Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

A PROGRAM FOR GENERATING REPORTS ON THE STATUS AND HISTORY OF STOCHASTICALLY MODIFIABLE SEMANTIC MODELS OF ARBITRARY UNIVERSES

Sheldon Klein, John D. Oakley,
David J. Suurballe and Robert A. Ziesemer

Technical Report #142

November 1971

ABSTRACT, March 12, 1971: As submitted to 1971 International Conference on Computational Linguistics, Debrecen, Hungary September 4-7.

A Program for Generating Reports on the Status and History of Stochastically Modifiable Semantic Models of Arbitrary Universes

Sheldon Klein, John D. Oakley,
David J. Suurballe and Robert A. Ziesemer

The system consists of five parts:

1. A Component for representing any universe of objects and relationships in the form of a directed graph with labelled nodes and edges, wherein nodes represent semantic units and edges, relationships.

2. A Component for altering the relationships in the graph model as a function of external events and elapsed time.

3. A Component for describing selected portions of the graph in any specified context-sensitive phrase structure language.

Additional components include:

4. A device for generating events to alter the state of the relations in the modelled universe as a function of selected stochastic rules in a Monte Carlo simulation.

5. A program for controlling the subject matter and syntactic complexity of the reports on the status of the modelled universe.

For the initial testing of the system (which is fully designed and partially programmed at the date of this abstract) we have chosen to model a universe with objects and relations representing a small group of human beings living in a city. The events that drive the model and alter the universe dynamically are generated by a Monte Carlo Simulation using stochastic rules that execute the plot of a murder mystery story.

The report generator describes the development of the plot in English discourse produced from a context-sensitive phrase structure grammar. Component 5 permits

the narrative to be restricted to the point of view of any character, with full generative control of a number of quantitative aspects of style, including complexity and frequency of syntactic structures, and relative frequency of vocabulary items.

The system itself is quite general. All rules and representations are input as abstract data; accordingly the system can model not only any 'story' plot, but also non-verbal universes. Applications in addition to automatic story writing include all phases of information retrieval on a dynamically changing data base.

A Program for Generating Reports on the Status and History of Stochastically Modifiable Semantic Models of Arbitrary Universes*

Sheldon Klein, John D. Oakley, David J. Suurballe & Robert A. Ziesemer**

Computer Sciences Department
The University of Wisconsin
Madison, Wisconsin 53706
U.S.A.

1. Introduction

The system described in this paper is really an automatic novel writer. The somewhat intimidating title was chosen to emphasize the theoretical significance of the language model and the methodology used in what might otherwise be viewed as an amusement without theoretical import.

The techniques and methodology underlying much of this work are considerable refinements of those used by the first author as early as 1963 (2,3,4,5,6,10). In particular, syntax derived dependency networks involving some transitivity were used to answer questions, generate coherrent discourse and to produce paraphrases of essays with some control of style. In the publications of this work it was recognized that a syntactically derived dependency network was, at best, a poor approximation to one based primarily on semantic criteria.

The current language model uses a three level system, at the top of which are semantic objects and relations in the form of a directed graph with labelled edges.

---

*This paper was presented at the 1971 International Meeting on Computational Linguistics, Debrecen, Hungary, September 4-7.

**Robert A. Ziesemer is responsible for the implementation of the first version of the style control and dependency merging programs, David J. Suurballe is responsible for the implementation of the natural language generating program, John D. Oakley is responsible for the design and implementation of the simulation rule language, and Sheldon Klein is responsible for the overall design of the system.

The bottom level corresponds to the final printed version of the language productions and might be called graphemic. The middle level may be viewed as perhaps two levels depending upon one's point of view. Network structural relationships of one form or another are reflected in the language generation process at each level. Were one forced to find an affinity to a theoretical model in pure linguistic theory, one might select the Stratificational Grammar Model of Sydney Lamb, although transformations play a role in the linguistic system (7).

Network and dependency representations of semantic structures have been used by many researchers in computational linguistics. Among the earliest is Quillian (8), and among the most recent, Schank (9).

As an inspirational source for the system design of this program we are compelled to cite Roald Dahl (1).

The Semantic Network

The semantic units of the system are nodes and relations that link them. The state of the modelled universe at any static point in time is represented by sets of triples, each consisting of a node in a 'subject of relation' position, a relation, and a node in 'object of relation' position. The nodes of the universe are linked together by the relations in the form of a directed graph with labelled edges, wherein each relation is an edge, and the name of the relation is the label of that edge.

Nodes may also be associated with attributes without overt indication of relationship; however the relation status of attributes is implicit.

Lexical Expression Lists

Each semantic node and each relation in the system is linked to a lexical expression list consisting of lexical stems, phrases or semantic triples. Lexical

expression lists constitute an intermediate stage in the output language representation of the semantic elements. To a certain extent, the lexical expression list may be viewed as a mixed collection of synonyms, idioms and triples of the type used in the semantic network. However, they exist on an intermediate plane between the semantic domain and the object language level. A triple that occurs on a lexical expression list is not part of the semantic network, but serves instead as a definition that may be recursive because the nodes and relations in this triple are also linked to lexical expression lists.

Syntactic Dependency Templates

After the work of Klein (ibid) two kinds of syntactic dependency are recognized, transitive and intransitive. The generation of coherent discourse from a syntactic dependency network consisting of lexical stems linked together by directed dependency paths has been established. However, the determination of the proper direction of syntactic dependencies and their transitivity or non-transitivity in computing connecting paths is governed by semantic criteria. Accordingly, each relation in the semantic network is linked to a dependency template consisting of a 3 x 3 array, with vertical and horizontal dimensioning labelled by variables α, β, γ. Each α may be viewed as representing a lexical stem from a lexical expression list linked to the subject of relation node in a particular triple. The γ variable represents a lexical stem from a lexical expression list linked to the relation of that triple and, in a similar fashion, the β represents a lexical stem utlimately related to a node in the object-of-relation position in that triple. The cells in the dependency template array are filled with appropriate marks indicated which vertical elements are dependent on which horizontal elements, and whether such dependencies are tran-

sitive or intransitive. The use of templates will yield the most benefit in control of dependencies pertaining to prepositions and the use of forms of the verb 'to be'. The system will acknowledge (with different dependency templates) four relationships that might be represented by English 'is': class equivalence, class inclusion in either direction and attribute status.

We believe that the number of dependency tamplates will not be large and that all the relations in the system will be linked perhaps to no more than a dozen.

The Master Merged Dependency Array Governing Syntactic Productions

Assume that a sentence is to be generated that is derived from the semantic content of several semantic level triples. Through the use of the lexical expression lists and the dependency templates, all the transitively related lexical stems are entered in a master transitive array, constructed in the same fashion as the template, only larger. All the intransitively related elements are plotted in a master intransitive array. The transitively related elements are treated similarly. Then all paths connecting all elements are computed and entered in a merged master array (6, 11). The vertical and horizontal dimensions of this master merged array each contain, in an ordered fashion, all the lexical stem representations of the pertinent semantic triples. All elements are uniquely and repetively represented as they occur in various triples, except that lexical representations of the same semantic nodes are not repeated. Each cell of the array is marked to indicate whether or not a dependency path has been computed to exist between the lexical stems associated with the vertical and horizontal dimensions of that cell.

The master merged array is then used to monitor the production of sentences in the object language. The procedure from this point and beyond is well documented. (4. 5)

Suffice it to say that the generative mechanism uses subscripted, binary phrase structure rules that also contain dependency governor information. Lexical selection is made from the stems in the master merged dependency array, almost each time a node in the generation tree is rewritten. The master restriction on the selection is that no syntactic dependency relation may exist in the output sentence that does not exist in the master array. Transformations are used to convert the selected lexical items into words of appropriate grammatical category and to handle problems of agreement. The result is an object language sentence that 'truthfully' describes a portion of the original semantic network.

Dynamic Modification of the Semantic Network

Thus far we have described a static portrait of the system. A special programming language for expressing rules of change in the modelled universe is a major part of the system. Time exists in the universe, and passes in fixed increments (nominally 10 minutes) in the current version, but at optionally determined variable rates in the one currently under construction. At the end of each time interval, probabilistic rules that can alter the state of the universe at the beginning of the next time interval are tested. These rules use the conditions of the semantic universe existing during the current time interval as their data. These rules may refer to variables and predetermined classes whose domains are sets of nodes. The language in which the rules are formulated may be viewed as a programming language, and any particular set of rules that govern the flow of changes in the universe over a period of time may be viewed as a kind of behavioral computer program with branching logic governed by probabilistic criteria.

The following explication of the rules is partly in reference format. Examples are given last. For comprehension we suggest reading through the syntax and explanation, then studying the examples, referring back to earlier explications of particular terminology when necessary.

Syntax of Rules

<rule> ::= $RULE <action list> ; <subrule list>

<action list> ::= <main action><secondary action list>

<secondary action list> ::= <empty>|,<secondary action>|<secondary action list>,
<secondary action>

<main action> ::= <main action subject><action relation><main action object>

<secondary action> ::= <subject name> | X . <class name>

<main action subject> ::= <subject name> | Y . <class name><Y-restr. part>

<action relation> ::= <delete field><directed relation>

<delete field> ::= <empty> | ≠

<Y-restr. part> ::= :<max. no. of Y's for given X>|<empty>

<max. no. of Y's for given X> ::= <unsigned integer>

<subrule list> ::= <subrule>|<subrule list><subrule>

<subrule> ::= <subrule header><sentence list> ;

<subrule header> ::= <true value> , <false value> :

<true value> ::= <real number>

<false value> ::= <real number>

<sentence list> ::= <AND sentence list>|<OR sentence list>

<AND sentence list> ::= <sentence>|<AND sentence list> & <sentence>

<OR sentence list> ::= <sentence>|<OR sentence list> / <sentence>

<sentence> ::= <negate field> (<noun field><relation field><noun field>)

| <negate field><Boolean time sentence>

<Time-of-day> ::= <unsigned integer>

<Time operand> ::= T <relational operator><Time-of-day>

<Time operand list> ::= <Time operand>|<Time operand list><logical op><Time operand>

<Boolean time sentence> ::= [<Time operand list>]

<relation field> ::= <relation primary>

| <relation field><logical op><relation primary>

<relation primary> ::= <negate field><directed relation>

| <directed relation><duration modifier>

<duration modifier> ::= <relational operator><duration>

<duration> ::= <unsigned integer>

<directed relation> ::= <relation name>| ← <relation name>

<noun field> ::= <fixed noun>|<variable occurrence>

<variable occurrence> ::= <unlabeled variable>|<variable definition>

| <variable reference>

<variable definition> ::= #<variable name>.<class name>

<variable reference> ::= #<variable name>

<unlabeled variable> ::= #.<class name>

<variable name> ::= {ALPHA string (letters or digits), 1-8 chars long}

<fixed noun> ::= X | Y | <subject name>

<logical op> ::= & | /

<relational operator> ::= = | > | < | ≠

<negate field> ::= <empty> | ≠

<subject name> ::= {name of a node in the network}

<relation name> ::= {name of a relation in the network (i.e., labeled edge)}

<class name> ::= {name of a class in the network that represents a fixed set of nodes}

Explanation of some of the Major Syntactic Features

<fixed noun> is a symbol which represents a single value. A <subject name> is analogous to a literal: its value is the node in the network that it names. "X" and "Y" are analogous to simple variables: they can each have a single node as their value at any one time, but their values can change during the evaluation of a rule. The domains of values for X and Y are specified in the $RULE header line. The names "X" and "Y" are reserved words which cannot be used to name a node in the network. Examples:

X GEORGE BATHROOM Y SUSIE

<variable occurrence> is a symbol which can represent a list of nodes, or objects. The contents of this list can change dynamically during the execution of a single subrule. The initial definition of a variable (labelled or unlabelled) must supply a <class name>. The initial list of objects for a variable is a copy of the objects which comprise this class. Each reference to a variable in a sentence

(including the initial definition) will delete zero or more objects from the variable's list. Only those objects in a variable's list which do not satisfy the specified conditions in the sentence are deleted. If all the objects in a list satisfy the conditions, zero are deleted. If none do, all are deleted, leaving the variable list empty. Thus the size of a variable list never grows -- it either shrinks or remains the same for each reference to it.

Variables are distinguished by the "#" character which is prefixed to any variable occurrence. The initial definition of a variable uses the #<variable name>.<class name> format. All subsequent references to this variable must use the #<variable name> format, or else an error will be produced. If the initial definition is the only reference that is used (i.e. if there are no subsequent references), <variable name> can be omitted, to give the alternate format #.<class name>. Examples:

#M1.MEN	#ROOMVAR.ROOMS	#V.PEOPLE
#M1	#ROOMVAR	#V
#.MEN	#.ROOMS	#.PEOPLE

(MEN, ROOMS, PEOPLE are class names.)

<directed relation> specifies either a forward relation (in the absence of "←"), or a passive relation (in the presence of "←"). Thus "←LOVES" means "is loved by". In any given triple of subject, relation, object (S R O), the form (S←R O) is equivalent to saying (O R S). Thus (JOHN ←LOVES MARY) is the same as saying (MARY LOVES JOHN). Similarly, (BEDROOM ← IN JANE) is equivalent to (JANE IN BEDROOM).

Note that if R is an attribute-type relation, ←R has no meaning. Thus (JOHN HAPPY) is meaningful, whereas (JOHN ←HAPPY) is not. Examples:

LOVES ←LOVES HAPPY ← IN

<relation primary> is the basic unit of <relation field>. The <negate field> is used to indicate whether one is testing for the existence or absence of a triple in the network. Thus (JOHN IN OFFICE) would evaluate T if John was indeed in the office, F if not, while (JOHN ≠IN OFFICE) would evaluate F and T respectively. Similarly, (BEDROOM ≠←IN JANE) would evaluate F if Jane was in the bedroom, T otherwise.

A <duration modifier> allows one to test how long (in plot-time minutes) a triple has been in the network. Thus (JOHN IN > 30 OFFICE) would evaluate T only if John is in the office, and has been for more than 30 minutes. Or: (JOHN←MAD<15 GEORGE) is T only if George is mad at John and has been for less than 15 minutes. Note that there is no way of testing how long a node has been in non-existence, so ≠IN > 20 would be illegal. Examples of <relation primary>:

IN MAD < 20 ≠HAPPY ≠←LOVES ≠← IN

<relation field> is one or more <relation primary>s joined by logical operators. Several relations can be joined together where the subject and object are the same, rather than make separate tests. For example, (JOHN LIKES/HATES MARY) returns T if John either likes or hates Mary.

Any relation field is evaluated left to right, with "&" and "/" having equal priority. Thus (FRED LIKES/LOVES & MAD < 30 GEORGE) tests to see if

Fred is mad at George (and has been for less than 30 minutes), and Fred either likes or loves George. Parenthesization is not allowed in the syntax, but using it for clarification would give (((LIKES)/LOVES) & MAD < 30).

<sentence> is the basic unit of test. Each sentence evaluates to a logical value, T or F (true or false), depending on whether the postulated situation in the sentence actually exists in the network or not. If a "≠" sign preceeds a sentence, the result of the test is complemented.

Each sentence (except time sentences) is of the form (S R O) or ≠(S R O). "S" and "O" are <noun fields> and "R" is <relation field>. A single sentence can test only for various relations between the same subject and object. For more complicated tests multiple sentences must be used, or possibly several subrules. Examples:

≠(JANE LOVES/LIKES X)	Returns T if Jane neither loves nor likes the object represented by X.
(Y ← EMPLOYS & ≠DISLIKES FRED)	Returns T if the object represented by Y is employed by, and does not dislike, Fred.
(#W1.WOMEN LOVES JOE)	Returns T if at least person (i.e., object) from the class of WOMEN loves Joe. After evaluation of this sentence, the variable W1 will represent the list of all WOMEN who love Joe. If no women love Joe, W1 will empty and F will be returned.
≠(#PP LOVES #LP.LEPERS)	Returns T if no one in the variable PP loves anybody in the class LEPERS. (Note that the variable PP must be defined in the previous sentence by means of a <variable definition>.) After evaluation, only those persons in PP who love someone in the class of LEPERS will be retained in the variable PP. The final value of LP will be only those who are in the class

LEPERS and are loved by at least one person in PP. Note that PP being empty implies LP is empty, and vice versa. In such a case, the quantity inside the parentheses will be F, giving an overall result for the sentence of T.

<Boolean time sentence> is a special form of sentence which one can use to test the time-of-day within the plot simulation. All times are based on a 24-hour clock. Each <time operand> returns T or F, and the results of several <time operand>s can be combined with logical operators into a logical expression. Evaluation of such an expression is similar to the method used in <relation field>: left to right, equal precedence of "&" and "/", no parenthesization. Note that time sentences can be combined with other sentences in a sentence list without restriction. Examples

[T < 0830]	Returns T if plot time is earlier than 8:30 a.m.
[T > 1300 & T < 1500 & T ≠ 1400]	Returns T if plot time is later than 1:00 p.m., earlier than 3:00 p.m., and not equal to 2:00 p.m.

<subrule> is made up of one or more sentences joined by logical operators. The sentences in a subrule can either be all ANDed together, or all ORed together, but the operations cannot be mixed in a single subrule. The result of the evaluation of a subrule is a real number, selected from one of the two at the start of the card. Which number is selected depends on the evaluation of the sentence list for that subrule: if the sentence list evaluates T, the first number is used; if F, the second number. Both numbers may be any valid real number (positive or negative), with or without a decimal point. The number from each subrule is used in the calculation of a cumulative numerical probability. This probability is eventually used to determine if the actions specified in the $RULE header line are implemented or not.

Variables cannot overlap subrule boundaries, i.e., they cannot be defined in one subrule and referenced in another. They are to be used as local variables only, within the scope of a single subrule. Example:

+.2, -.2: (X IN #R.ROOMS) & (JOHN IN #R) & (JOHN LIKES & ←LIKES X);

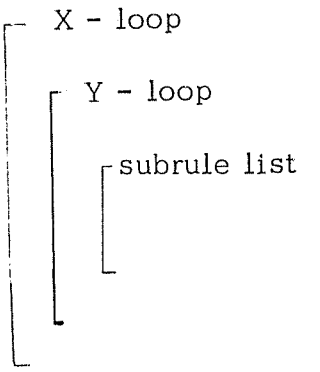
The sentence list in this subrule will return T if X is in some room R, John is in the same room R, and John both likes and is liked by X. In this case, the subrule will return +0.2 as its value. However, if X is not in any room, R will be set empty and F will be returned. Or, if John is not in the same room as X, R will also be set empty (after the second sentence) and F returned, causing the subrule to return -0.2. The third sentence could also the sentence list to fail.

<rule> is the result obtained by combining tests of the network with the desired actions. For any rule, there are two parts: the tests which test for certain situations in the network, and the actions that are to occur if these certain situations do indeed exist.

The <action list> may contain one or more actions. The first action, which is mandatory, is called the <main action>. The <main action> is distinguished from subsequent actions because it is used to specify the domains for X and Y (if they are used). For example:

$RULE X.MEN LIKES Y.WOMEN;

is a header line which says the following: Let X represent, one at a time, each object in the MEN class. For each such X, cycle Y through all possible values in the class WOMEN. Then, for each X and Y pair, evaluate the <subrule list> for this rule. The loop effect is as shown:



Within any particular evaluation of the subrule list, X and Y have a fixed value.

Note that if the class specified for the X variable contains 8 objects, and that for the Y variables contains 5 objects, the subrule list for this rule will be evaluated 40 times. Such nested loops can be very time consuming and should be used with caution.

The use of the subrules is as follows. For any given cycle through the subrule list, a counter is initially set to zero. This represents a cumulative probability. Each subrule is evaluated in succession, each returning some real number which is then added to this cumulative total. Negative numbers thus decrement the counter, positive ones increment. After the last subrule in the subrule list is evaluated, this cumulative probability counter is taken as the probability that the actions specified in the <action list> of the rule be implemented. Consequently a random number is generated and compared to the counter: if it is less than or equal to the counter, the rule actions are implemented; if it is greater than the counter, nothing happens. For example:

```
$RULE   GEORGE INVITES Y.PEOPLE;
.2, 0:  (GEORGE KNOWS & ≠HATES & ≠DISLIKES Y);
.6, 0:  (GEORGE LIKES/LOVES Y);
.3, 0:  (#P.WOMEN MARRIED GEORGE) & (#P LIKES/LOVES Y);
```

Since there is no X-loop, the loop structure is:

Y - loop

Subrule list of 3 subrules.

For each Y, the three subrules are evaluated. George will invite Y (to a party) with the following probabilities: 0.2 if George knows Y and neither hates nor dislikes him, an <u>additional</u> 0.6 if George likes or loves Y, and an <u>additional</u> 0.3 if there is a women who is married to George (i.e., George's wife) who either likes or loves Y. If all three subrules evaluate to their first number, the cumulative probability becomes 1.10, which is of course equivalent to absolute certainty. If only the last subrule evaluated to its true value, the probability that George invites this particular Y would be 0.3. The above steps are repeated for every object in the class PEOPLE.

Multiple actions can be specified in a $RULE header line by specifying a non-empty <secondary action list>. For example,

```
$RULE   GEORGE   ≠MAD   Y.MEN
        Y        ≠MAD   GEORGE;
```

is a rule header line which would <u>delete</u> the triples (GEORGE MAD Y) and (Y MAD GEORGE) from the network, where Y represents a particular object from the class of MEN. The delete operation is specified by the "≠" character.

In some cases, one needs to be able to limit the triples that can be implemented in a rule. For example,

```
$RULE   JOHN   IN   Y.ROOMS;
```

is a rule header line which has the possibility (depending on the subrule list) of setting (JOHN IN BEDROOM), (JOHN IN KITCHEN), (JOHN IN LOUNGE), etc., into the network at once, creating a semantic question of where John really is located. Problems such as this can be solved by appending a <Y-restriction part> to the <main action object>. This allows one to specify a maximum number of Y's that can be set true for a given <main action subject>. Thus,

```
$RULE  JOHN  IN  Y.ROOMS:1;
```

would produce a random pick of all the rooms that John would have moved into (as specified by the subrule list evaluations) to select just one (in this example). Note that the <Y-restriction part> is a maximum; it does not guarantee that at least that many Y's will be chosen.

It is frequently desirable to be able to test a condition and, depending on the outcome of the test, either immediately terminate the rule for this X-Y pair, or immediately set the actions true for this X-Y pair and forget about further subrule evaluations. This is done by specifying a probability of +10 or -10. For example,

```
$RULE          GEORGE KISSES Y.GIRL;
0, -10:        (GEORGE IN #R.ROOM) & (Y IN #R);
.3, -.1:       (GEORGE LIKES Y);
.3, 0:         (Y LIKES GEORGE);
```

This rule is used for determining if GEORGE kisses a girl. Obviously they must be in the same room for this to happen. The first rule tests this condition and returns -10 as the probability if George and Y are not in the same room. This immediately terminates the rule for this X (i.e., GEORGE) and this Y (some GIRL). The rule

would then go on to the next X-Y pair. Similarly if the cumulative probability total ever exceeds a certain positive number* which +10 does), the actions are immediately set true and the next X-Y pair is tested (or the next rule is evaluated, if there are no more X-Y pairs to be tested). Essentially, this technique allows the rule writer to test for the absence of a necessary condition (in which case -10 would be used), or the presence of a sufficient condition (when +10 would be used). In either case, further processing of subrules would be superfluous.

Attributes

Attributes are special types of relations which do not have object nodes. They serve only to describe the subject node from which they emanate. Since the BNF does not describe the syntax for attributes, the syntactical differences are mentioned here.

The major syntactic difference is due to the fact that an object must not be specified for a triple with an attribute as a relation. For example, (X SLEEPY) is a triple with SLEEPY as an attribute. The object is implicitly null.

Attributes can occur in a <relation field> with other relations. If one or more of the relations in a <relation field> is a non-attribute, however, then an object must be specified for such relations; if all the relations are attributes, then the object must be omitted. Attributes can occur in an <action list> also. An object must not be specified in such a case. Finally, as stated previously, it makes no sense to say (JOHN ← HAPPY) since there is no object, so this form is illegal.

Examples of correct usage of attributes:

---

*The actual limits were chosen arbitrarily as ± 5.0.

1. (X ≠HAPPY & ≠SAD)

2. (FRED LIKES & HAPPY > 45 MARY)

3. $RULE X.WOMEN HAPPY,
X ≠SAD;

4. $RULE GEORGE LIKES JOHN
JOHN ≠SAD,
JOHN LIKES GEORGE,
GEORGE HAPPY,
GEORGE ≠SAD;

Examples of Rules

1. $RULE X.PEOPLE GETOUT BED,
X ≠ASLEEP;
.1, -10: [T > 0400];
.2, 0: [T > 0700];
.4, 0: (X ≠SLEEPY)/[T > 0830];
-.2, 0: (X ASLEEP < 300);
.99, 0: [T > 1300];

This is a possible rule for people getting out of bed in the morning (and awakening). The first subrule says that if it is 4:00 a.m. or earlier, no one will get up (probability = -10), but if the time is later than 4:00 a.m. there is a .10 chance that X will get up. There is an additional .20 chance if it is later than 7:00 a.m. The third subrule says that if X is not sleepy or if it is later than 8:30 a.m., there is an additional .40 chance that X will get up. However, if X has been asleep for less than 5 hours (300 minutes), then the fourth subrule deducts .20 from the overall probability. Finally, if it is later than 1:00 p.m., .99 is added to the cumulative probability.

```
2.      $RULE   X.MEN    TALKS    Y.MEN;

        -10,0:  [T < 0800] / [T > 1700] / (X IS Y);
        0, -10: (X IN OFFICE) & (Y IN OFFICE);
        .3,0:   (X LIKES Y);
        -.3,0:  (Y HATES/DISLIKES X);
        -.15,0: (#EMP.MEN EMPLOYS X) & (#EMP IN OFFICE)
                      & (Y ≠IS #EMP);
        .7,0:   [T > 0959 & T < 1016]/[T > 1359 & T > 1416];
```

This is a possible rule for determining if one man talks to another at work. The first subrule requires that the time of day between 8:00 a.m. and 5:00 p.m., and that X is not the same person as Y (we assume that triples like (JOHN IS JOHN) exist in the network for all objects which represent people). The second subrule requires that X and Y are both in the room called OFFICE. The third subrule gives a .30 chance that X will talk to Y if X likes Y. However, if Y either hates or dislikes X, then.30 is deduced from the cumulative probability (the fourth subrule). The fifth subrule essentially states the following: if one or more of X's superiors is in the office (i.e., someone who employs X), and Y is <u>not</u> that person, then.15 is <u>deduced</u> from the cumulative probability that X talks to Y. This corresponds to an employee tending to be more industrious when his boss is in the room. The last subrule adds .70 to the probability of the time is between 10:00 a.m. and 10:15 a.m. or between 2:00 p.m. and 2:15 p.m. These are assumed to be break periods for the employees.

Narrative Style Control Monitor

A report on the status of the modelled universe is issued at the end of each time frame in a selected context-sensitive phrase structure language (English for all current work).

The problem is what to describe. One might use the total semantic network to generate a total description of itself each and every time frame at a horrendous cost in time and redundancy.

The first version of this program was set to issue descriptions of just the changes in the semantic network that occurred during the previous time frame.

The style control monitor currently under construction is intended to yield a global control of style. The control of lexical choices and syntactic complexity in a dependency driven essay generator has already been demonstrated (4,5). The features of control intended for the current system include:

a. lexical frequency

b. syntactic structure frequency

c. narrative subject matter

d. descriptive complexity

e. internal paragraph structure

f. paragraph grouping

Lexical frequency and syntactic structure frequency are obtained through weighted probabilistic selection of lexical items, and through weighted probabilistic selection of phrase structure rules.

To control the subject matter, it is necessary to have a powerful device for selecting particular subpaths through the semantic network to serve as inputs to

the narrative generation component. Accordingly, a program is under construction for finding complex paths through the network that are a function of a variety of logical conditions.

Typical requests to the program might be:

"Find a path between node A and node B that passes through relation R_1, but not through R_2"

If R_1 is "likes" and R_2 is "knows", the retrieved path, if any, would describe the network connection between John and Mary based on friendship and acquaintance, direct or indirect, but not involving what connection R_3 represents.

Another request might be:

"List all paths from A, to a distance of 2 nodes, that pass through relations R_5 and R_6."

If R_5 is "likes" and R_6 a locative "in", then the second formulation might extract that portion of the network indicating whom John likes, and where such parties are located.

The minimal unit of description is the sentence. The narrative control can force selection of any subject matter or lexical item in part of the sentence generation process, or leave any or all details to random choice. The selection of subject matter is of two types: the choice of subportion of the network for description (an ultimate constraint on the domain of discourse), and selection of specific lexical items in the major syntactic slots of particular sentences.

Global control of subject matter can be obtained through the first method. For example, one may choose to write a narrative from the point of view of one particular character, in which case all network subset extractions would be limited to informa-

tion accessible to that character.

In all phases of control, the weighted parameters used in making the random selections can be modified so as to make the control of the various elements of style fall anywhere in the range of total randomness to total determinism.

A Trial Run

Presented next are most of the data for the first trial run, and the resultant output. We note, as both a virtue and a flaw,that there are many inconsistencies in the way the information is distributed. For example information that might occur as a relation is sometimes expressed as a single unit in the lexical expression list, and vice versa, or both. The flaw is a matter of perceived theoretical esthetics; the virture is that the operation of the system is immune to such inconsistencies. While one might wish to use ideal semantic & lexical units, their prior determination is not essential to the operation of the model.

The data is very sketchy, and in no way reflects any limitations of the system. It was for the first test run of the full system. Output was obtained for several time cycles before an error in the program caused it to stop. The computer, a Burroughs B5500,was taken away from us and the University of Wisconsin before the error could be corrected. The data and output presented represent the state of the system the night before the computer departed.

Node	Lexical Expression List
George	Computer-manufacturer, boss, George
Margaret	Margaret, George's wife
Laslo	Laslo, systems-analyst
Medea	Medea, Laslo's wife
Henri	Henri, director, director-of-the-computer-center
Helene	Helene, Henri's wife
Umberto	Umberto, bachelor, race-car-driver, Italian
Philip	Philip, poet
Lili	Lili, computer-programmer
Suzanne	Suzanne
Gilda	Gilda, nighclub-singer
giroom	living room
gbasement	basement
gfactory	factory
goffice	office
gapt	apartment, George's apartment
suzapt	Suzanne's apartment
lasloapt	Laslo's apartment
umbhtrm	Umberto's hotel room
computing	computing
cigars	cigars
gambling	gambling
niteclubs	nightclubs
drink	drink, cocktail
party	party
gourmets	gourmet cooking
beethovn	beethoven
debussy	Debussy
alcholol	alchol, liquor, wine

Relations

The following relations are classified T, I or A, accordingly as they are transitive, intransitive or attribute types. Transitivity here is used differently from the way we have defined it for the dependency relations that feed the natural language generator. Here the terms are of a logical semantic nature. For example, if a rule condition asked if node A were in node C, and the existing network triples were "A in B", and "B in C", the rule evaluator would compute the condition as being true if the particular 'in' was labelled T, for transitive. Other 'in' relations (not indicated) might not be transitive.

Relation	Lexical Expression List
In T	in
married I	married to
dislikes I	dislike
isin I	is in
works I	works for
mistress I	mistress of
angry I	angry
invites I	invite
IX1 A	tipsy
IX3 A	drunk
accept I	accept
brother I	brother of
dark A	dark
short A	short
plump A	plump
rich A	rich
jealous A	jealous
blond A	blond

Relation	Lexical Expression List
likes I	like
smokes I	smoke
loves I	love
is in	
bores I	bore
hates I	hate
wants I	want
get I	get, obtain
IX2 A	mildly drunk
props I	proposition
knows I	know
sister I	sister of
bald A	bald
passion A	passionate
slim A	slim
Hungarn A	Hungarian
tall A	tall
blueyed A	blue-eyed

Relation	Lexical Expression List	Relation	Lexical Expression List
medium A	medium-height	Italian A	Italian
handome A	handsom	sexy A	sexy
single A	single		

Classes	Nodes and Class Members
men	George, Laslo, Henri, Umberto, Philip
women	Margaret, Medea, Helene, Lili, Suzanne, Gilda
people	(men), (women)
rooms	glroom, gbasement
offices	goffice

Attribute List

Node	Attribute Relations
George	dark, bald
Margaret	short, dark, passion, plump
Laslo	slim, rich, dark, Hungarn
Medea	short, dark, slim, passion, jealous
Henri	tall, passion, jealous
Helene	blond, tall, slim, blueyed
Umberto	medium, dark, Italian
Philip	tall, dark, handsome
Lili	blond, tall, slim, single
Suzanne	single, blond, sexy
Gilda	sexy

Triples

Node	Relation plus Node
George	likes niteclb, likes Margaret, married Margaret, loves Lili, smokes cigars, likes gambling, isin computing, wants party, likes Gilda, knows Henri, likes Umberto, likes Suzanne, likes Philip, loves Medea, in glroom, knows Laslo
	* * * * * *
Laslo	likes Beethovn, in goffice, isin computing, works Henri, married Medea, likes Umberto, likes Medea
	* * * * * *
Medea	married Laslo, loves Laslo, bores Laslo
	* * * * * *
Henri	likes Debussy, likes alchohol, brother Suzanne, loves Helene, isin computing, married Helene
	* * * * * *
Helene	loves Henri, married Henri
	* * * * * *
Lili	isin computing, mistress George, likes Philip, in Lasloapt
	* * * * * *
Suzanne	sister Henri, in Suzapt, likes Philip
	* * * * * *
Gilda	sister Margeret, in gapt
	* * * * * *
Umberto	likes Laslo, in umbrtrm
	* * * * * *
Philip	likes Lili, likes Suzanne
	* * * * * *

Trial Run Simulation Rules

The following rules are the first segment of a plot that was to yield a murder mystery. The initial model covers invitations to a cocktail party, drinking behavior of guests, and amorous behavior of the hostess of the party and various males guests.

```
%       RULE FOR INVITING PEOPLE TO THE PARTY
$RULE          GEORGE INVITES Y.PEOPLE,
                GEORGE ≠WANTS PARTY:
-10, 0:        (GEORGE ≠ PARTY);
10, 0:         (Y ISIN COMPUTING)
            & (GEORGE KNOWS/LIKES/LOVES Y);
.7, 0:         (GEORGE LIKES/LOVES Y);
.7, 0:         (MARGARET LIKES/LOVES Y);

%       RULE FOR PEOPLE ARRIVING AT GEORGES LIVING ROOM
$RULE          X.PEOPLE IN GLROOM,
                GEORGE ≠INVITES X;
0, -10:        (GEORGE INVITES X);
.4, 0:         [T>1859];
.4, 0:         [T>1911];
.2, 0:         [T>1921];

%       RULE FOR GETTING PEOPLE TO DRINK
$RULE          X.PEOPLE GET* DRINK;
0, -10:        (X IN GLROOM);
-.1, 0:        [T>2200 / T<0530];
-.1, 0:        [T>2300 / T<0530];
-.1, 0:        [T<0530];
.7, 0:         (X ≠IX1 & GET>19 DRINK);
.3, 0:         (X LIKES ALCOHOL);
.8, 0:         (X ≠GET DRINK);
```

```
%       RULES FOR ENTERING STATES IX1, IX2, IX3

$RULE           X.PEOPLE IX1;
.9, 0:          (X GET=0 DRINK);
$RULE           X.PEOPLE IX2*,
                X        IX1;
.9, 0:          (X IX1≠0 & IX1<21 & GET=0 DRINK);
$RULE           X.PEOPLE IX3*,
                X        IX2,
                X        IX1;
.9, 0:          (X IX2≠0 & IX2<21 & GET=0 DRINK);

%       RULES FOR MAKING PEOPLE LEAVE STATES IX1, IX2, IX3
$RULE           X.PEOPLE ≠IX3;
.6, 0:          (X GET>90 DRINK);
$RULE           X.PEOPLE ≠IX2;
.7, 0:          (X ≠IX3 & GET>120 DRINK);
$RULE           X.PEOPLE ≠IX1;
.8, 0:          (X ≠IX2 & GET>150 DRINK);

%       RULE FOR DETERMINING WHETHER MARGARET PROPOSITIONS SOME MAN
$RULE           MARGERET PROPS Y.MEN:1;
-10, 0:         (Y MARRIED MARGARET);
0, -10:         (MARGARET IN GLROOM) & (Y IN GLROOM);
.2, .1:         (MARGARET IX1);
.2, 0:          (MARGARET IX2);
-.1, 0:         (MARGARET IX3);
-.1, 0:         (MARGARET DISLIKES Y);
.2, 0:          (Y HANDSOME);

%       RULE FOR DETERMINING ACCEPTANCE OF MARGARETS PROPOSITION BY \
0, -10:         (MARGARET ACCEPT Y.MEN:1;
0, -10:         (MARGARET PROPS≠0 Y) & (MARGARET IN GLROOM)
          & (Y IN GLROOM);
```

```
.1, .9:          (Y  MARRIED  #.WOMEN);
.2, 0:           (Y  IX2);
.1, 0:           (Y  IX3);
$END;
```

The following output is derived from an extremely simple grammar used for the first test purpose. As indicated earlier, errors in the program prevented the completion of the trial run.

Initial conditions output:

"George knows Laslo. The boss is in the living room. George loves Laslo's wife. The boss likes Philip. The boss likes Suzanne. The boss likes Umberto. The boss knows the director. The boss likes Gilda. The boss wants the party. The boss is in computing. George likes gambling. The boss smokes cigars. George loves the computer programmer."

First Time Frame Output

"The boss invites the systems analyst. George invites the director. The boss invites Lili."

Second Time Frame Output

No English output due to program error.

Discussion

It is a rather long distance from promise of the title to the fragmentary output just presented. The first author of this paper is an outspoken logical positivist, and accordingly, forsees, as the inevitable test of the system and the semantic model it embodies, the publication of a novel produced by the program, with the publisher remaining ignorant of the non-human origin of the material.

Future Plans

The system was originally written in extended ALGOL for the Burroughs B5500 computer. It is currently being reprogrammed in FORTRAN V for the Univac 1108. The special features of FORTRAN V that are not available in the more widely available FORTRAN IV language are being utilized in a way that makes them easily replaceable by special assembly language macros if the program is run under a FORTRAN IV compiler.

The task of writing rules for generating a novel will probably involve as much work as the old-fashioned method. Accordingly, additional researchers have joined the project, and plans are being made to increase the number of rule writers.

A new plot has been chosen for our first test on the 1108 computer: the simulation of the origin and execution of a street-party in the Madison Wisconsin Mifflin Street student ghetto in 1970, that resulted in a student-police riot.

The 1108 version of the system will also offer a number of important additions to the rule formulation programming language.

One is the ability to provide attribute relations with dynamically modifiable quantitative values. This will enable the system to model quite fine degrees of intensity, such as happiness or unhappiness on a scale of 0 to 100. The device may also be used as a simple counter and will enable rule writers to avoid the cumbersome intoxication-state rules in the model described earlier.

A significant addition is the capability for the system to create node classes dynamically. If properly utilized, it will offer a device for incorporating memory

of network relations that have ceased to exist. The device also permits the formulation of plans for actions more than one time frame in advance. Other features include variable time scales for different rules and, accordingly, variable subdivision of the major time frame into subtime frames. Interestingly, this feature creates problems that tempt one to assume Einstein's view of time and space; discussion of that topic is reserved for another paper.

A Long Term Goal

Eventually, we hope to provide each character in a model with a private semantic network as well as access to a public one.

Given this, we plan to generate conversations between characters that result in the transfer of information from one private semantic domain to another through the medium of the text of the conversation. It is anticipated that the achievement of this goal will include a parser that utilizes physical and socio-cultural data as well as semantic and syntactic information. Indeed, we feel that only such a parser can ever succeed in handling the ambiguity potentially inherent in natural language. While such a parser is difficult to build for the real world, the current system will have a totally controlled, limited but heterogeneous universe available for the task.

Information Retrieval Applications

We believe that a successful working version of this system will lend itself to applications in all phases of information retrieval on a dynamically changing data base insofar as that data base may be represented in the form of a semantic network, and insofar as one wishes to retrieve information on that data base in the form of context-sensitive language narrative.

Bibliography

1. Dahl, Roald, The Great Automatic Grammatisator, in Someone Like You, Secker & Warburg, Ltd. London 1954.

2. Klein, S. and Simmons, R. F., Syntactic Dependence and the Computer Generation of Coherent Discourse. Mechanical Translation, Vol. 7, No. 2 August 1963.

3. Klein, S., Syntactic Dependency and the Determinination of Meaning in Written English. Automation and Scientific Communication, Part I, American Documentation Institute, October 1963.

4. Klein, S., Automatic Paraphrasing in Essay Format. Mechanical Translation, Vol. 8, Issues 3 & 4 combined, August-December, 1965.

5. Klein, S., Control of Style with a Generative Grammar. Language, Vol. 41, No. 4, October-December, 1965.

6. Klein, S., Leiman, S. L. & Lindstrom, G. E., DISEMINER: A Distributional-Semantics Inference Maker. Technical Report, Carnegie Institute of Technology 1966. & Journal of Computer Studies in the Humanities & Verbal Behavior, Vol. 1, No. 1, January 1968.

7. Lamb, Sydney, Outline of Stratificational Grammar, Georgetown University Press, Washington, D. C., 1966.

8. Quillian, Ross, Semantic Memory, in Semantic Information Processing, M. Minsky, MIT Press, Cambridge, Mass., 1968.

9. Schank, Roger, Understanding Natural Language Meaning and Intention, Presented at 1971 International Conference on Computational Linguistics, Debrecen, Hungary, Sept. 4-7, 1971.

10. Simmons, R. F., Klein, S., and McConlogue, K., Indexing and Dependency Logic for Answering English Questions. American Documentation, Vol. 15, No. 3, July 1964.

11. Warshall, S., A Theorem on Boolean Matrices. Journal of the Association for Computing Machinery, Vol. 9, No. 1, January 1962.