American Journal of Computational Linguistics *Microfiche 33 : 45*

GENERATION AS PARSING FROM A NETWORK INTO A LINEAR STRING

STUART C. SHAPIRO

Computer Science Department
Indiana University
Bloomington 47401

ABSTRACT

Generation of English surface strings from a semantic network is viewed as the creation of a linear surface string that describes a node of the semantic network. The form of the surface string is controlled by a recursive augmented transition network grammar, which is capable of examining the form and content of the semantic network connected to the semantic node being described. A single node of the grammar network may result in different forms of surface strings depending on the semantic node it is given, and a single semantic node may be described by different surface strings depending on the grammar node it is given to. Since generation from a semantic network rather than from disconnected phrase markers, the surface string may be generated directly, left to right.

Introduction

In this paper, we discuss the approach being taken in the English generation subsystem of a natural language understanding system presently under development at Indiana University. The core of the understander is a semantic network processing system, SNePS (Shapiro, 1975), which is a descendant of the MENTAL semantic subsystem (Shapiro, 1971a, 1971b) of the MIND system (Kay, 1973). The role of the generator is to describe, in English, any of the nodes in the semantic network, all of which represent concepts of the understanding system.

and other computations are required in the process of pasting these trees tog ther in appropriate piaces until a single phrase marker is attained which will lead to the surface string. Since we are generating from a semantic network, all the pasting together is already done. Grabbing the network by the node of interest and letting the network dangle from it gives a structure which may be searched appropriately in order to generate the surface string directly in left to right fashion.

Our system bears a superficial resemblance to that described in Simmons and Slocum, 1972 and in Simmons, 1973. That system, however, stores surface information such as tense and voice in its semantic network and its ATN takes as input a linear list containing the semantic node and a generation pattern consisting of a "series of constraints on the modality" (Simmons et al., 1973, p. 92

The generator described in Schank et al., 1973, translates from a "conceptual structure" into a network of the form of Simmons' network which is then given to a version of Simmons generation program. The two stages use different mechanisms. Our system amounts to a unificatio of these two stages.

The generator, as described in this paper, as well as SNePS, a parser and an inference mechanism have been written in LISP 1.6 and are running interactively on a DEC system-10 on the Indiana University Computing Network.

Representation in the Semantic Network

Conceptual information derived from parsed sentences or deduced from other information (or input directly via the SNePS user's language) is stored in a semantic network. The nodes in the network represent concepts which may be discussed and reasoned about. The edges represent semantic but non-conceptual binary relations between nodes. There are also auxiliary nodes which SNePS can use or which the user can use as SNePS variables. (For a more complete discussion of SNePS and the network see Shapiro, 1975.)

The semantic network representation being used does not include information considered to be features of the surface string such as tense, voice or main vs. relative clause. Instead of tense, temporal information is stored relative to a growing time line in a manner similar to that of Bruce, 1972. From this information a tense can be generated for an output sentence, but it may be a different tense than that of the original input sentence if time has progressed in the interim. The voice of a generated sentence is usually determined by the top level call to the generator function. However, sometimes it is determined by the generator grammar. For example, when generating a relative clause, voice is determined by whether the noun being modified is the agent or object of the action described by the relative clause. The main clause of a generated sentence depends on which semantic node is given to the generator in the top level call. Other nodes connected to it may result in relative clauses being generated. These roles may be reversed in other top level calls to the generator.

The generator is driven by two sets of data: the semantic network and a grammar in the form of a recursive augmented transition network (ATN) similar to that of Woods, 1973. The edges on our ATN are somewhat different from those of Woods since our view is that the generator is a tranducer from a network into a linear string, whereas a parser is a transducer from a linear string into a tree or network. The changes this entails are discussed below. During any point in generation, the generator is working on some particular semantic node. Functions on the edges of the ATN can examine the network connected to this node and fail or succeed accordingly. In this way, nodes of the ATN can "decide" what surface form is most appropriate for describing a semantic node, while different ATN nodes may generate different surface forms to describe the same semantic node.

A common assumption among linguists is that generation begins with a set of disconnected deep phrase markers. Transformations

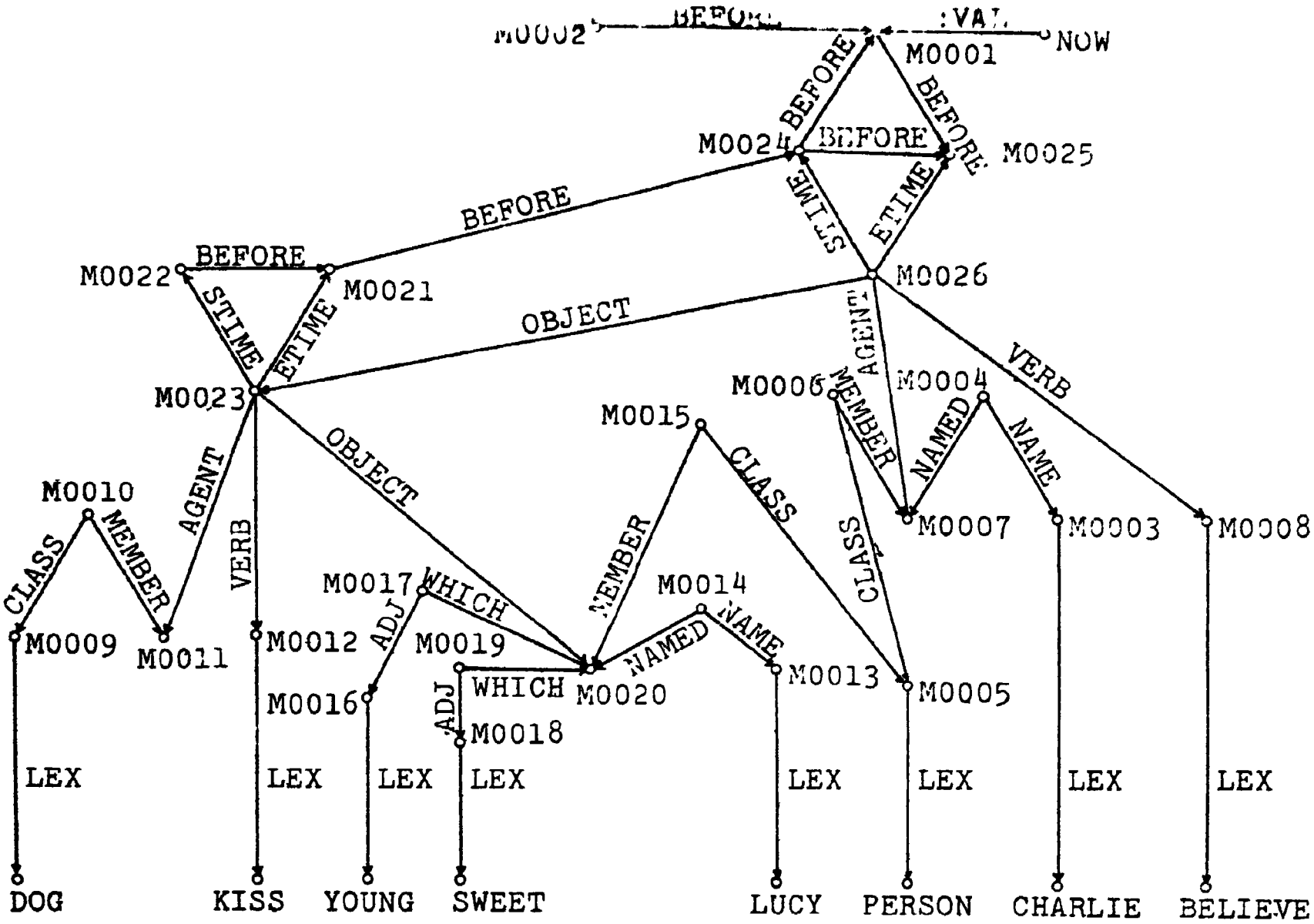

Figure 1: Semantic Network Representation for "Charlie believes that a dog kissed sweet young Lucy," "Charlie is a person," and "Lucy is a person."

Information considered to be features of surface strings are not stored in the semantic network, but are used by the parser in constructing the network from the input sentence and by the generator for generating a surface string from the network. For example, tense is mapped into and from temporal relations between a node representing that some action has, is, or will occur and a growing time line. Restrictive relative clauses are used by the parser to identify a node being discussed, while non-restrictive relative clauses may result in new information being added to the network.

The example used in this paper is designed to illustrate the generation issues being discussed. Although it also illustrates our general approach to representational issues, some details will

```
*(SNEG M0026)
(CHARLIE IS BELIEVING THAT A DOG KISSED SWEET YOUNG LUCY)
*(SNEG M0023)
(A DOG KISSED SWEET YOUNG LUCY)
*(SNEG M0007)
(CHARLIE WHO IS BELIEVING THAT A DOG KISSED SWEET YOUNG LUCY)
*(SNEG M0004)
(CHARLIE IS A PERSON WHO IS BELIEVING THAT A DOG KISSED SWEET YOUNG LUCY)
*(SNEG M0006)
(CHARLIE WHO IS BELIEVING THAT A DOG KISSED SWEET YOUNG LUCY IS A PERSON)
*(SNEG M0008)
(THE BELIEVING THAT A DOG KISSED SWEET YOUNG LUCY BY CHARLIE)
*(SNEG M0011)
(A DOG WHICH KISSED SWEET YOUNG LUCY)
*(SNEG M0010)
(THAT WHICH KISSED SWEET YOUNG LUCY IS A DOG)
*(SNEG M0012)
(THE KISSING OF SWEET YOUNG LUCY BY A DOG)
*(SNEG M0020)
(SWEET YOUNG LUCY WHO WAS KISSED BY A DOG)
*(SNEG M0014)
(LUCY IS A SWEET YOUNG PERSON WHO WAS KISSED BY A DOG)
*(SNEG M0015)
(SWEET YOUNG LUCY WHO WAS KISSED BY A DOG IS A PERSON)
*(SNEG M0017)
(SWEET LUCY WHO WAS KISSED BY A DOG IS YOUNG)
*(SNEG M0019)
(YOUNG LUCY WHO WAS KISSED BY A DOG IS SWEET)
```

Figure 2: Results of calls to the generator with nodes from Figure 1. User input is on lines beginning with *.

certainly change as work progresses. Figure 1 shows the semantic network representation for the information in the sentences, "Charlie believes that a dog kissed sweet young Lucy," "Charlie is a person," and "Lucy is a person." Converse edges are not shown, but in all cases the label of a converse edge is the label of the forward edge with '*' appended except for BEFORE, whose converse edge is labelled AFTER. LEX pointers point to nodes containing lexical entries. STIME points to the starting time of an action and ETIME to its ending time. Nodes representing instants of time are related to each other by the BEFORE/AFTER edges. The auxiliary node NOW has a :VAL pointer to the current instant of time.

Figure 2 shows the generator's output for many of the nodes of Figure 1. Figure 3 shows the lexicon used in the example.

```
(BELIEVE((CTGY.V)(INF.BELIEVE)
    (PRES.BELIEVES)(PAST.BELIEVED)(PASTP.BELIEVED)(PRESP.BELIEVING)))
(CHARLIE((CTGY.NPR)(PI.CHARLIE)))
(DOG((CTGY.N)(SING.DOG)(PLUR.DOGS)))
(KISS((CTGY.V)(INF.KISS)
    (PRES.KISSES)(PAST.KISSED)(PASTP.KISSED)(PRESP.KISSING)))
(LUCY((CTGY.NPR)(PI.LUCY)))
(PERSON((CTGY.N)(SING.PERSON)(PLUR.PEOPLE)))
(SWEET((CTGY.ADJ)(PI.SWEET)))
(YOUNG((CTGY.ADJ)(PI.YOUNG)))
```

Figure 3: The lexicon used in the example of Figures 1 and 2.

Generation as Parsing

Normal parsing involves taking input from a linear string and producing a tree or network structure as output. Viewing this in terms of an ATN grammar as described in Woods, 1973, there is a well-defined next input function which simply places successive words into the * register. The output function, however, is more complicated, using BUILDQ to build pieces of trees, or, as in our parser, a BUILD function to build pieces of network.

If we now consider generating in these terms, we see that there is no simple next input function. The generator will focus on some semantic node for a while, recursively shifting its attention to adjacent nodes and back. Since there are several adjacent nodes, connected by variously labelled edges, the grammar author must specify which edge to follow when the generator is to move to another semantic node. For these reasons, the same focal semantic node is used when traversing edges of the grammar network and a new semantic node is specified by giving a path from the current semantic node when pushing to a new grammar node. The register SNODE is used to hold the current semantic node.

The output function of generation is straightforward, simply being concatenation onto a growing string. Since the output string is analogous to the parser's input string, we store it in the reg-

```
garc ::= (TEST test [action]*(TO gnode))
         (JUMP [action]*(TO gnode))
         (MEM wform (word*) test [action]*(TO gnode))
         (NOTMEM wform (word*) test [action]*(TO gnode))
         (TRANSR ([regname] regname regname) test [action]*(TO gnode))
         (GEN gnode sform [action]*regname [action]*(TO gnode))

sform ::= wform
          SNODE

wform ::= (CONCAT form form*)
          (GETF sarc [sform])
          (GETR regname)
          (LEXLOOK lfeat [sform])
          sexp

form ::= wform
         sform

action ::= (SETR regname form
           (ADDTO regname form*)
           (ADDON regname form*)
           sexp

test ::= (MEMS form form)
         (PATH sform sarc* sform)
         form
         sexp

gnode ::= <any LISP atom which represents a grammar node>
word ::= <any LISP atom>
regname ::= <any non-numeric LISP atom used as a register name>
sarc ::= <any LISP atom used as a semantic arc label>
lfeat ::= <any LISP atom used as a lexical feature>
sexp ::= <any LISP s-expression>
```

Figure 4: Syntax of edges of generator ATN grammars

ister *. When a pop occurs, it is always the current value of * that is returned.

Figure 4 shows the syntax of the generator ATN grammar. Object language symbols are), (, and elements in capital letters. Meta-language symbols are in lower case, Square brackets enclose optional elements. Elements followed by * may be repeated one or more times. Angle brackets enclose informal English descriptions.

Semantics of Edge Functions

In this section, the semantics of the grammar arcs, forms and tests are presented and compared to those of Woods' ATNs.† The

† All comparisons are with Woods, 1973.

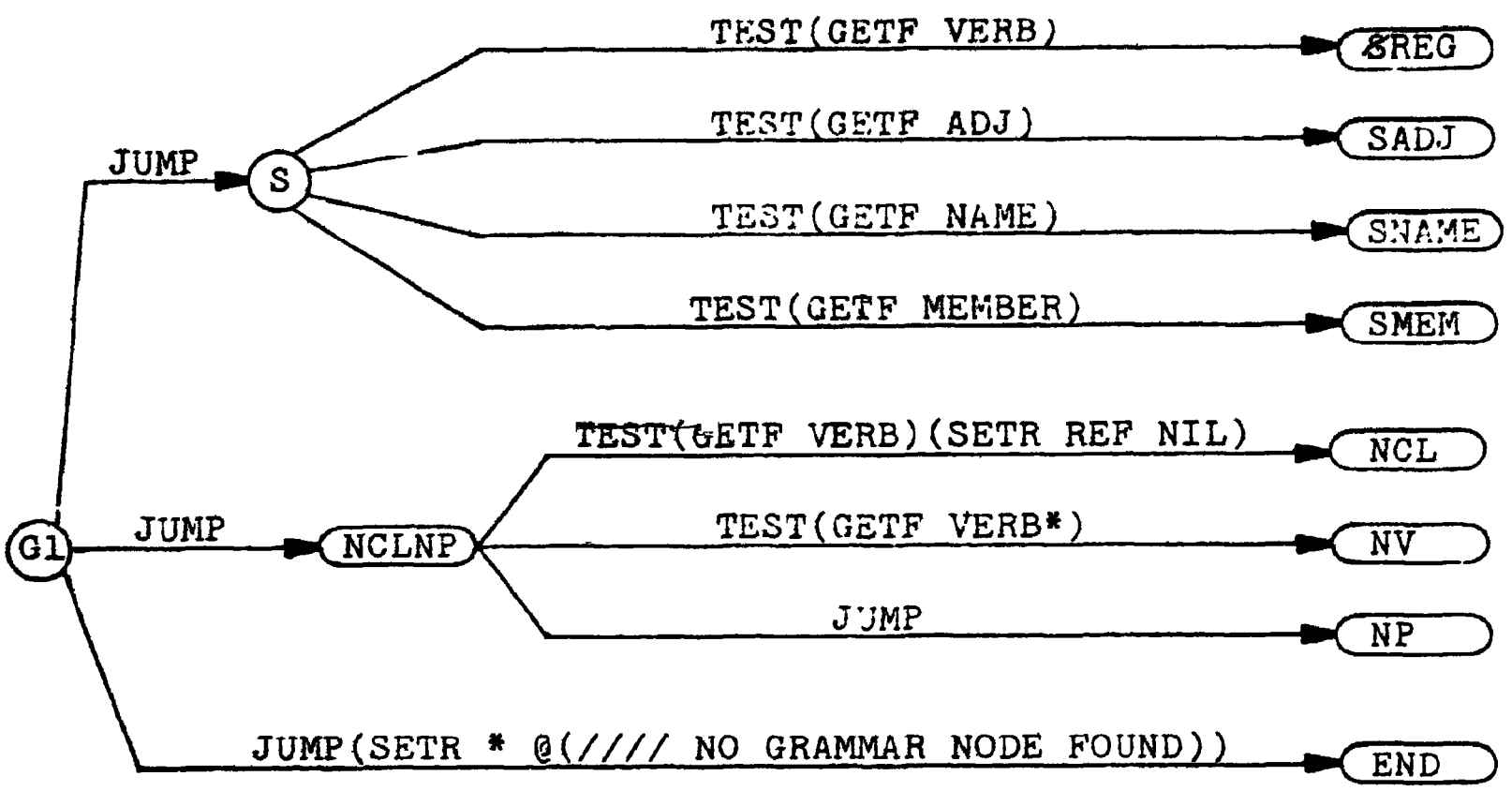

Figure 5: The default entry into the grammar network.

essential differences are those required by the differences between generating and parsing as discussed in the previous section.

(TEST test [action]*(TO gnode))

If the test is successful (evaluates to non-NIL), the actions are performed and generation continues at gnode. If the test fails, this edge is not taken. TEST is the same as Woods' TST, while TEST(GETF sarc) is analogous to Woods' CAT.

(JUMP [action]*(TO gnode))

Equivalent to (TEST T [action]*(TO gnode)). JUMP is similar in use to Woods' JUMP, but the difference from TEST T disappears since no edge "consumes" anything.

(MEM wform (word*) test [action]*(TO gnode))

If the value of wform has a non-null intersection with the list of words, the test is performed. If the test is also successful the actions are performed and generation continues at gnode. if either the intersection is null or the test fails, the edge

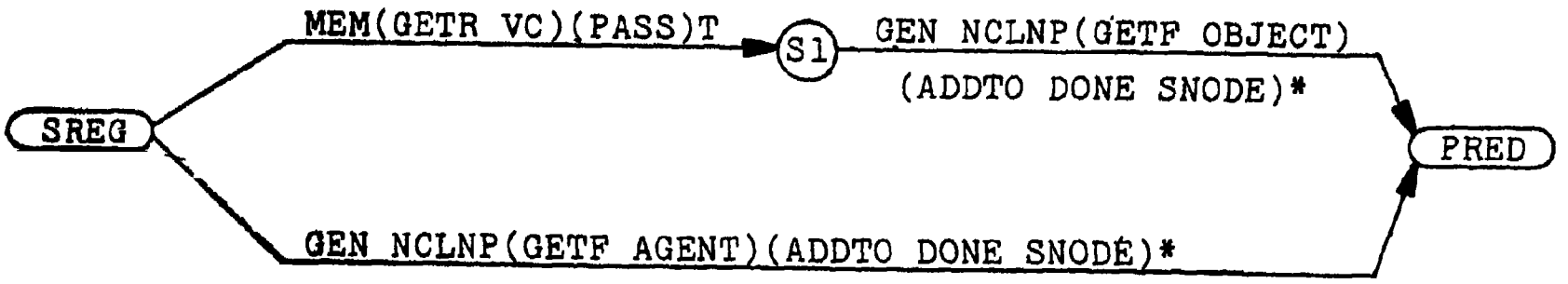

Figure 6: Generation of subject of subject-verb-object sentence.

is not taken. This is similar in form to Woods' MEM, but mainly used for testing registers.

(NOTMEM wform (word*) test [action]*(TO gnode))

This is exactly like MEM except the intersection must be null.

(TRANSR ([regname$_1$] regname$_2$ regname$_3$) test [action]*(TO gnode))

If regname$_1$ is present, the contents of regname$_2$ are added on the end of regname$_1$. If regname$_3$ is empty, the edge is not taken. Otherwise, the first element in regname$_3$ is removed and placed in regname$_2$ and the test is performed. If the test fails, the edge is not taken, but if it succeeds, the actions are performed and generation continues at gnode. TRANSR is used to iterate through several nodes all in the same semantic relation with the main semantic node.

(GEN gnode$_1$ sform [action]*regname [action]*(TO gnode$_2$))

The first set of actions are performed and the generation is called recursively with the semantic node that is the value of sform and at the grammar node gnode$_1$. If this generation is successful (returns non-NIL), the result is placed in the register regname, the second set of actions are performed and generation continues at gnode$_2$. If the generation fails, the edge is not taken. This is the same as Woods' PUSH but requires a semantic node to be specified and allows any register to be used to hold the result. Instead of having a POP edge, a return automatically occurs when

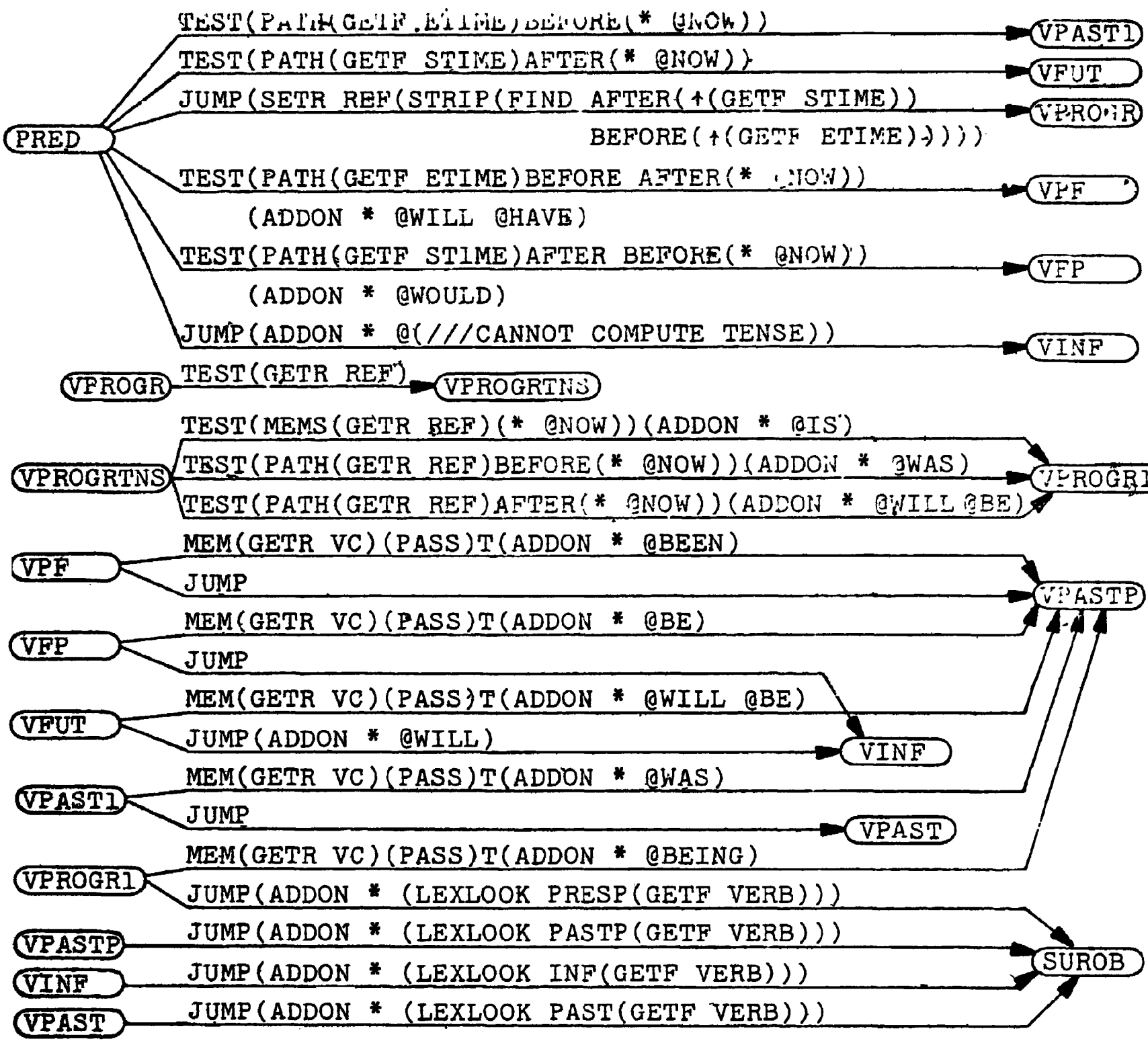

Figure 7: Tense generation network.

transfer is made to the node END. At that point, the contents of the register named * are returned.

(CONCAT form form*)

The forms are evaluated and concatenated in the order given. Performs a role analogous to that of Woods' BUILDQ.

(GETF sarc [sform])

Returns a list of all semantic nodes at the end of the semantic arcs labelled sarc from the semantic node which is the value

Tense	Active	Passive
past	broke	was broken
future	will break	will be broken
present progressive	is breaking	is being broken
past progressive	was breaking	was being broken
future progressive	will be breaking	will be being broken
past in future	will have broken	will have been broken
future in past	would break	would be broken

Figure 8: The tenses of "break" which the network of Figure 7 can generate.

of sform. If sform is missing, SNODE is assumed. Returns NIL if there are no such semantic nodes. It is similar in the semantic domain to Woods' GETF in the lexical domain.

(GETR regname)

Returns the contents of register regname. It is essentially the same as Woods' GETR.

(LEXLOOK lfeat [sform])

Returns the value of the lexical feature, lfeat, of the lexical entry associated with the semantic node which is the value of sform. If sform is missing, SNODE is assumed. If no lexical entry is associated with the semantic node, NIL is returned. LEXLOOK is similar to Woods' GETR and as also in the lexical domain.

(SETR regname form)

The value of form is placed in the register regname. It is the same as Woods' SETR.

(ADDTO regname form*)

Equivalent to (SETR regname (CONCAT (GETR regname) form*)).

(ADDON regname form*)

Equivalent to (SETR regname (CONCAT form* (GETR regname))).

(MEMS form form)

Returns T if the values of the two forms have a non-null intersection, NIL otherwise.

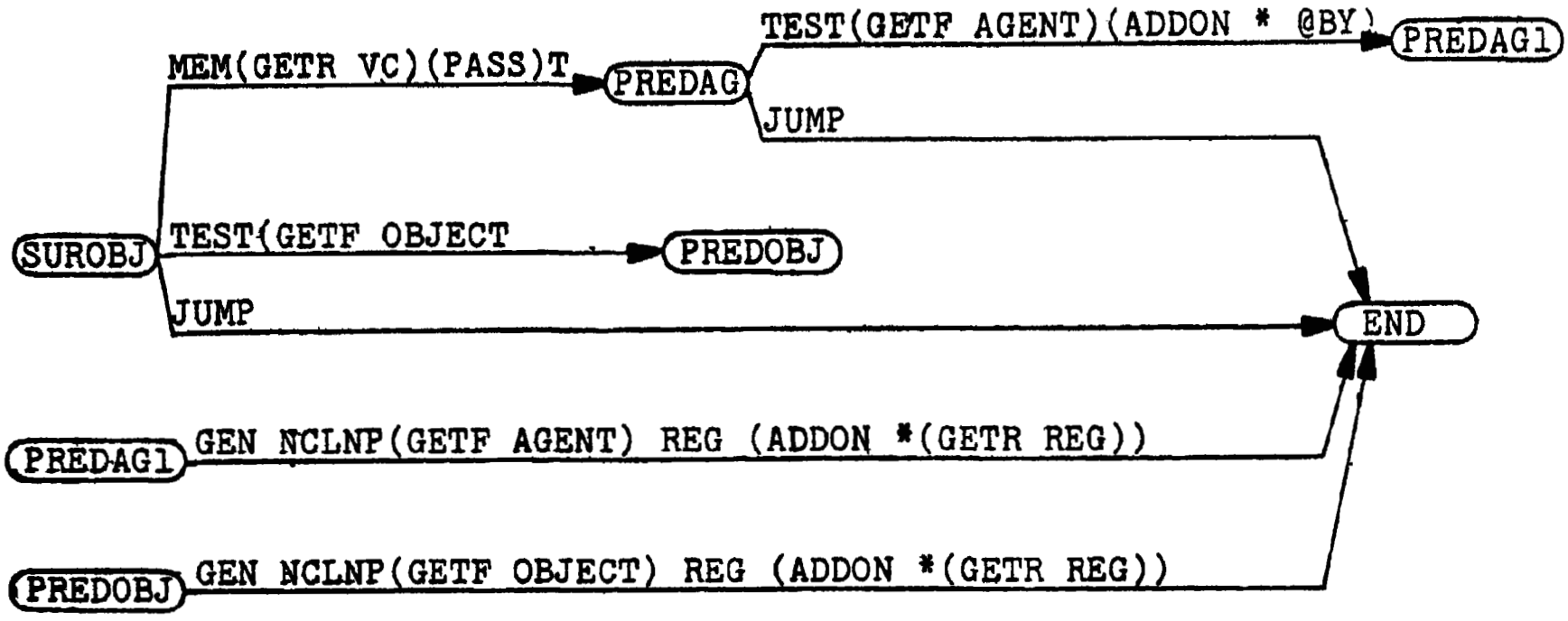

Figure 9: Generating the surface object.

(PATH $sform_1$ sarc* $sform_2$)

Returns T if a path described by the sequence of semantic arcs exists between the value of $sform_1$ and $sform_2$. If the sequence is $sarc_1$ $sarc_2$ ... $sarc_n$, the path described is the same as that indicated by $sarc_1$* $sarc_2$* ... $sarc_n$*. If no such path exists, NIL is returned. (Remember, * means repeat one or more times.)

Discussion of an Example Grammar Network

The top level generator function, SNEG, is given as arguments a semantic node and, optionally, a grammar node. If the grammar node is not given, generation begins at the node G1 which should be a small discrimination net to choose the preferred description for the given semantic node. This part of the example grammar is shown in Figure 5. In it we see that the preferred description for any semantic node is a sentence. If no sentence can be formed a noun phrase will be tried. Those are the only presently available options.

Semantic nodes with an outgoing VERB edge can be described by a normal SUBJECT-VERB-OBJECT sentence. (For this example, we have not used additional cases.) First the subject is generated,

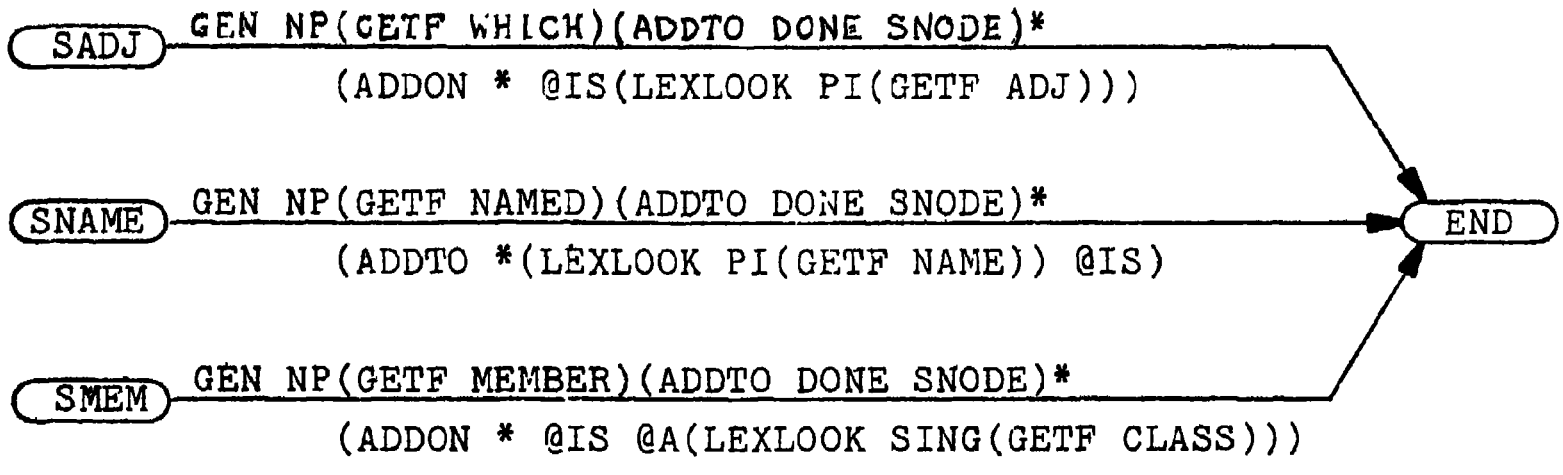

Figure 10: Generating the three "non-regular" sentences.

which depends on whether the sentence is to be in active or passive voice. Alternatively, the choice could be expressed in terms of whether the agent or object is to be the topic as suggested by Kay, 1975. Figure 6 shows the network that generates the subject. The register DONE holds semantic nodes for which sentences are being generated for later checking to prevent infinite recursion. Without it, node M0023 of Figure 1 would be described as, "A dog which kissed young sweet Lucy who was kissed by a dog which kissed..."

The initial part of the PRED network is concerned with generating the tense. This depends on the BEFORE/AFTER path between the starting and/or ending time of the action and the current value of NOW, which is given by the form (* @NOW). Figure 7 shows the tense generation network. Figure 8 shows the tenses this network is able to generate.

After the verb group is generated, the surface object is generated by describing either the semantic agent or object. Figure 9 shows this part of the network

The other three kinds of sentences are for describing nodes representing: (1) that something has a particular adjective attributable to it, (2) that something has a name, (3) that something is a member of some class. The networks for these are shown in Figure 10. Again, the DONE register is used to prevent such sentences as "Sweet young Lucy is sweet," "Charlie is Charlie," and "A dog is a dog."

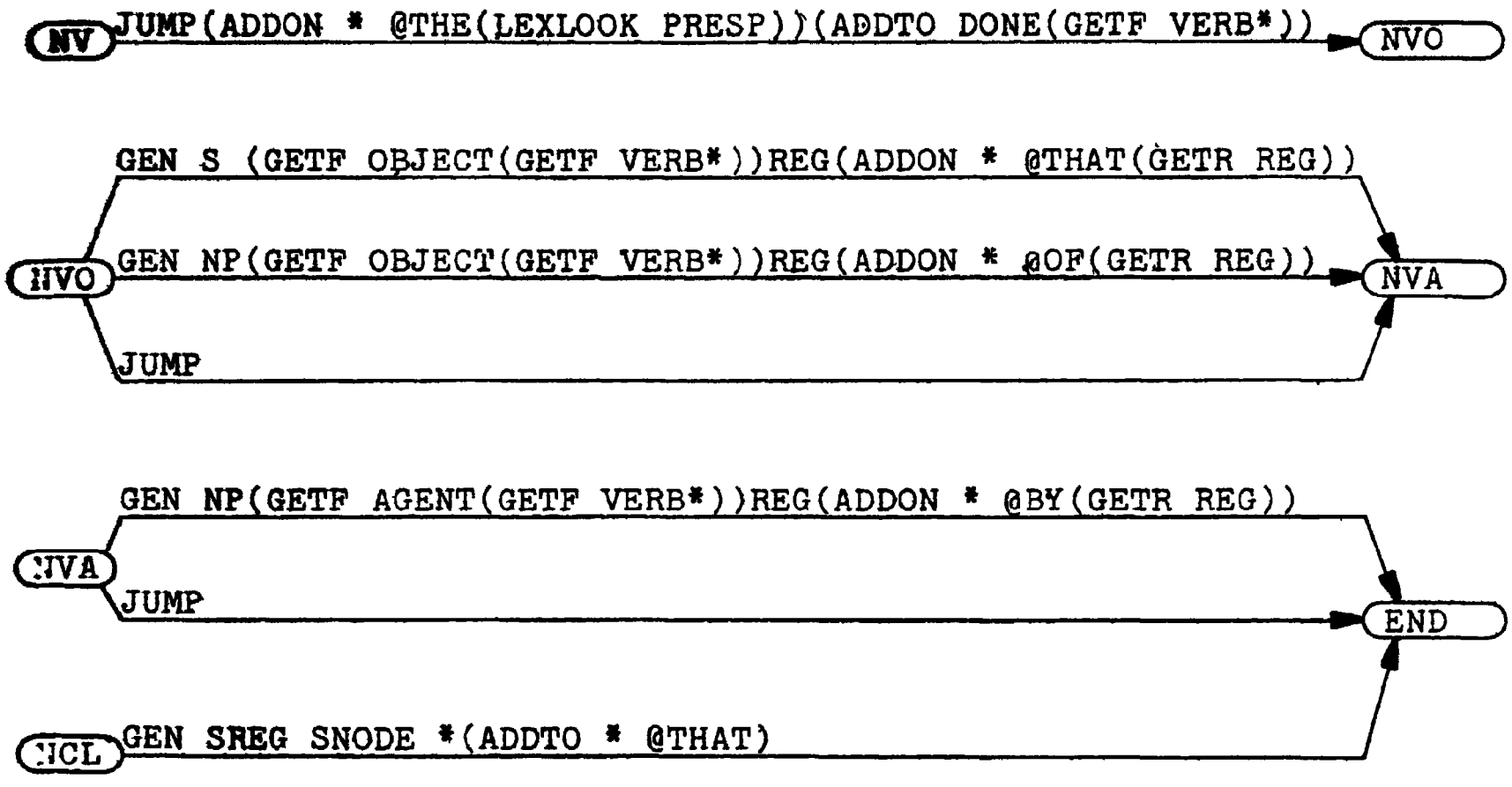

Figure 11: Generating nominalized verbs and sentences.

Fugure 5 showed three basic kinds of noun phrases that can be generated: the noun clause or nominalized sentence, such as "that a dog kissed sweet young Lucy"; the nominalized verb, such as "the kissing of sweet young Lucy by a dog"; the regular noun phrase. The first two of these are generated by the network shown in Figure 11. Here DONE is used to prevent, for example, "the kissing of sweet young Lucy who was kissed by a dog by a dog."

The regular noun phrase network begins with another descrimination net which has the following priorities: use a name of the object; use a class the object belongs to; use something else known about the object. A lower priority description will be used if all higher priority descriptions are already in DONE. Figure 12 shows the beginning of the noun phrase network. Adjectives are added before the name or before the class name and a relative clause is added after.

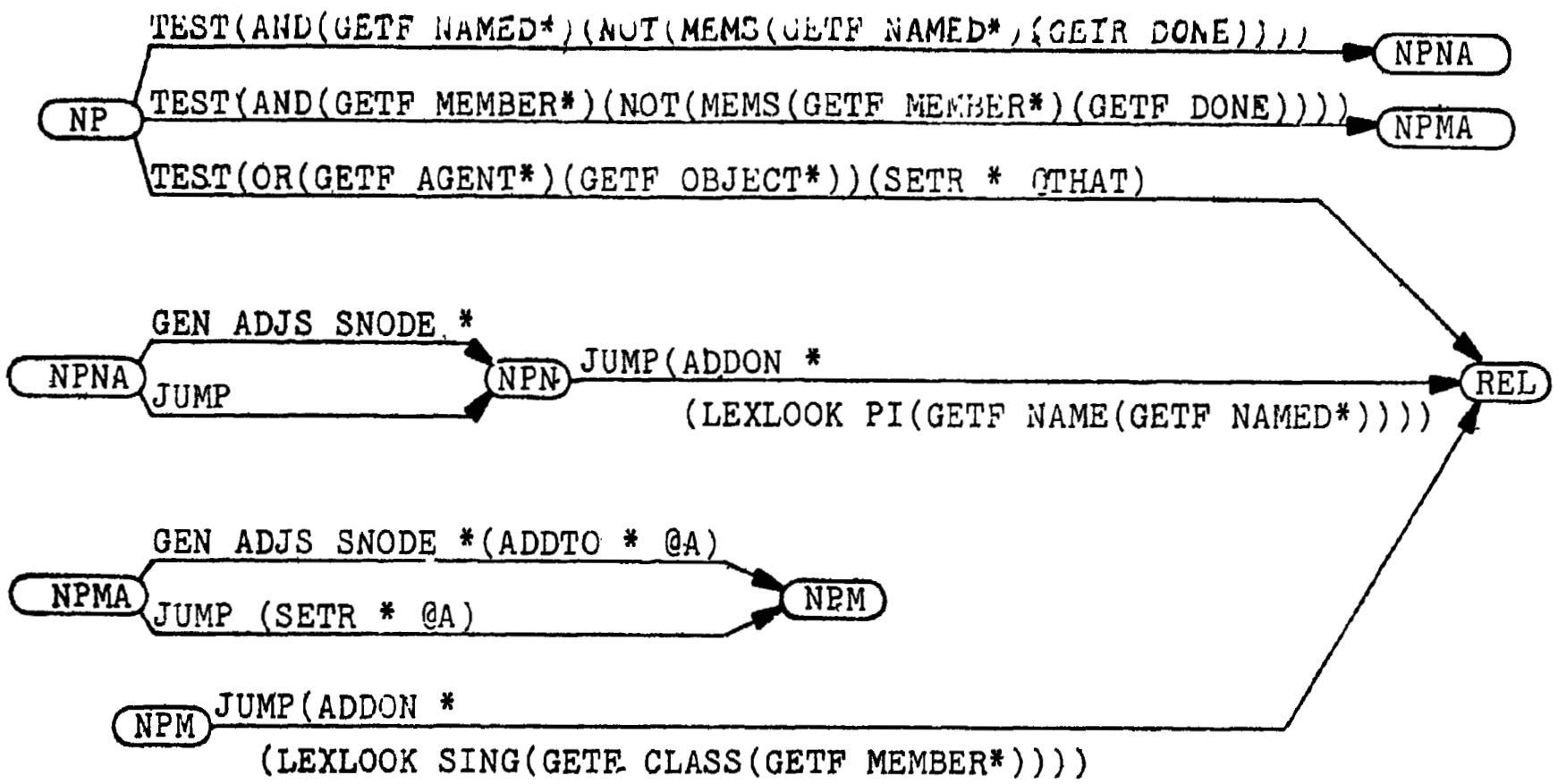

Figure 12: The beginning of the noun phrase network.

Figure 13 shows the adjective string generator and Figure 14 shows the relative clause generator. Notice the use of the TRANSR edges for iterating. At this time, we have no theory for determining the number or which adjectives and relative clauses to generate, so arbitrarily we generate all adjectives not already on DONE but only one relative clause. We have not yet implemented any ordering of adjectives. It is merely fortuitous that "sweet young Lucy" is generated rather than "young sweet Lucy". The network is written so that a relative clause for which the noun is the deep agent is preferred over one in which the noun is the deep object. Notice that this choice determines the voice of the embedded clause. The form (STRIP(FIND MEMBER (↑ SNODE) CLASS (FIND LEX PERSON))) is a call to a SNePS function that determines if the object is known to be a person, in which case "WHO" is used rather than "WHICH". This determination is made by referring to the semantic network rather than by including a HUMAN feature on the lexical entries for LUCY and CHARLIE.

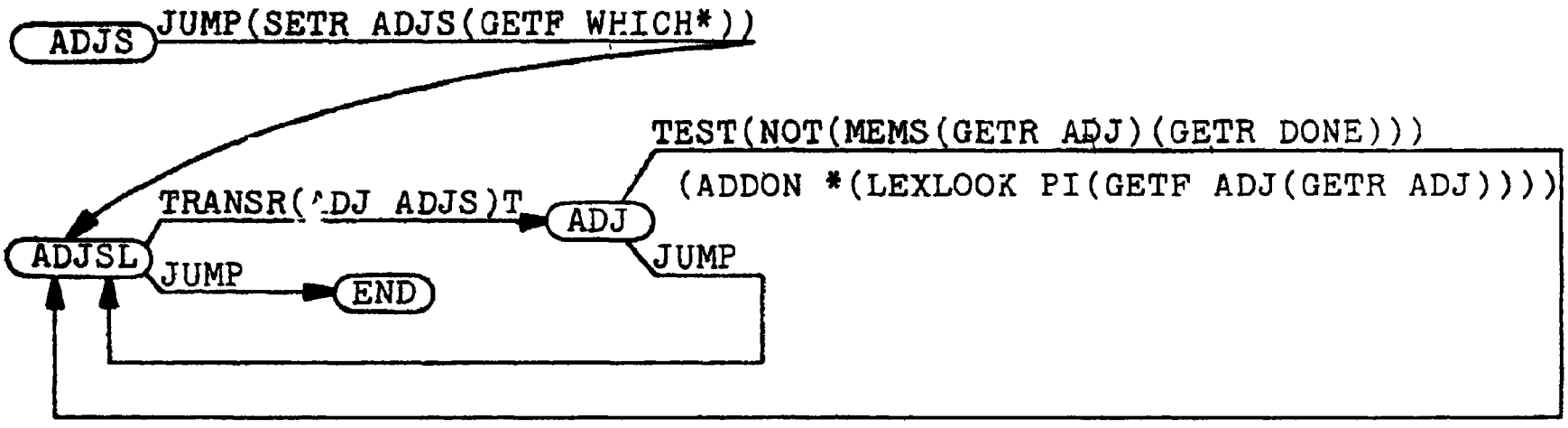

Figure 13: The network for generating a string of adjectives.

Notice that any information about the object being described by a noun phrase may be used to construct a relative clause even if that information derived from some main clause. Also, while the generator is examining a semantic node all the information about that node is reachable from it and may be used directly. There is no need to examine disjoint deep phrase markers to discover where they can be attached to each other so that a complex sentence can be derived.

Future Work

Additional work needs to be done in developing the style of generation described in this paper. Experience with larger and richer networks will lead to the following issues: describing a node by a pronoun when that node has been described earlier in the string; regulating verbosity and complexity, possibly by the use of resource bounds simulating the limitations of short term memory; keeping subordinate clauses and descriptions to the point of the conversation possibly by the use of a TO-DO register holding thenodes that are to be included in the string.

In this paper, only indefinite descriptions were generated. We are working on a routine that will identify the proper subnet of the semantic network to justify a definite description. This must be such that it uniquely identifies the node being described.

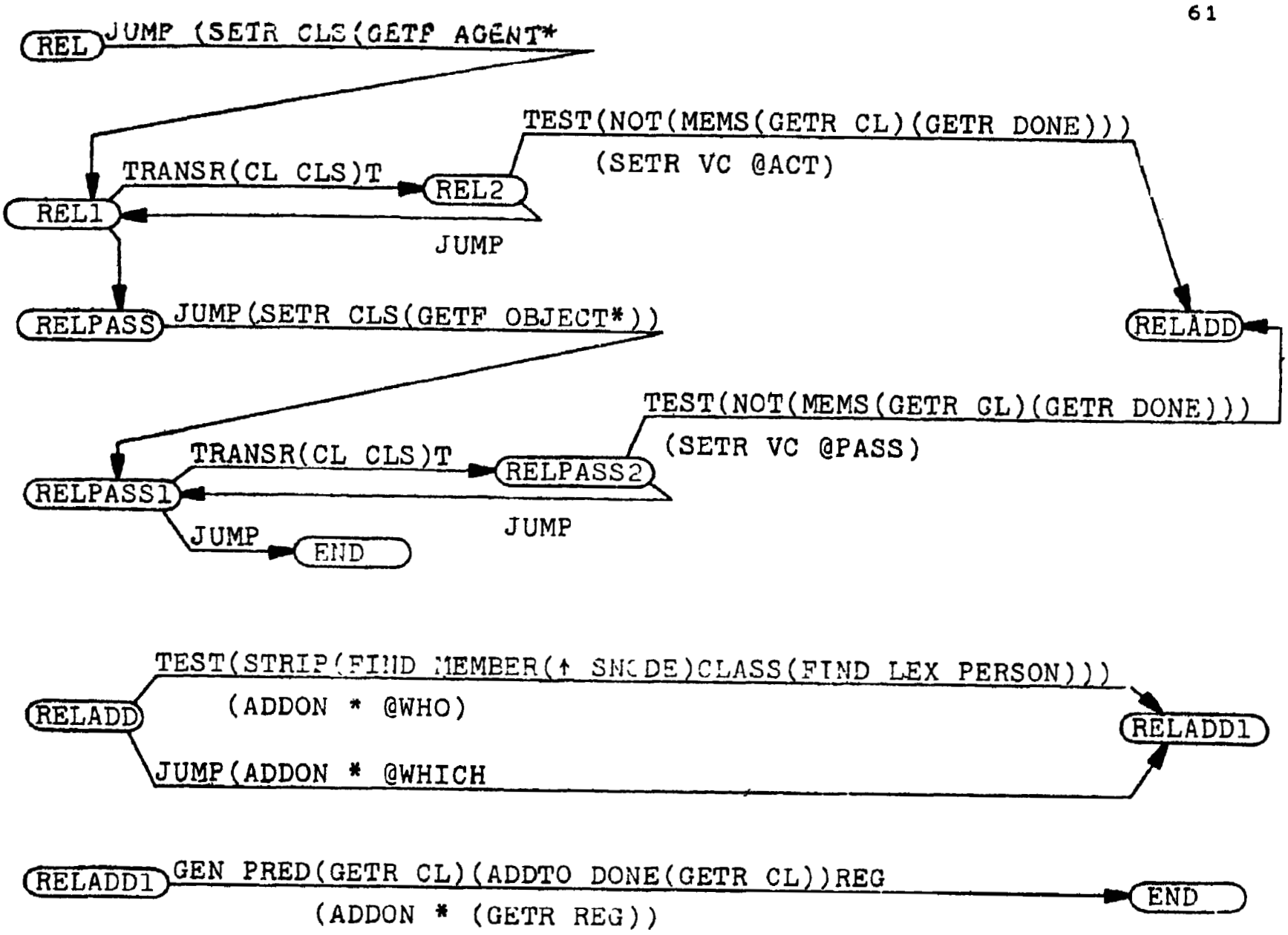

Figure 14: The relative clause generator.

Acknowledgements

The author is indebted to John Lowrance, who implemented the generator, Stan Kwasny, who implemented the parser, Bob Bechtel, who worked out the temporal representation, Nich Vitulli and Nick Eastridge, who implemented versions of SNePS, and Jim McKew for general software support. Computer service was provided by the IUPUI Computing Facilities. Typing and graphics were done by Christopher Charles.

References

Bruce, B.C. 1972. A model for temporal references and its application in a question answering program. Artificial Intelligence 3, 1, 1-25.

Kay, M. 1973. The MIND system. Natural Language Processing, R. Rustin (Ed.), Algorithmics Press, New York, 155-188.

Kay, M. 1975. Syntactic processing and functional sentence perspective. Theoretical Issues in Natural Language Processing R. Schank and B.L. Nash-Webber (Eds.), Bolt Beranek,& Newman, Inc., Cambridge, Massachusetts.

Schank, R.C.; Goldman, N.; Rieger, C.J., III; and Riesbeck, C. 1973. MARGIE: memory, analysis, response generation, and inference on English. Proc. Third International Joint Conference on Artificial Intelligence, Stanford University, August 20-23, 255-261.

Shapiro, S.C. 1971a. The MIND system: a data structure for semantic information processing. R-837-PR. The Rand Corp., Santa Monica, California.

Shapiro, S.C. 1971b. A net structure for semantic information storage, deduction and retrieval. 2nd International Joint Conference on Artificial Intelligence: Advance Papers of the Conference, British Computer Society, London, 512-523.

Shapiro, S.C. 1975. An introduction to SNePS. Technical Report No. 31, Computer Science Department, Indiana University, Bloomington,

Simmons, R.F. 1973. Semantic networks: their computation and use for understanding English sentences. Computer Models of Thought and Language, R.C. Schank and K.M. Colby (Eds.), W.H. Freeman and Co., San Francisco, 63-113.

Simmons, R.F., and Slocum, J. 1972. Generating English discourse from semantic nets. Comm. ACM 15, 10, 891-905.

Woods, W.A. 1973. An experimental parsing system for transition network grammars. Natural Language Processing, R. Rustin (Ed.), Algorithmics Press, New York, 111-154.