAST) (FORMOF BELIEVE))            (1)
```

Here <CATEGORYLIST> is just (V) and <FEATURELIST> is composed of the <UNARYFEATURE> PPRT followed by the <BINARYFEATURE>s (TENSE PAST) and (FORMOF BELIEVE).

As we will see in the discussion on ATNL semantics these two lists provide information by which the ATN analyzes sentences.

ATNL

The syntax for ATNL is given in Fig. 2. When a state is entered each of its arcs are tried, in order, until a successful one is found.

```
<ATN> ::= <STATE>+
<STATE> ::= (<Statename><ARC>+)
<ARC> ::= ({<TEST>|<ACTION>}+)
<TEST> ::= (CAT<Category>)|<FORM>
<ACTION> ::= (PUSH<Statename>)|(POP<FORM>)|
             (SETR<Register><FORM>)|(TO<Statename>)|
             (SENDR<Register><FORM>)|(TRACEON)|
             (LIFTR<Register><FORM>)|(TRACEOFF)|
             (JUMP<Statename>)|
             <Any Lisp S-expression>
<FORM> ::= *|(GETR<Register>)|
             (GETF<FEATURENAME><WordSpec>)|
             (BUILDQ<TreePattern><Register>)|
             <Any Lisp S-expression>
<Statename> ::= lisp atom
<Register> ::= lisp atom
<Category> ::= lisp atom
<WordSpec> ::=      }
                    } see text
<TreePattern> ::=   }
```

Figure 2. Syntax for ATNL

An arc is tried by interpreting, in order, the <ACTION>s and <TEST>s on the arc. Each <ACTION> and <TEST> returns a value of either T or NIL. If an <ACTION> or <TEST> returns the value NIL the arc fails and the next arc is tried. An arc is successful when all of its <ACTION>s and <TEST>s have been interpreted and have returned the value T.

The semantics of the individual <ACTION>s and <TEST>s are as follows:

(CAT<Category>) -- This function looks in the DICTIONARY for the word that is currently in the * register. When it is found the <CATEGORYLIST> is inspected to see if one of its elements matches <CATEGORY>. If such an element exists T is returned, otherwise NIL is returned.

(PUSH<Statename>) and (POP<FORM>) -- These are the <ACTION>s by which the recursion mechanism is invoked. PUSH suspends interpretation at the present level and starts interpretation one level down in the state specified by <Statename>. At some point we want to terminate this lower level processing which we do with a POP. When POP is interpreted <FORM> is evaluated and its value is placed in the * register in the next higher level. Processing then resumes with the <ACTION> or <TEST> following PUSH.

(SETR<Register><FORM>) -- <FORM> is evaluated and its value is placed in the register specified by <Register>. The * register cannot be set as it is a special register. However there really isn't any reason to want to change the contents of the * register. If the register specified by <Register> does not exist it is created. This is in fact the way in which registers come to be.

(SENDR<Register><FORM>) -- Same as SETR except the register is set at the next level down.

(LIFTR<Register><FORM>) -- Same as SETR except the register is set at the next level up.

(TO<Statename>) and (JUMP<Statename>) -- These <ACTION>s transfer control to the state specified by <Statename>. When TO is interpreted * is reset to contain the next input word. However with <JUMP> * is not reset.

(TRACEON) and (TRACEOFF) -- These are the debugging facilities that allow the ATNL programmer to monitor state transitions. After TRACEON and before TRACEOFF is interpreted all state transitions are printed. Each TRACEON and TRACEOFF pertains only to the subnetwork it occurs in.

(GETR<Register>) -- Returns the contents of the register specified by <Register>. If such a register does not exist the value returned is NIL.

(GETF<FEATURENAME><WordSpec>) -- This function allows us to obtain features of words from the DICTIONARY. It should be noted that this GETF is different from the one found in Woods. <WordSpec> specifies a word in the DICTIONARY to be looked up. Next the <FEATURELIST> is inspected to see if it has the particular feature specified by <FEATURENAME>. If there is no such feature or there is no such word in the DICTIONARY the value NIL is returned. In the case that <FEATURENAME> matches a <UNARYFEATURE> the value T is returned. Finally if <FEATURENAME> matches a <BINARYFEATURE> then <FEATUREVALUE> is returned. As an example consider the <DICTENTRY> specified by 1).

(GETF PPRT (QUOTE BELIEVED)) yields T.

(GETF TENSE *) yields PAST if the * register contains BELIEVED.

Suppose the V register contains EAT and

(EAT (V) (TENSE PRES) TRANS) is a <DICTENTRY>.

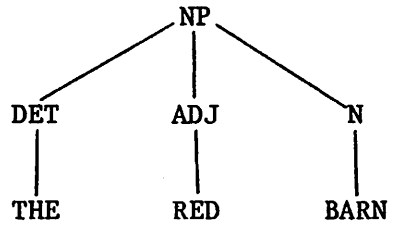
Then (GETF TRANS (GETR V)) would yield T.

Usually we want to obtain features of the current input word. To do this just set <WordSpec> = *. However since <WordSpec> is evaluated we can let it be any Lisp S-expression, <ACTION> or <FORM> as the preceding example shows.

* -- As mentioned earlier this <FORM> specifies a register. Usually it will contain the current input word. However directly after a successful PUSH the * register will contain the result of the lower level computation.

(BUILDQ<TreePattern><Register>*) -- BUILDQ is used to build the tree structures that express the analysis of a sentence. <TreePattern> specifies a tree that may contain special nodes labeled +. The operation of BUILDQ is to substitute the contents of registers (specified by <Register>*) for these nodes marked +. The tree is searched depth-first and the first + encountered is replaced by the first register in <Register>*; the second + by the second register in <Register>*, etc. Consider the following example: Suppose the register contents are

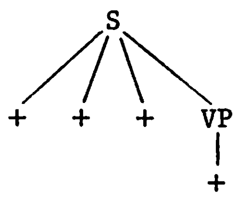
TYPE : Q
AUX : DID
V : COLLAPSE
SUBJ : (NP(DET THE)(ADJ RED)(N BARN)), which is just the tree



We want to evaluate the <FORM>

```
(BUILDQ (S + + +(VP +)) TYPE SUBJ AUX V)
```

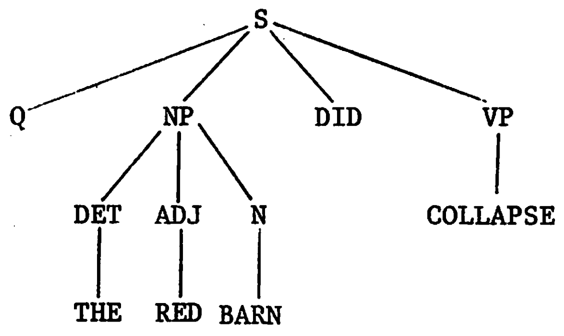
Here we have <TreePattern> = (S + + +(VP +)) which is the tree



The value of this <FORM> then is

```
(S Q (NP(DET THE)(ADJ RED)(N BARN)) DID (VP COLLAPSE))
```

which is just the tree



<Any Lisp S-expression> -- Since this is an interpreter under the LISP system we can have any S-expression for an <ACTION> or <FORM>.

IV. <u>An ATNL Program</u>

The ATN found in section 8 of Woods [1] has been programmed in ATNL. This program and analyses of sentences done by it can be found in the Appendix. In this section we will attempt to point out the differences in the languages and how they can be resolved.

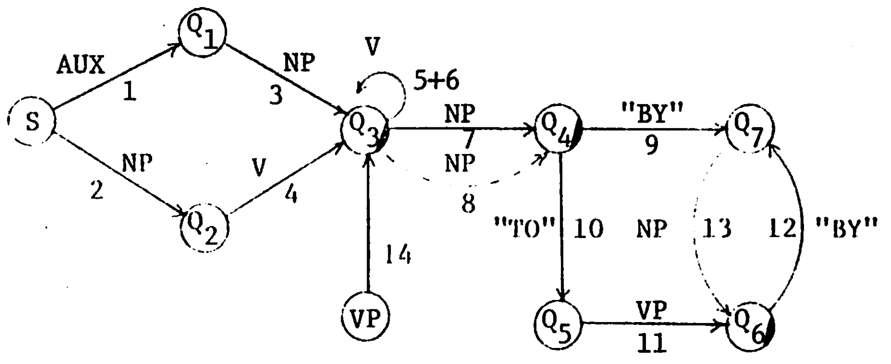


Figure 3. Transition Network from Section 8 of Woods [1]

In our comparison we will consider only a few arcs. For the convenience of the reader Table 1 lists the commands on these arcs.

```
ARC 4 : (CAT V T)
        (SETR V *)
        (SETR TNS (GETF TENSE))
ARC 5 : (CAT V (AND (GETF PPRT)
                (EQ (GETR V)(QUOTE BE))))
        (HOLD (GETR SUBJ))
        (SETR SUBJ (BUILDQ (NP (PRO SOMEONE))))
        (SETR AGFLAG T)
        (SETR V *)
ARC 7 : (PUSH NP (TRANS (GETR V)))
        (SETR OBJ *)
ARC 8 : (VIRTUAL PUSH NP (TRANS (GETR V))
        (SETR OBJ *)
ARC 9 : (AND (EQ * (QUOTE BY))(GETR AGFLAG))
        (SETR AGFLAG NIL)
```

Table 1. Arcs 4, 5, 7, 8 and 9 of Woods' ATN

We can observe several differences by considering arc 4. In ATNL Arc 4 looks like

```
(CAT V)
(SETR V (GETF FORMOF *))
(SETR TNS (GETF TENSE *))
```

The first difference is that CAT takes just one argument. Secondly it would seem that the function SETR in Woods' system does special processing when it comes to syntactic catetory registers. In his example whenever (SETR V *) is interpreted the V register ends up with the root form of the verb. However, in ATNL we must specify (GETF FORMOF *) to get the root form of the word in *. As was mentioned earlier GETF is different than the one in Woods' system. The reason for this is shown by arcs 7 and 9 in which the special functions TRANS and S-TRANS are used. TRANS and S-TRANS are after all just features of words and the number of these special functions might be quite large in a sophisticated ATN. In Woods' system GETF applies only to the word in * but we desire to test features of other words as well. In ATNL this dilemma is resolved by having a second argument for GETF which specifies the word whose feature we are testing.

In Woods' language there is a special register called HOLD. However in ATNL we can just simulate the effect of the HOLD register. On arc 5 we have (SETR HOLD(GETR SUBJ)) as opposed to (HOLD(GETR SUBJ)). In ATNL we have no "virtual" arcs and they must also be simulated. Accordingly on arc 8 we have (NOT(EQ(GETR HOLD)NIL)) to make sure the HOLD register contains something; (SETR OBJ(GETR HOLD)) as opposed to (SETR OBJ *); and finally we have (SETR HOLD NIL) to empty the HOLD register. In Woods' system the HOLD register must be empty in order that a POP be executed. In ATNL we have (NOT(GETR HOLD)) preceding a POP to accomplish this; as in states Q3, Q4, and Q6.

From the differences pointed out above we can see that ATNL is more primitive than the language of Woods' system. However ATNL is just as powerful and it can be argued that it is more flexible. In addition ATNL has debugging facilities to aid the ATNL programmer. Also it should be noted that the two programs are logically the same. That is, the approach from the linguistic point of view is the same.

Summary

The ATN model is a sophisticated and convenient method for sentential analysis. It allows one to use TG principles for sentential analysis in a way that circumvents the serious disadvantages of TGs.

The discussion of ATNs has served as a basis for a detailed account of the use and operation of the system. Finally we have illustrated the use of the system by the implementation of an ATN of moderate size.

Acknowledgment

The author wishes to express his gratitude to his research adviser, Professor Lotfi A. Zadeh, for his guidance. Thanks are also due to Professor M.H. O'Malley and Colin McMaster for their helpful suggestions.

REFERENCES

1. Woods, W. A., "Transition Network Grammars for Natural Language Analysis," Comm. ACM, Vol. 13, No. 10, pp. 591-606.

2. Chomsky, N., Aspects of the Theory of Syntax, MIT Press, Cambridge, Mass., 1965.

APPENDIX

A. DECK SETUP

The deck setup for using the interpreter is shown in Fig. A1. Both the Dictionary and the ATNL program are each a set of S-Expressions

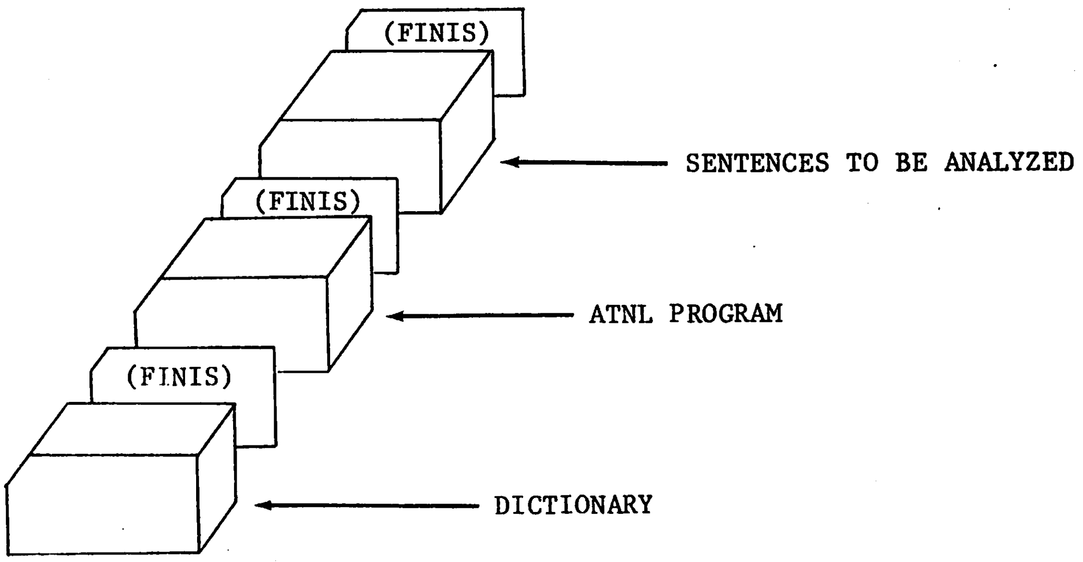


Fig. A1. Deck Setup

as defined in Section III. Comments can occur anywhere within the DICTIONARY and ATNL program and are delimited by the equal sign (=). The sentences to be analyzed are treated as LISP lists and must be enclosed in parentheses.

B. EXAMPLE OF ATNL PROGRAM

The following pages show the output of the system for the ATNL equivalent of Wood's ATN.

```
 (ATE (V) (VOICE ACTIVE)(TENSE PAST)(FORMOF EAT))
 (BARN (N) (N))
  (BEEN (V) PPRT (FORMOF BE))
  (BELIEVE (V) TRANS S-TRANS)
   (BELIEVED (V) PPRT (TENSE PAST)(FORMOF BELIEVE))
  (CAT (N)  () )
  (COLLAPSE (V) (TENSE PRES))
 (COLLAPSED (V) (FORMOF COLLAPSE)(TENSE PAST))
  (DID (AUX)  ())
   (EAT (V) (TENSE PRES)TRANS)
 (EATEN (V) (VOICE PASSIVE)(FORMOF EAT)(TENSE PAST))
  (HAVE (V) PPRT UNTENSED)
  (JOHN (NPR) NIL)
   (MARY (NPR) NIL)
  (MOUSE (N) ())
 (RED (ADJ) ())
   (SHOOT (V) TRANS)
  (SHOT (V) PPRT (FORMOF SHOOT))
 (THE (DET) () )
  (TO (PREP) NIL)
  (WAS (V)(TENSE PAST) (FORMOF BE))
    (FINIS)
STATES

  (S
      ( (TRACEON) )
      =  A R C  1
        IF THERE IS AN AUXILLARY THEN PUT IT IN THE AUX REGISTER AND
        SET SENTENCE TYPE TO QUESTION. =
      ( (CAT AUX)
        (SETR V *)
        (SETR TNS (GETF TENSE *))
        (SETR TYPE (QUOTE Q))
        (TO Q1))

      =   A R C  2
        IF THERE IS A NOUN PHRASE THEN PUT IT IN THE SUBJECT
        REGISTER AND SET SENTENCE TYPE TO DECLARATIVE. =
      ( (PUSH NP)
        (SETR SUBJ *)
        (SETR TYPE (QUOTE DCL))
        (JUMP Q2))     )


  (Q1
      =    A R C  3
        LOOK FOR A NOUN PHRASE HERE AND TENTATIVELY ASSIGN IT AS THE
        SUBJECT OF THE SENTENCE. =
      ( (PUSH NP)
        (SETR SUBJ *)
        (JUMP Q3))      )


  (Q2
      =  A R C  4
        THE VERB FOUND HERE IS ASSUMED TO BE THE MAIN VERB OF THE
      SENTENCE. =
    ( (CAT V)
      (SETR V (GETF FORMOF *))
      (SETR TNS (GETF TENSE *))
      (TO Q3)))


(Q3
    =     A R C  5
      * CONTAINS A VERB THAT IS A PAST PARTICIPLE. THIS ALONG
      THE FACT THAT THE TENTATIVE VERB IS ≠BE≠ INDICATES THE
      PASSIVE CONSTRUCTION. ACCORDINGLY THE OLD TENTATIVE SUBJECT
      IS PUT IN THE HOLD REGISTER AND THE SUBJECT IS RESET TO THE
      INDEFINITE ≠SOMEONE≠. ALSO THE AGFLAG IS SET WHICH INDICATES
      THAT A SUBSEQUENT AGENT INTRODUCED BY THE PREPOSITION ≠BY≠
      (AS ON ARC 9) MAY SPECIFY THE SUBJECT. =
    ( (CAT V)
      (AND (GETF PPRT *)(EQ (GETR V)(QUOTE BE)))
      (SETR HOLD (GETR SUBJ))
      (SETR SUBJ (BUILDQ (NP (PRO SOMEONE))))
      (SETR AGFLAG T)
      (SETR V (GETF FORMOF *))
      (TO Q3))

    =     A R C  6
      * CONTAINS A VERB THAT IS A PAST PARTICIPLE. THIS ALONG WITH
      THE FACT THAT THE TENTATIVE VERB IS ≠HAVE≠ INDICATES THE
      PERFECT TENSE. ACCORDINGLY THE TNS REGISTER IS SET AND THE
      VERB IS ASSUMED TO BE THE WORD IN *. =
    ( (CAT V)
      (AND (GETF PPRT *)(EQ (GETR V) (QUOTE HAVE)))
      (SETR TNS (LIST (GETR TNS)(QUOTE PERFECT)))
      (SETR V (GETF FORMOF *))
      (TO Q3))

    =     A R C  7
      THE V REGISTER CONTAINS A TRANSITIVE VERB AND REQUIRES AN
      OBJECT. IF A NOUN PHRASE IS PRESENT THEN IT IS TENTATIVELY
      ASSIGNED AS THE OBJECT OF THE VERB. =
    ( (GETF TRANS (GETR V))
      (PUSH NP)
      (SETR OBJ *)
      (JUMP Q4))

    =     A R C  8
      AS ON ARC 7 WE REQUIRE AN OBJECT FOR THE TRANSITIVE VERB. IN
      THIS CASE, HOWEVER, THE HOLD REGISTER IS INSPECTED TO SEE IF
      IT CONTAINS ANYTHING. IF IT DOES THEN THOSE CONTENTS ARE
      TENTATIVELY ASSIGNED AS THE OBJECT OF THE VERB. THE HOLD
      REGISTER IS THEN CLEARED, REFLECTING THE FACT THAT ITS
      CONTENTS HAVE BEEN USED. =
    ( (NOT (EQ (GETR HOLD) NIL))
      (GETF TRANS (GETR V))
      (SETR OBJ (GETR HOLD))
      (SETR HOLD NIL)
      (JUMP Q4))
```

```
= AT THIS POINT ARCS 5,6,7 AND 8 HAVE FAILED. THE ONLY THING
  LEFT TO DO IS TO PERFORM A POP AND RETURN TO THE NEXT HIGHEST
  LEVEL. WE MUST, HOWEVER, BE SURE THE HOLD REGISTER DOES NOT
  CONTAIN ANYTHING. IF IT DID THIS WOULD MEAN A CONSTITUENT CAN
  NOT BE ACCOUNTED FOR. =
( (NOT (GETR HOLD))
  (POP (BUILDQ (S   +     +   (TNS  + )(VP(V + )))
                  TYPE SUBJ      TNS        V       ))))


(Q4
  =    A R C  9
  ≠BY≠ INDICATES AN AGENT WILL PROBABLY FOLLOW. THIS AGENT WILL
  BE IN THE FORM OF A NOUN PHRASE WHICH WILL BE ANALYZED BY A
  RECURSIVE CALL ON ARC 13. FIRST WE CHECK AGFLAG TO SEE IF AN
  AGENT IS NEEDED BY A STRUCTURE. =
( (EQ * (QUOTE BY))
  (GETR AGFLAG)
  (SETR AGFLAG NIL)
  (TO Q7))

  =  A R C   1 0
  THE WORD ≠TO≠ TELLS US THAT THE OBJECT OF THE VERB IS A
  NOMINALIZED SENTENCE. THIS EMBEDDED CLAUSE WILL BE
  ANALYZED BY A RECURSIVE CALL ON ARC 11. WHAT WAS PREVIOUSLY
  THOUGHT TO BE THE OBJECT IS REALLY THE SUBJECT OF THE
  NOMINALIZED SENTENCE. ACCORDINGLY THE SUBJ REGISTER OF THE
  LOWER LEVEL COMPUTATION IS SET (BY USING SENDR) TO CONTAIN
  THE OLD OBJECT. LIKEWISE THE TNS AND TYPE REGISTERS OF THE
  LOWER LEVEL COMPUTATION ARE SET.  =
( (EQ * (QUOTE TO))
  (GETF S-TRANS (GETR V))
  (SENDR SUBJ (GETR OBJ))
  (SENDR TNS (GETR TNS))
  (SENDR TYPE (QUOTE DCL))
  (TO Q5))

  = ARCS 9 AND 10 HAVE FAILED SO A POP IS PERFORMED AFTER CHECKING
  THE HOLD REGISTER, AS IN STATE Q3. =
( (NOT (GETR HOLD))
  (POP (BUILDQ (S   +     +   (TNS  + )(VP(V + )   +   ))
                  TYPE SUBJ      TNS        V    OBJ    ))))


(Q5
  =   A R C 1 1
  IF A VERB PHRASE IS FOUND IT IS ASSUMED TO BE THE OBJECT. =
( (PUSH VP)
  (SETR OBJ *)
  (JUMP Q6)))


(Q6
  =    A R C  1 2
  ≠BY≠ INTRODUCES THE SUBJECT OF THE ACTION WHICH WILL BE
```

```
      ANALYZED BY ARC ARC 13. =
    ( (EQ * (QUOTE BY))
      (TO Q7))

    ( (NOT (GETR HOLD))
      (POP (BUILDQ (S   +     +   (TNS   + )(VP(V + )   +   ))
                        TYPE SUBJ      TNS          V    OBJ    ))))


 (Q7
    =   A R C  1 3
      WE ARRIVE AT THIS STATE LOOKING FOR A NOUN PHRASE TO BE THE
      SUBJECT. ACCORDINGLY A PUSH TO NP IS PERFORMED AND THE RESULTS,
      IF THE PUSH WAS SUCCESSFUL, ARE PUT IN THE SUBJ REGISTER. =
    ( (PUSH NP)
      (SETR SUBJ *)
      (JUMP Q6)))


 (VP
    ( (TRACEON) )
    =  A R C  1 4
      THIS IS THE INITIAL STATE OF THE VERB PHRASE SUBNETWORK. THIS
      SUBNETWORK ACCEPTS ONLY VERB PHRASES THAT BEGIN WITH A
      STANDARD UNTENSED FORM OF A VERB. =
    ( (CAT V)
      (GETF UNTENSED *)
      (SETR V *)
      (TO Q3)))


 (NP
    ( (TRACEON) )
    = THIS IS THE INITIAL STATE OF THE NOUN PHRASE SUBNETWORK. TWO
      TYPES OF NOUN PHRASES ARE CONSIDERED, THOSE THAT BEGIN WITH A
      DETERMINER AND THOSE THAT CONSIST OF A PROPER NOUN. =
    ( (CAT DET)
      (SETR DET *)
      (TO Q8))

    ( (CAT NPR)
      (SETR NPR *)
      (TO Q10)))


 (Q8
    = A DETERMINER IS FOLLOWED BY THE NOUN IT DETERMINES. HOWEVER
      THERE MAY BE AN INTERVENING ADJECTIVE TO MODIFY THE NOUN. IF
      SUCH AN ADJECTIVE IS PRESENT IT IS PROCESSED BEFORE THE NOUN
      IS PROCESSED. =
    ( (CAT ADJ)
      (SETR ADJ *)
      (TO Q8) )

    ( (CAT N)
      (SETR N *)
```

```
        (TO Q9) ) )


   (Q9
       = RETURN THE RESULTS VIA POP. =
        (POP (BUILDQ (NP(DET + )(ADJ + )(N +))
                             DET     ADJ    N   ))))


   (Q10
       = THIS NOUN PHRASE IS JUST A PROPER NOUN AND THE RESULTS ARE
        RETURNED VIA POP. =
       ( (POP (BUILDQ (NP(NPR  + ))
                              NPR     ))))
   (FINIS)
INPUT SENTENCE
(JOHN WAS BELIEVED TO HAVE BEEN SHOT)

(TRACE ON IN STATE S)
WITH REGS
(* JOHN)

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(* JOHN)

(TRACE ON IN STATE NP)
WITH REGS
(* JOHN)


(LEAVING STATE NP AND ENTERING STATE Q10)
WITH REGS
(NPR . JOHN)
(* WAS)

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(* (NP (NPR JOHN)))

(LEAVING STATE S AND ENTERING STATE Q2)
WITH REGS
(SUBJ NP (NPR JOHN))
(TYPE . DCL)
(* WAS)


(LEAVING STATE Q2 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (NPR JOHN))
(TYPE . DCL)
(V . BE)
(TNS . PAST)
(* BELIEVED)
```

```
(LEAVING STATE Q3 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* TO)

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* TO)

(TRACE ON IN STATE NP)
WITH REGS
(* TO)

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* TO)

(LEAVING STATE Q3 AND ENTERING STATE Q4)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* TO)

(LEAVING STATE Q4 AND ENTERING STATE Q5)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* HAVE)
```

```
(PUSHING TO  VP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* HAVE)

(TRACE ON  IN STATE VP)
WITH REGS
(SUBJ NP (NPR JOHN))
(TNS . PAST)
(TYPE . DCL)
(* HAVE)


(LEAVING STATE VP AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (NPR JOHN))
(TNS . PAST)
(TYPE . DCL)
(V . HAVE)
(* BEEN)


(LEAVING STATE Q3 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (NPR JOHN))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . BE)
(* SHOT)


(LEAVING STATE Q3 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* (NIL . ))

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
```

```
(* (NIL . ))

(TRACE ON IN STATE NP)
WITH REGS
(* (NIL . ))

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* (NIL . ))

(LEAVING STATE Q3 AND ENTERING STATE Q4)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* (NIL . ))

(RETURNING FROM PUSH TO VP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* (S DCL (NP (PRO SOMEONE)) (TNS (PAST PERFECT)) (VP (V SHOOT) (NP (NPR JOHN)))))

(LEAVING STATE Q5 AND ENTERING STATE Q6)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ S DCL (NP (PRO SOMEONE)) (TNS (PAST PERFECT)) (VP (V SHOOT) (NP (NPR JOHN))))
(* (NIL . ))

RETURNED REG
(S DCL (NP (PRO SOMEONE))(TNS PAST)(VP (V BELIEVE)
  (S DCL (NP (PRO SOMEONE))(TNS (PAST PERFECT))(VP (V SHOOT)(NP (NPR JOHN))))))



INPUT SENTENCE
```

```
(JOHN WAS BELIEVED TO HAVE BEEN SHOT BY MARY)

(TRACE ON IN STATE S)
WITH REGS
(* JOHN)

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(* JOHN)

(TRACE ON IN STATE NP)
WITH REGS
(* JOHN)


(LEAVING STATE NP AND ENTERING STATE Q10)
WITH REGS
(NPR . JOHN)
(* WAS)

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(* (NP (NPR JOHN)))

(LEAVING STATE S AND ENTERING STATE Q2)
WITH REGS
(SUBJ NP (NPR JOHN))
(TYPE . DCL)
(* WAS)


(LEAVING STATE Q2 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (NPR JOHN))
(TYPE . DCL)
(V . BE)
(TNS . PAST)
(* BELIEVED)


(LEAVING STATE Q3 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* TO)

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
```

```
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* TO)

(TRACE ON IN STATE NP)
WITH REGS
(* TO)

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* TO)

(LEAVING STATE Q3 AND ENTERING STATE Q4)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* TO)


(LEAVING STATE Q4 AND ENTERING STATE Q5)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* HAVE)

(PUSHING TO  VP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* HAVE)

(TRACE ON IN STATE VP)
WITH REGS
(SUBJ NP (NPR JOHN))
(TNS . PAST)
```

```
(TYPE . DCL)
(* HAVE)


(LEAVING STATE VP AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (NPR JOHN))
(TNS . PAST)
(TYPE . DCL)
(V . HAVE)
(* BEEN)


(LEAVING STATE Q3 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (NPR JOHN))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . BE)
(* SHOT)


(LEAVING STATE Q3 AND ENTERING STATE Q3)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* BY)

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* BY)

(TRACE ON IN STATE NP)
WITH REGS
(* BY)

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD NP (NPR JOHN))
(AGFLAG . *T*)
(* BY)
```

```
(LEAVING STATE Q3 AND ENTERING STATE Q4)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* BY)


(LEAVING STATE Q4 AND ENTERING STATE Q7)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD)
(AGFLAG)
(OBJ NP (NPR JOHN))
(* MARY)

(PUSHING TO  NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD)
(AGFLAG)
(OBJ NP (NPR JOHN))
(* MARY)

(TRACE ON IN STATE NP)
WITH REGS
(* MARY)


(LEAVING STATE NP AND ENTERING STATE Q10)
WITH REGS
(NPR . MARY)
(* (NIL . ))

(RETURNING FROM PUSH TO NP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD)
(AGFLAG)
(OBJ NP (NPR JOHN))
(* (NP (NPR MARY)))

(LEAVING STATE Q7 AND ENTERING STATE Q6)
```

```
WITH REGS
(SUBJ NP (NPR MARY))
(TNS PAST PERFECT)
(TYPE . DCL)
(V . SHOOT)
(HOLD)
(AGFLAG)
(OBJ NP (NPR JOHN))
(* (NIL . ))

(RETURNING FROM PUSH TO VP SUBNETWORK)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ NP (NPR JOHN))
(* (S DCL (NP (NPR MARY)) (TNS (PAST PERFECT)) (VP (V SHOOT) (NP (NPR JOHN)))))

(LEAVING STATE Q5 AND ENTERING STATE Q6)
WITH REGS
(SUBJ NP (PRO SOMEONE))
(TYPE . DCL)
(V . BELIEVE)
(TNS . PAST)
(HOLD)
(AGFLAG . *T*)
(OBJ S DCL (NP (NPR MARY)) (TNS (PAST PERFECT)) (VP (V SHOOT) (NP (NPR JOHN))))
(* (NIL . ))

RETURNED REG
(S DCL (NP (PRO SOMEONE))(TNS PAST)(VP (V BELIEVE)
  (S DCL (NP (NPR MARY))(TNS (PAST PERFECT))(VP (V SHOOT)(NP (NPR JOHN)))))
```