PARTITIONED NETWORKS FOR THE MATHEMATICAL MODELING OF NATURAL LANGUAGE SEMANTICS

by

Gary G. Hendrix

Technical Report NL-28
Department of Computer Sciences
The University of Texas at Austin
December 1975

ABSTRACT

The main concern of this dissertation is to advance a modeling scheme for use in artificially intelligent natural language understanding which is capable of encoding knowledge about the world in a uniform, precise and easily manipulatable form.

As a prerequisite to understanding, fundamental concepts such as time, space, relation and change are formalized in terms of set-theoretic constructs. Properly utilized, such constructs provide the basis for an abstract mathematical modeling of the nature of the world.

Certain human-like organization schemes are imposed upon the set-theoretic models so that knowledge becomes grouped into bundles which are meaningful and convenient units for discourse. The organization schemes include a hierarchical classification of objects and the packaging of sets of related changes into units called processes. Processes are defined by "process automata," structures capable of encoding discrete, continuous and parallel change. In conjunction with the hierarchy, process automata may be used to describe certain objects (both events and physical objects) at multiple levels of detail. Further, such automata may encode linguistic processes such as

parsing and generation procedures.

To facilitate use by computers, the set-theoretic modeling scheme is mapped onto a network representation. This network representation provides easily accessible cross-linkage between semantically related pieces of information, thus facilitating algorithms which interrogate the information modeled in the network. A special network partitioning mechanism is used to delimit the scopes of quantified variables, to distinguish hypothetical and imaginary situations from reality, to encode the multiple, alternative worlds considered in planning, and to focus attention at particular levels of detail.

The modeling scheme uses a single paradigm to encode information relating to all types of objects including physical objects, situations, times, events, categories and processes.

ACKNOWLEDGEMENTS

This document is a reprint of the author's Ph.D. Dissertation. The research reported herein was supported in part by NSF Grant GJ509X. I am indebted to my committee chairman, Professor Robert F. Simmons for guidance and hours of productive discussions and to the members of my committee, Dr. Norman Martin, Mrs. Angus (Erna) Pearson, Dr. Laurent Siklóssy and Dr. Raymond Yeh for their critiques. My thanks also to Don Walker, Ann Robinson, Barbara Deutsch and Rich Fikes for their contributions toward a partitioned network facility for the SRI speech understanding system.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

One of the most fundamental tasks in the construction of an artificial intelligence system is the selection and implementation of a scheme for representing knowldege. Experience has proven that the representation scheme influences not only the implementation of various algorithms, but also the designer's very conception (or metaphysical interpretation) of a problem. The main concern of this dissertation is to advance a modeling scheme for use in artificial natural language understanding which is capable of encoding knowledge about the world in a uniform, precise and easily manipulatable form.

Several partially successful techniques have, of course, already been developed for representing various aspects of knowledge, a brief summary of pointers into the relevant literature being provided in Chapter 2. However, many problems relating to the modeling of time and change have never been adequately resolved. Further, how best to organize information for efficient question-answering remains an open problem.

In designing a modeling scheme for use in natural language understanding by computers, attention must be paid

to three principal considerations. First, the modeling scheme must be based on some formalism which is capable of adequately characterizing all the subjects of interest. This basic formalism should be linked to a well understood body of mathematics so that the legacy of existing mathematical theory and technique is immediately applicable in the model. Since it seems reasonable to assume that an artificial intelligence system will most easily be able to converse with humans if its organization of knowledge reflects a human-like conception of the world, the building blocks of the model should correspond to units of knowledge which are familiar to an average person (although the blocks may actually be quite complex structures). That is, the building blocks should correspond to such things as physical objects, events, processes and circumstances which are talked about in everyday discourses and should not (to the exclusion of the aforementioned) correspond to either exotic phenomena known only to scientists or mathematical structures.

Second, the modeling scheme must impose some type of organization on the modeled information so that it may be efficiently stored and easily manipulated by natural language understanding and question-answering procedures. Organization strategies include such techniques as bundling related pieces of information into larger conceptual units

and generalizing particular pieces of information into quantified expressions and abstractions. Since, depending upon the point of view of the processing algorithm, a given piece of data may be interpreted in various ways, the modeling system would do well to represent all information through a uniform paradigm, allowing system processes to interpret the data in the way which seems most appropriate for the task at hand.

Third, the modeling system must be structured in such a way that it may be conveniently used by a digital computer.

Since these three basic considerations interact strongly with one another, a fourth consideration becomes the coordination of the original three.

These basic considerations have guided the development of the modeling scheme discussed herein. Further, an attempt has been made to build upon previous work by amalgamating important features from predicate calculus, set theory, taxonomic systems, semantic networks, robot planners, automata theory, simulation systems, and programming languages providing context maintenance facilities. As a preview to the main body of this work, a few of the more interesting features of the modeling system may be summarized by the following items:

- Because of their explicit interlinkage between

semantically related items, semantic networks have been selected as the basic representation structure.

- Conventional network notions have been augmented by introducing a context-like mechanism which partitions a network into regions called "spaces." These spaces have manifold applications.

- The conventional context mechanism has itself been generalized to allow arbitrary partial orderings of spaces (contexts).

- Spaces are used to delimit the scopes of quantified variables. Through this usage, the full flexibility of predicate calculus quantification is made available in networks.

- Spaces are used to distinguish alternative worlds such as the hypothetical worlds needed for planning and for answering hypothetical questions. Alternative worlds also include the fantasy worlds of dreams and wishes and of lies, stories and alternative beliefs.

- Spaces help encode certain logical connectives which are somewhat troublesome in conventional networks. These include disjunction and implication.

- Spaces allow both the semantics and syntax (of one reading) of an English sentence to be represented simultaneously in one integrated structure. This structure, which is very useful in discourse analysis, shows the syntactic parse tree superimposed on the semantic network translation. Alternative parse hypotheses produced during the parsing process may share portions of a translation which they have in common.

- Spaces may be used to focus attention on particular areas of the network which are "relevant" to the current conversation.
- Information contained within a space which is not currently "activated" is effectively invisible to the rest of the system.
- Using quantification, intentional definitions may be given for classes. For classes of situations (called relations elsewhere), the components of the situation and their possible ranges of values may be given.
- Classes are carefully distinguished from their members.
- Classes are arranged into a hierarchical taxonomy with subclasses inheriting properties from superclasses.
- Process plots are represented in terms of "process automata." (The instantiations of a given plot constitute a category of events.) A process automaton may be used to "parse" a sequence of changes to determine whether it constitutes the enactment of a particular plot.
- Process automata may depict both discrete and continuous changes and show the decomposition of processes into subprocesses.
- Time is treated as a continuous phenomenon (but no claims are made concerning the adequacy of this treatment in the light of modern physics).
- Using spaces and process automata, many types of entities may be depicted at multiple (even recursive) levels of detail.

- Process automata may also be used to describe test procedures for determining membership in selected classes (including fuzzy categories).
- For robots, process automata may be linked to "black boxes" which actually perform operations. These boxes provide a bridge between the network and physical reality.

The reader who wishes to gain additional perspective before launching into the body of this work may find it useful to browse through the "Concluding Remarks," Chapter 21, in which the major components of the modeling system are summarized. For convenience, a glossary of terms and symbols is included.

Chapter 2

Brief Summary of Pointers into Relevant Literature

The following paragraphs provide a number of references to papers which underpin the work reported herein. Citations range over early computer representations of semantic information, semantic networks, predicate calculus, world modeling systems, automata theory and contextual mechanisms.

The pioneering attempts to construct artificial intelligence systems include GPS, the General Problem Solver devised by Newell, Shaw and Simon (1960), Samuel's checker-playing program (1963) and the early chess-players, examples of which are discussed by Newell *et al.* (1958). The internal representations of knowledge used in these and other early systems were almost always *ad hoc* and most representations (that of GPS being a notable exception) were designed only to accommodate a very narrowly defined task. Other early systems with highly specific representations include BASEBALL, a natural language question-answering system devised by B. F. Green *et al.* (1963) and the inferential memory of R. K. Lindsay's SAD SAM program (1963). The numerical equations used by Bobrow's STUDENT (1968) constitute yet another special purpose representation.

The last several years have been marked by the coming of age of more general purpose representations. Semantic networks provide the representative fiber of such natural language systems as Raphael's SIR (1968), which is perhaps the earliest network system, Quillian's semantic memory (1968) and TLC (1969), Simmons' PROTOSYNTHEX III (Simmons et al., 1968) and subsequent systems (Simmons, 1973), Brown's SOPHIE (1974), the Rumelhart, Lindsay and Norman systems (1971, 1973) and Kay's MIND (1973). The conceptual dependency structures of Schank (1972) and followers, notably Rieger (1974), are networks also and thus fall within this general category. Other noteworthy research in net representations includes the work of Shapiro (1971), Winston (1970), Sandewall (1970), Anderson and Bower (1973), Palme (1971), Mylopoulos et al. (1973) and Schubert (1974). A recent discussion of semantic networks and their associated problems is provided by Woods (1975).

Predicate calculus has enjoyed an extended popularity as a representation scheme, being used in such computer systems as J. A. Robinson's (1965), C. Green and Raphael's (1968), C. Green's (1969) and in Sandewall's PCDB (1971a). Predicate calculus, of course, predates the computer age. Quine (1951) dates mathematical logic from the work of George Boole (1847) in the middle of the last century. Out of the basic predicate calculus representation have grown

the semantic system of Montague (1974) and the techniques of "world modeling" or "robot modeling." Systems using robot modeling schemes include Winograd's SHRDLU (1972), the STRIPS system of Fikes, Hart and Nilsson (1972a), Siklóssy and Dreussi's LAWALY (1973), the Hendrix world simulator (1973b) and the recent procedural networks of Sacerdoti (1975).

The "world modeling" systems have had considerable influence on the approach to process knowledge presented here as have conventional automata theory (exemplified by Salomma, 1969, and Hopcroft and Ullman, 1969) and Wood's AFSTN system (1970). Conventional system simulation texts, such as Gordon (1969), also provide insights into the nature of processes as do such philosophies as Whitehead (1929).

The idea of network partitioning has its roots in the context mechanisms of such programming languages as PLANNER (Hewitt, 1971), CONNIVER (McDermott and Sussman, 1972) and QLISP (Reboh and Sacerdoti, 1973).

DIVISION I

Rethinking the Rudiments of Modeling

Chapter 3

Goals of Division I

As stated in the introduction, it is desirable that any formalisms which are developed should be linked to a well understood body of mathematics such as mathematical logic. It is assumed that the reader is familiar with logic and with such works as Quine (1951). In what follows, terms such as "set" and "relation" are intended to carry the same meaning as in texts on logic. However, since computer scientists (including the author) are prone to use these terms rather loosely, this division attempts to clarify the meaning of modeling nomenclature and to establish the relationship between mathematical entities and the objects which they model.

Further, the nature of certain aspects of the world, particularly the phenomena of time and change, are pondered to gain some insight into how they may be modeled more realistically.

Chapter 4

Objects

To begin the discussion of fundamentals, consider the notion of an object. Although most modeling systems place restrictions on what may be classified as an object, the concept of object used herein is to be quite broad and will include such entities as relations, sets, n-tuples, situations, colors, events and instants in time. To establish a common denominator for all the aspects of existence, the notion of object may be characterized by saying that there is nothing which is not (at least at some time) an object. Logical constraints presented below will force certain objects (such as extremely large classes) to be excluded from consideration. However, it is unlikely that any objects needed for the understanding of common, natural language discourse will be excluded.

4.1 The universal set

In order to talk about objects, it would seem convenient to group all objects into a single universal set U and to assume that all objects are elements of U and all elements of U are objects. However, considerable care must be taken in considering the nature of U. Whole sets, being objects themselves, would logically be elements of U. Even

U, being a set and hence an object, would necessarily be an element of itself. As is well known (see Quine), this situation leads to Russell's paradox.

To circumvent the paradox, U might itself be excluded from being an object (and hence an element of itself). This is roughly the approach of Von Neumann (1926) and Bernays (1937). Alternatively, as shown by Quine, U might be redefined as being the set of all entities which belong to some set. Quine has shown that a U defined in this fashion may indeed be an element of itself (theorem 210 of Quine (1951)) and that if set s is an element of U, then the complement of s will likewise be an element (theorem 274). Clearly, Quine's redefinition of U excludes certain entities from consideration as objects. However, those entities which are excluded are hopefully of little or no pragmatic consequence to a system intended to understand normal discourse concerning more common objects.

4.2 The time set

The set T, the set of all instants of time, is both a subset and an element of U. This set is of great theoretical importance since it is customary to think of all objects in relationship to this special set. That is, each object in U is regarded as having (contemporary) existence (either in the real world or in some hypothetical world)

with respect to some continuous subset of T. In particular, each element of T is regarded as having existence only with respect to itself.

Using set T as an index, set U may be fragmented into a family of sets U_t where each U_t is the set of all objects existing at time t. The intersection U' of all U_t, $\cap U_t$, is the set of all objects which always exist. This set of objects is time independent. The union $\bar{U}$ of all U_t, $\cup U_t$, is the set of all objects which ever existed or ever will exist. Set $\bar{U}$ is recognized as the original set U. The set U - U' is the set of all objects which have the property of being created and/or destroyed.

If $x \in U_t$, then x is said to have <u>contemporary existence with respect to time t</u>. If $x \in U$, then x is said to have <u>historical existence</u>. If $x \in U'$, then x is said to have <u>omnichronic existence</u> since x exists at all times.

Set T itself has omnichronic existence (by virtue of being a set, as explained shortly). However, the elements of T do not have this property. Each instant of time has contemporary existence for only a fleeting moment. In this regard it is interesting to note that if $t_1, t_2 \in T$, then $t_1 \in U_{t_2}$ iff $t_1 = t_2$.

A set which lends itself quite nicely to the representation of the knowledge of objects in the set U*, a subset of $T \times U$. A pair $(t, x) \in U^*$ iff $x \in U_t$. Now U* is

itself time invariant (i.e., $U^* \in U'$), but it encodes a complete description of contemporary existence. That is, if $x \in U$, then for exactly those instants t with respect to which x has contemporary existence, $(t, x) \in U^*$. Stated in another way, U^* is a time invariant object which encodes the temporal existence of all objects.

4.3 Properties of sets

Since the notion of a set or class plays such an important role in what follows, it is necessary to clarify certain properties of sets which have become confused in informal usage.

4.3.1 The constancy of sets

The first property to be clarified is the constancy of sets. A given set is simply not subject to change. One sometimes hears such contrary sounding phrases as "adding new elements to a set S," but the complete phrase should be "adding new elements to a set S to produce a new (i.e., to specify a different) set X."

A would-be obstruction to the acceptance of the constancy of sets is posed by such common usages as "the set of men at the top is constantly changing." The problem here is the interpretation of "the set changes." In "a man's face is changed from sad to happy," the old face has a change of structure. Since the only structure associated

with a set is the relationship of the set to its elements, the set's structure cannot change without producing a new set. That is, if a set could change its structure, it would not be the same set after the change as it was before. In "the car's spark plugs were changed," the old spark plugs did not change structure. Rather, the old spark plugs were replaced by (exchanged for) new plugs. Under this interpretation, the process of "changing a set" maintains the integrity of the original set. The old set is simply replaced by a new set.

Perhaps the confusion in this matter has arisen from the use of variables to denote sets. Suppose the variable "S" originally stands for the set {JOHN, SAM}. Then when one talks about the set S, one means the value of variable "S" which is the set {JOHN, SAM}. If the binding of "S" is changed to {ELMO}, then it appears that "set S" has changed. But, in fact, it is the binding of variable "S" which has changed and not the set {JOHN, SAM}.

4.3.2 The omnichronic existence of sets

A second property of sets which requires some clarification is the omnichronic existence of sets. In some quarters there is a tendency to believe that the temporal existence of a set is somehow tied to the temporal existence of its elements. That is, it may at first seem

tempting to believe that a set S has contemporary existence with respect to exactly those times t with respect to which all the elements of S have contemporary existence. For example, the set {John, Sam, Philip} might be thought to have contemporary existence only with respect to those times in which John, Sam and Philip are simultaneously alive. However, this notion is shattered if applied to set D, the set of all days in the year 1970. Clearly, no two of the elements of D have simultaneous contemporary existence and, if the temporal existence of D were truly dependent on the temporal existence of its parts, D would have no contemporary existence. That is, D could never have existence.

There is good reason to believe that the null set has existence, yet it has no members. Hence its existence cannot in any way be member dependent.

Since the temporal existence of a set cannot depend on the temporal existence of its members, sets may be assumed to enjoy omnichronic existence.

The assumption (or convention) that sets are omnichronic is important in the modeling scheme because there must be something that is constant through time and that can be used to help record history. By their constancy, sets provide such an anchor for the rest of the system.

Chapter 5

Relations and Situations

A relation may loosely be defined as any interconnection existing between a subset of a group of objects which are participants in a unifying condition or situation. A more formal definition of relation has been well developed in the theory of logic and is familiar to all students of mathematics. According to this definition, a formal relation R is a set of n-tuples, i.e., a subset of the cross product of n sets. For example, the mathematical relation "is the counting number succeeding" is the set S of pairs $S = \{(1, 0), (2, 1), (3, 2), \ldots\}$. S is a subset of the cross product of the positive integers and the non-negative integers. For each pair (X, Y) in S, it may be said that X "is the counting number succeeding" Y. Conversely, if X "is the counting number succeeding" Y, then (X, Y) is necessarily a pair in S.

From the outset it must be observed that the informal idea of a relation refers to an "interconnection" while the formal notion refers to sets of n-tuples. Given this observation, it is important to note that n-tuples and sets of n-tuples are objects in themselves, but these objects are typically used to <u>model</u> situations and sets of situations.

Also worth noting is the tendency to associate the informal notion of a relation with a single situation while associating the formal notion of a relation with a set of situations. For example, Lafe being Grant's father is looked upon as being an acceptable informal relation, but the pair (Lafe, Grant) is only an element of the formal father-son relation set.

The significance of these remarks is to emphasize that formal relations are merely vehicles for encoding classes of situations. Throughout this work, emphasis will be given to the notion of a situation as opposed to the notion of an n-tuple. For convenience, the term "relation" will be used only in the formal sense while the term "relationship" will be used more loosely.

5.1 Lay relations

Since, according to the formal definition, relations are sets of n-tuples, a formal relation must be invariant. (This follows from the fact that all sets enjoy omnichronic existence.)

The invariance of formal relations imposed by the set-based nature of their definition would seem to be at odds with the everyday, common sense notion of relation which is used by the laity. Lay relations are often subject to variation as is seen in the following two examples.

Consider a microworld containing only a creature

and two boxes, BOX1 and BOX2. At some time t_1, BOX1 may be on top of BOX2. Hence, the lay relation "is on top of" holds between BOX1 and BOX2. Between times t_1 and t_2, the creature may remove BOX1 from the top of BOX2 and place BOX2 atop BOX1. Thus, at time t_2, it is no longer true that the lay relation "is on top of" holds between BOX1 and BOX2. Rather, "is on top of" holds between BOX2 and BOX1. The lay relation "is on top of" has changed with time.

A man on a riverboat on the Mississippi River near New Orleans might ask the riverboat captain how wide the river is. The captain might reply that the river is 1½ miles wide. Hence, near New Orleans, the lay relation "width" holds between Mississippi and 1½ miles. Two boys wading in the Mississippi near its source might measure the river's width and determine that it is only 8 feet wide. Hence, near the source, the lay relation "width" holds between Mississippi and 8 feet and not between Mississippi and 1½ miles. The lay relation "width" has changed with location.

A lay relation (such as "width" or "is on top of") is intended for use in a <u>context</u> or <u>setting</u> in which numerous background conditions are assumed. The width of a river varies with the position along that river. Taking this into account, the lay relation "width" is a kind of shorthand notation intended for use in situations where the

position at which the width is measured is understood. (To change the position, one may say that the river R is x feet wide at some location L.) The lay relation "is on top of" changes with time and is intended for use in contexts where the referent time is assumed to be understood.

In general, lay relations are subject to variation because they do not fully specify the "conditions or situations" they are intended to express and because, unlike mathematical constructs, they are not isolated from the objects they are intended to express by a formal system and hence are subject to constant review. The width relation neglected to specify the place. (Further, since at any location along the river, the width may change depending on rains, etc., the time of measurement is also implicitly needed.) The "is on top of" relation failed to specify the time component of the condition.

5.2 Formalizing lay relations

Hopefully, it is intuitively clear that if sufficient background information is added to a lay relation (i.e., if certain background parameters are elevated to the position of explicit relation components), then the lay relation may be expressed by a formal relation which adequately characterizes the underlying situations and circumstances in all contexts. To understand how this process of elevating background information may be accomplished, the

following strategy will be used. A formal relation R which is fully known will be presented explicitly. Then a lay relation r which omits parts of the information will be extracted and discussed. Finally, the original formal relation R will be reconstructed from r.

As the explicit formal relation, let

$$R = \{(X, 1, B),$$
$$(Y, 2, B),$$
$$(X, 2, A),$$
$$(Y, 3, A)\}.$$

The reader may render this abstract example more concrete if he imposes some comfortable interpretation upon the symbols. R may be thought of as expressing the relationship of location in a particular microworld. This microworld contains two objects, X and Y, and three locations, numbered 1, 2 and 3. Life in the microworld is very boring. In all of its existence only one event E has ever taken place. During event E the objects changed their positions. All times Before the change are referred to by B, while all times After are referred to by A. Thus object X was at place 1 before the event, but after the event X was in place 2.

In the context of times before event E, it is fair to say that X is at point 1 and Y is at point 2. Abstractly, if B is the assumed third component of the formal

relation R, then it is meaningful to say that some lay relation r holds between X and 1 and between Y and 2.

Clearly, the lay relation r changes if the background condition fixing the third component of the formal relation is altered. For example, if the third component is changed from B to A, then lay relation r no longer holds between X and 1. Rather r holds between X and 2.

The behavior of r exhibited in the above example is very much like the behavior of a variable. As background assumptions change, the value of a variable is altered. Likewise, as background assumptions change, the "value" of lay relation r is altered.

The "value" of lay relation r in background q corresponds to a formal relation R_q. For example, let b be the background which assumes that the attribute a_3 associated with the third component of the elements of R is assigned the value B. (Concretely, the time is assumed to be before the event E.) In context b, the "value" of r is $R_b = \{(X, 1), (Y, 2)\}$ because, in context b, lay relation r holds between X and 1 and between Y and 2. Note that R_b is invariant. The value of a_3 has no effect on R_b. However, whether or not R_b is assigned as the "value" of lay relation r is very much dependent on the assumed value of a_3.

Lay relation r is intended for use in contexts where the value of a_3 is fixed. Each possible value V of

a_3 corresponds to a context v in which the value of a_3 is assumed to be fixed at V. Associated with each such context v is a formal relation R_v which is the "value" of r in that context. From the various R_v it is possible to construct a new formal relation R^+ which combines all the information contained in the many R_v. Let

$$(V, u, w) \subset R^+ \text{ iff } (u, w) \in R_v.$$

To determine R^+, first list the various R_v.

For V = B,

$$R_b = \{(X, 1), (Y, 2)\}.$$

For V = A,

$$R_a = \{(X, 2), (Y, 3)\}.$$

Hence,

$$R^+ = \{(B, X, 1),$$
$$(B, Y, 2),$$
$$(A, X, 2),$$
$$(A, Y, 3)\}$$

which encodes exactly the same information as the original formal relation R. (That is, there exists a one-to-one, onto correspondence between R and R^+. $(i, j, k) \in R$ iff $(k, i, j) \in R^+$.)

In general, an arbitrary lay relation r assumes that a set S of attributes (such as a_3 - see Section 11.3 for a definition of "attribute") have preassigned values. Suppose lay relation r assumes values for each of m

attributes a_1 through a_m. From the set S a subset S' may be selected such that S' = $\{b_1, b_2, \ldots, b_n\}$, $n \leq m$, and if the values of b_1 through b_n are known then the values of a_1 through a_m are uniquely determined. The arbitrary lay relation r may be defined in terms of formal relations R_c where the contexts c range over all legal assignments of values to the attributes in S'.

Following the procedure outlined earlier, any lay relation r may be associated with an invariant formal relation R^+ where $(V_1, V_2, \ldots, V_n, X_1, X_2, \ldots, X_j) \in R^+$ iff $(X_1, X_2, \ldots, X_j) \in R_c$ and c is the context for which attributes b_1 through b_n take on values V_1 through V_n. Since S' may not be unique, it may be possible to capture a lay relation r by a variety of formal relations R^+.

5.3 The definition of "holding"

As might well be expected, there is a formal relationship between an informal relation r and the formal relations R_c which encode r in the various contexts c. Let J be this relation. Thus

$$(c, r, R_c) \in J$$

iff R_c encodes r in context c.

The notion of a lay relation "existing" or "holding" among a group of objects $X_1, X_2, \ldots, X_n$ in a particular context c may be formalized by defining formal relations H'_c. (H'_c means holds in c. H_c is defined below.)

For every context c and for every lay relation r, let

$$((X_1, X_2, \ldots, X_n), r) \in H'_c$$

iff

$$(c, r, R_c) \in J$$

and

$$(X_1, X_2, \ldots, X_n) \in R_c.$$

Thus in context c the lay relation r may be said to hold or exist among objects $X_1, X_2, \ldots, X_n$ iff $((X_1, X_2, \ldots, X_n), r) \in H'_c$.

Of course, formal relations by their very nature are invariant. In all contexts it is correct to say that a formal relation R holds among a group of objects $X_1, X_2, \ldots, X_n$ iff $(X_1, X_2, \ldots, X_n) \in R$. Thus, in any context c, R holds among $X_1, X_2, \ldots, X_n$ iff $(X_1, X_2, \ldots, X_n) \in R$.

Given a relation $\underline{r}$ which may either be a lay relation or a formal relation, $\underline{r}$ holds among objects $X_1, X_2, \ldots, X_n$ in context c iff

$$((X_1, X_2, \ldots, X_n), \underline{r}) \in H_c$$

where H_c is defined as follows. Let

$$((X_1, X_2, \ldots, X_n), \underline{r}) \in H_c$$

iff

either $\underline{r}$ is a formal relation R and

$$(X_1, X_2, \ldots, X_n) \in R$$

or $\underline{r}$ is a lay relation r and

$$((X_1, X_2, \ldots, X_n), r) \in H'_c.$$

5.4 Relational statements

A relational statement or simply a statement is an (n + 1)-tuple of the form

$$(\underline{r}, X_1, X_2, \ldots, X_n)$$

where $\underline{r}$ is a relation (either lay or formal) and X_1 through X_n are objects. A statement $(\underline{r}, X_1, X_2, \ldots, X_n)$ is said to be valid in context c iff

$$((X_1, X_2, \ldots, X_n), \underline{r}) \in H_c.$$

For example, in a context which assumes location to be near New Orleans, the statements

(WIDTH, MISSISSIPPI, 1½-miles) (lay statement)

and

(GREATERTHAN, 5, 4) (formal statement)

are valid. In the same context the statement

(WIDTH, MISSISSIPPI, 8-feet)

is not valid. Note that if a statement involving a formal relation is valid in one context, it is necessarily valid in every context.

5.5 Clusters

The definition of formal relation used previously is based on n-tuples. That is, a relation is defined as a set of n-tuples.

Relations may also be defined in terms of clusters,

a generalization of the n-tuple. An n-tuple $N = (x_1, x_2, .., x_i, ..., x_n)$ has exactly one i^{th} component, namely x_i. A cluster $C = (x_{1,1}, x_{1,2}, ..., x_{1,m_1}; x_{2,1}, ..., x_{2,m_2}; ...., ; x_{n,1}, ..., x_{n,m_n})$ may have m_i components of the i^{th} type. The order of the i^{th} components is of no importance. Hence, (1, 2 ; 3, 4) is the same binary cluster as (2, 1 ; 3, 4) which is the same as (2, 1 ; 4, 3).

An alternative formal definition of "relation" would make a relation a set of clusters. These two definitions are really equivalent. Rather than present a formal proof of this fact, consider the following example.

Let $N{:}BETWEEN_{1\text{-}4}$ be the between relation defined in terms of n-tuples for integers between 1 and 4. That is,

$$N{:}BETWEEN_{1\text{-}4} = \{(2, 1, 3), (2, 3, 1), (2, 1, 4), (2, 4, 1), (3, 1, 4), (3, 4, 1), (3, 2, 4), (3, 4, 2)\}.$$

In general $(x, y, z) \in N{:}BETWEEN_{1\text{-}4}$ if x is between y and z (and x, y and $z \in \{1, 2, 3, 4\}$). Note that y may be either less than or greater than z, producing two different n-tuples.

$C{:}BETWEEN_{1\text{-}4}$ describes this same situation by using clusters.

$$C{:}BETWEEN_{1\text{-}4} = \{(2;\ 1,\ 3),$$
$$(2;\ 1,\ 4),$$
$$(3;\ 1,\ 4),$$
$$(3;\ 2,\ 4)\}.$$

In general $(x;\ y,\ z) \in C{:}BETWEEN_{1\text{-}4}$ if x is between y and z. Although they look different in print, the clusters $(x;\ y,\ z)$ and $(x;\ z,\ y)$ are identical. Thus, by using clusters, BETWEEN may be realized with only half as many elements.

It is hopefully clear that any relation realized through n-tuples may be realized through clusters and vice versa.

Chapter 6

The Nature of Time and Change

Now that the concept of a lay relation has been established, the associations existing between time, change and relation may be addressed.

6.1 Some observations concerning time

As illustrated by the tense systems of Western languages, the notion of time is fundamental to the Western conceptualization of the world. Rather than attempt a definition of time, it will be sufficient for present purposes to make a few observations. (These observations ignore modern physics, but so does language.)

The first observation about time is that it involves an indexing scheme. As an index, time may be modeled by an index set T, the indices (elements) of T being called instants. Since time is thought to be an ordered phenomenon, the index set T must be ordered (by a relationship called "BEFORE"). Since time is thought to be continuous, the index set must be dense. (That is, between any two instants t_1 and t_2 in the ordering BEFORE, there are an infinite number of other instants.) Further, if time is thought of as unbounded, set T contains neither a least nor a greatest element.

Another observation is that time is related to the notion of existence. In particular, all objects are thought of as existing over some time interval (i.e., over a continuous subset of T). Since all objects are related through existence to intervals of time and since the instants of time are ordered, the time index may be used to order the existence of other objects. It is in this ordering of the existence of objects that time finds both its utility and ultimate meaning. Specifically, as will be explained more fully in the remainder of this chapter, the sequence of change is an ordering of the existence of certain types of lay relationships.

6.2 The variation of lay relations with respect to time

Pragmatically, the most important lay relations are those which assume a specified time but which make no further background assumptions. Such relations are called time-only lay relations or, more briefly, chronorelations. The nature of the variation in the value of a chronorelation with respect to time is of particular importance.

Variations in lay relations with respect to time manifest themselves in two forms: discrete change and continuous change. A relationship whose variation is always discrete is the relation "owns" or "possesses." Assume that "owns" is designated at times t by formal relation sets P_t. Since changes in ownership may be thought of as

occurring at discrete points in time, changes in the designation of P_t occur at discrete instants. That is, if there is only one change in ownership in the closed time interval $[t_1, t_3]$ and that change occurs at time t_2, then $P_{t_x} = P_{t_y}$ for all t_x and t_y in the interval $[t_1, t_2)$ and $P_{t_x} = P_{t_y}$ for all t_x and t_y in the interval $[t_2, t_3]$, but $P_{t_1} \neq P_{t_3}$.

(Note the use of the half open interval $[t_1, t_2)$ above. If there is a change from state S_1 to state S_2 at time t, it is assumed (as a convention) that state S_2 was reached at t and that state S_1 was left just before t. Thus, S_1 held for some interval $[t', t)$ and S_2 will hold for some interval $[t, t")$. For example, suppose John bought a candy bar at time t_1, that he sold it to Tom at t_2 and that Tom ate it in one gulp at t_3. Then we will say "John owned the candy bar from t_1 to t_2," when in fact John only owned the bar over $[t_1, t_2)$. If asked "who owned the bar at t_2?" the answer should be "Tom did." Further, we may say "Tom owned the bar from t_2 to t_3.")

A relationship such as AT is subject to variation of the continuous type. If "at" is defined over a coordinate system, then at some instant t, a physical object Q might be "at" point (x, y). Thus, at that instant, $(Q, (x, y)) \in AT_t$. If Q is moved to a new point (v, w), then Q must pass through a continuous sequence of points

connecting (x, y) and (v, w). If Q is moved from (x, y) to (v, w) in one continuous motion over a time interval I, then for any two distinct elements t_1 and t_2 of I, $AT_{t_1} \neq AT_{t_2}$. Thus, "at" is in a continuous state of flux over I. (Note that "at" relationships may also undergo discrete change. For example, the border of a country may change in an instant with the signing of a treaty.)

6.3 The invariant mathematical relation associated with a chronorelation

In general, each chronorelation r has an invariant formal counterpart R*. If r is designated at times t by formal relations R_t, let R* be the smallest set containing all (n + 1)-tuples $(t, a_1, a_2, \ldots, a_n)$ such that $(a_1, a_2, \ldots, a_n) \in R_t$. The relation R* is called a star relation. By considering the notion of time within the relation itself (i.e., by adding a time designator to the relation n-tuples) the external variance of a chronorelation is removed. The resulting relation is invariant.

The similarity of star relations and the set U* (see glossary) is indicated by the following parallel formulas.

$(t, x) \in U^*$

iff $x \in U_t$

$(t, a_1, a_2, \ldots, a_n) \in R^*$

iff $(a_1, a_2, \ldots, a_n) \in R_t$

As would be expected, the star relations provide a convenient method for recording history. For example, the fact that ROBOT1 was AT point (10, 20) at time t_1 may be recorded by asserting that the triple (T1, ROBOT1, (10, 20)) is an element of the set AT*. The fact that ROBOT1 was AT point (50, 60) at time t_2 does not alter the truth of (T1, ROBOT1, (10, 20)) $\in$ AT*. That is, the statements

(AT*, T1, ROBOT1, (10, 20))

and

(AT*, T2, ROBOT1, (50, 60))

are universally valid whereas

(AT, ROBOT1, (10, 20))

is valid only in some contexts. For example, the latter is valid in contexts fixing time at t_1, but is not valid in contexts fixing time at t_2.

To concentrate the information encoded in R* relation sets and the object encoding set U*, sets of the star-star variety may be used. A triple (t_1, t_2, x) is an element of U** if and only if $I = [t_1, t_2)$ is a continuous half-open interval such that

1) for all $t \in I$, $x \in U_t$
2) if I' is a continuous interval such that
   a) for all $t \in I'$, $x \in U_t$
   b) $I \subseteq I'$

   then $I = I'$.

Analogously, $(t_1, t_2, a_1, a_2, \ldots, a_n) \in R^{**}$ iff

1) for all $t \in I = [t_1, t_2)$,
   $(a_1, a_2, \ldots, a_n) \in R_t$
2) if I' is a continuous interval such that
   a) for all $t \in I'$, $(a_1, a_2, \ldots, a_n) \in R_t$
   b) $I \subseteq I'$
   then $I = I'$.

To see how a star-star relation may encode relational information, consider the following simple example. Suppose John owned a yoyo from time t_1 to time t_2. Suppose further that John sold the yoyo to Sam at time t_2. Sam then owned the yoyo from time t_2 to t_3. Using star-star relations, the facts of ownership may be recorded as

$(t_1, t_2, \text{JOHN}, \text{YOYO}) \in \text{OWNS}^{**}$

$(t_2, t_3, \text{SAM}, \text{YOYO}) \in \text{OWNS}^{**}$.

This yoyo example involves a discrete change. Continuous change is somewhat more difficult to record and is deferred to Section 9.3.

6.4 Change as a phenomenon of limited perspective

The nature of change is perhaps the most elusive aspect of any metaphysics. Change corresponds roughly to the passing from one set of circumstances and situations to another. Any variation of circumstances and situations is apparently, however, only a localized manifestation (or

interpretation) of a broader, encompassing phenomenon.

To see this, consider the gradient of temperature along the road to hell. As a demon walks along the road (perhaps returning from an earthly mission in a computer program) he may notice an increase in temperature as he approaches the fiery pit. Even though the temperature at each point along the road remains constant, the demon will feel that the temperature is rising. In such an instance, the global concept of temperature is fixed while the local concept (the demon's concept) is undergoing change. That is to say, from a remote, detached point of view, it seems convenient to describe the temperature by a relation set TEMP composed of pairs (d, k) where d is the distance from hell and k is the temperature (in degrees Kelvin). (Over some appropriate range of distances it may be that

$$k = \gamma / d^2$$

where γ is a constant.) TEMP constitutes a formal relation and is immune to change.

The demon's concept of temperature is limited to his feeling of how warm or cold he is at any one moment. This limited concept of temperature may be described in terms of a simple unary lay relation <u>temp</u>. If the demon is in some context c (where c specifies the demon's position along the road) then the temperature k of the demon is given by

$$((k), \underline{temp}) \in H_c$$

(H_c is defined in Section 5.3). That is, (k) is an (the) element of the unary formal relation encoding temp in context c.

The relation temp is unstable relative to relation TEMP. In other words, temp may vary while TEMP remains constant. In terms of the example, when the demon changes position, he notices a change in temperature (a change in temp). It is hopefully clear that from a global perspective nothing has actually changed (with the possible exception of the demon's location). The demon, having a limited perspective, does indeed perceive a change in temperature, but the demon's perception is the result of a narrow interpretation of the situation. The lay relation temp becomes vitalized only when attention is focused upon a particular context. Assuming that the demon may contemplate neither his future nor his past, his focus of attention is tied inextricably to his current context. As the demon's focus changes (as the context within which the demon finds himself changes) the temperature singled out by relation temp changes.

In this demonic example, temperature changes with respect to position. In general, any entity Y which changes does so with respect to some second entity X which in some way is coordinated with Y. From a detached

standpoint, there seems always to be a fixed relationship between Y and X (such as TEMP in the case of Y = k and X = d). The idea of a changing Y is introduced only when attention is focused first on one particular value of X and then upon another.

In other words, the concept of change is explainable in terms of limiting one's perspective to particular contexts. From a detached point of view, the notion of temperature is a relatively fixed notion described by the relation TEMP. The relation TEMP indicates that temperature and position are coordinated. Roughly this means that two different positions will be associated with two different temperatures. If temperature is isolated from position (i.e., if temperature is thought of in terms of temp), then the temperature becomes "ambiguous" and it is necessary to specify a particular context (which in turn fixes the position) for the isolated concept of temperature expressed through temp to have meaning. Divorced from the coordinated concept of position, the lay relation temp is subject to variation whenever attention is focused on a new context which assumes a different position.

6.5 The manifestation of change in lay relations

Change has roughly been defined as the passing from one set of circumstances and situations to another. If

circumstances and situations are altered, it follows that those lay relationships which capture the circumstances and situations must be altered. Thus, abstractly, change may be thought of as the variance in lay relations.

Formal relations, which are themselves invariant, may be used by modelers to encode facts about a dynamic world. The variable lay relations, whose variance may be thought of as an encoding of changes in the world at large, have been tied to these invariant formal relations by conventions involving contexts. To the point, recall that a lay relation r may be associated with a formal relation R_{c_1} in context c_1 and associated with relation R_{c_2} in context c_2. While r is invariant within either c_1 or c_2, r differs between c_1 and c_2. That is to say, by "passing" from context c_1 to context c_2 there is a difference in the associated formal relation. Thus, in terms of the formalisms developed thus far, change may be viewed as the alteration of lay relations accomplished by substituting one formal relation for another when "passing" the focus of attention from one context to another.

From an alternate point of view, formal relations such as the R* (Section 6.3) have been designed to envelop change. To regain a feeling of change, attention may be focused sequentially on particular contexts. While focusing attention on context c, it is assumed that a lay

relation r is given by formal relation R_c. That is, R_c is the definition (or "value in context") of r which is in the foreground. In passing the focus of attention from one context to another, the foreground definitions of lay relation r are altered.

Those changes which are of the greatest pragmatic importance are the variations in lay relations which are associated with the passing from one context to another where the contexts differ only in their background assumption of current time. These special time covariant changes are called chronofluxions.

To gain an intuitive feel for which changes are chronofluxions and which are not, consider the lay relation TEMPERATURE. As the day (time) progresses from early morning, the sun causes the temperature at a fixed location to increase. Such changes of temperature with the changing of time are chronofluxions. If temperatures are simultaneously taken at various altitudes above the earth's surface, there will be a marked decrease in temperature at higher altitudes. Such changes in temperature with differences in altitude are not chronofluxions.

(As a counterintuitive example, consider the temperature of a weather balloon as it rises from the earth's surface. It is reasonable to assume that the temperature of the balloon is a function of its altitude. This might

lead to the conclusion that changes in the balloon's temperature are not chronofluxions. However, the balloon is only at one altitude at one time. Indeed, the balloon's altitude is a function of time. Since the balloon's temperature is a function of altitude which is in turn a function of time, it follows that changes in the balloon's temperature are chronofluxions.)

To formally define the concept of chronofluxion, it is first necessary to consider that class of contexts which are characterized by a solitary background assumption which freezes time at some particular instant. These contexts, whose only presupposition is a fixed time, are called isochronal contexts, or simply isochronons. Since the only background assumption allowed an isochronon is a time assumption, an isochronon which assumes time fixed at instant t may be written uniquely as $\underline{I}(t)$.

Closely associated with the notion of isochronon is the notion of <u>macro-state</u>. A macro-state is the sum total of conditions and situations which exist (or hold) in the context of an isochronon. The macro-state M associated with an isochronon c may be characterized by an infinite set S, the set of all relational statements which are valid in c. S is called the macro-state set of M.

Relation statements derived from formal relations (such as (GREATER, 2, 1)) are, of course, valid in all

macro-states and hence are members of all macro-state sets. Relation statements derived from chronorelations, such as (OWNS, JOHN5, CAR17), are often members of only some macro-state sets.

Any variation in lay relations produced by passing from one isochronon to another (i.e., any difference in the macro-state sets of two isochronons) is called a chronofluxion.

The special set J (Section 5.3) which relates lay relations to their associated formal relations provides a ready-made encoding of all chronofluxions. Suppose a lay relation r is defined in terms of formal relation R_1 in isochronon $\underline{I}(t_1)$, but is defined by another formal relation R_2 in isochronon $\underline{I}(t_2)$. Then

$$(\underline{I}(t_1),\ r,\ R_1) \in J,$$
$$(\underline{I}(t_2),\ r,\ R_2) \in J$$

and

$$R_1 \neq R_2.$$

Hence, the change in r produced by passing attention from $\underline{I}(t_1)$ to $\underline{I}(t_2)$ may be derived from J.

Consider a time interval T. If attention is successively focused on sequential instants in the interval T, a continuous sequence of state sets is defined. This continuous sequence of states, called the chronoflux over T, is like a showing of a motion picture of the universe taken

by an all-seeing camera. Alternatively, the sequence of states constitutes a time propelled phantasmagoria which encodes the history of the universe over the time interval T.

The whole of history (that is, the sequence of states over the interval encompassing all time) is called the chronoflux.

Chapter 7

Indirect Reference

This chapter briefly examines the concept of indirect reference and develops formalisms for its subsequent use. In Division II, indirect references will play a major role in encoding continuous sequences of change and in expressing general (quantified) statements of fact.

7.1 The value relation

A very important relationship which finds wide application in mathematics and language is the relationship which holds between two objects when one object is to be interpreted as a symbol which represents the other. If symbol (object) s represents object v, it is often said that v is the value of s. Hence, the term "value" may be adopted as the name of the relation under discussion.

It is hopefully clear that value is a lay relation since the value of a symbol s may vary with the encompassing context. For example, the string "The President" may in some settings refer to George Washington and in other settings to Abe Lincoln. If a symbol s has a value v in setting q, then write

$$(s, v) \in V_q.$$

Here V_q is intended to represent the formal relation set

defining value in setting q.

Unfortunately, it is necessary to make at least implicit use of V when discussing V, since symbolism is an integral component of communication. As an example of the problems which can arise, consider the statement

$$(X, 1) \in V_q.$$

This may easily be misinterpreted as meaning that the value of symbol "X" is to be the number one in setting q. It seems clear that the symbol "q" refers to some setting while the symbol "1" refers to the number one. Yet there is a tendency to believe that the symbol "X" is intended to stand for itself, i.e., for the letter "X". To be consistent with the interpretation of the other symbols, the symbol "X" might stand for any object (e.g., for the symbol "Y"). The object referred to by the symbol "X" is the symbol which is to stand for the number one. (A similar referent problem arises in LISP. Hence, the introduction in LISP of such functions as QUOTE and SETQ.)

To aid communication, quote marks will be used when necessary to distinguish a symbol from the value of the symbol as intended in the context of the present work. Thus "X" is the 25th letter of the alphabet, while X is the value associated with the 25th letter of the alphabet. Also, "1" is a symbol (a numeral) while 1 is the first counting number. "Washington" is the name of the first

president of the United States, while Washington was that man.

Let the present context be context q and assume that the value of the letter "X" is to be the letter "Y". Assume further that the value of letter "Y" is to be the number 1. Then the following hold

$$("X", "Y") \in V_q$$

$$("Y", 1) \in V_q$$

$$(X, 1) \in V_q.$$

This last statement is of particular interest since it is meant to indicate that the number 1 is the value of the letter "Y" (not the value of the letter "X"!).

To indicate the value of a particular symbol X in q, the functional notation

$$\underline{\text{value}}_q(X)$$

will be used. When the context is clear, this notation will be simplified to $\underline{\text{value}}(X)$. In context q described above

$$\text{value}(X) = 1.$$

7.2 Templates

Very often it will be desirable to make indirect reference to n-tuples and to relational statements (which are a particular type of n-tuple). To refer to a given n-tuple, a symbol such as "X" may be used. Unfortunately,

the symbol "X" conveys little or no information about the n-tuple it represents (except, of course, through the value function).

For purposes of discussion, it is desirable to adopt as symbols for n-tuples constructs which in themselves give some clue to the nature of the n-tuples which are their values. LISP S-expressions have this property. For example, the S-expression

(LIST (QUOTE A) B)

is a composite symbol which will evaluate in the context of LISP to

(A #B)

where #B is to be interpreted as the value of variable B. If B evaluates to the atom C, then by (A #B) is meant (A C).

To simplify the notation of LISP, let

[a b ... c]

be shorthand for

(LIST a b ... c).

Let

≡expression

be a shorthand for

(QUOTE expression)

and let

(function-name arg_1 arg_2 ... arg_n)

be replaced by

<function-name arg_1 arg_2 ... arg_n>.

Using this notation,

[≡A B]

is equivalent to the earlier example

(LIST (QUOTE A) B).

The expression

[≡KNOWS PERSON ≡(KISSED JOHN MARY)]

is equivalent to

(LIST (QUOTE KNOWS) PERSON
(QUOTE (KISSED JOHN MARY)))

and, if PERSON is bound to SAM, evaluates to

(KNOWS SAM (KISSED JOHN MARY)).

The expression

[≡KNOWS PERSON [≡KISSED ≡JOHN ≡MARY]]

also evaluates to

(KNOWS SAM (KISSED JOHN MARY)).

As a final example

[≡AT* t ≡ROBOT <times 5 t>]

is equivalent to the S-expression

(LIST (QUOTE AT*) t (QUOTE ROBOT)
(TIMES 5 t))

which, if t is bound to 7, evaluates to

(AT* 7 ROBOT 35).

This resultant S-expression is the LISP equivalent of the

n-tuple

(AT*, 7, ROBOT, 35)

and, interpreted as a relational statement, indicates that ROBOT was at point 35 at time 7.

S-expressions used as symbols to denote n-tuples will hereafter be called "templates." If the value of a template may be a relational statement, then the template may be called a template relation statement.

DIVISION II

Organizing Knowledge

Chapter 8

Goals of Division II

The methodology of mathematical logic provides tools for building an abstract, mathematical model of the world. However, in addition to the purely mathematical aspects of modeling, attention must be paid to pragmatics. The problem of organizing modeled information in a finite form convenient for question-answering and for the performance of other reasoning tasks must be considered. Further, thought must be given to the question of how knowledge may be bundled into units suitable for interaction with humans. The chapters of this division attempt to meet the problems of efficiency and human-like knowledge bundling simultaneously, reasoning that the packaging of facts into organized and meaningful units will lead to efficiencies in both storage and operations.

This division begins with a discussion of how sets of similar facts may be compressed into units called patterns which may be encoded compactly and regarded externally as an integral whole. The formalisms which are developed may be used to encode the sequences of continuous change that characterize some types of events.

Using the ability to encode continuous change, the second chapter of this division introduces the notion of

process as a human-like method for organizing temporal change. Then, in the third chapter, the categorization of objects into hierarchies is discussed as a means of coordinating and compacting various pieces of knowledge. Capitalizing upon certain features of the classification system, it is possible to represent objects at various levels of detail and to both predict and assign default values to missing pieces of information.

The final chapter in this division considers the interaction of process information and the classification system. This discussion leads to the interesting problems arising from a system's participation in the processes which it models.

Chapter 9

Reoccurring Patterns and General Statements

The formalisms discussed in Division I may be used to encode specific facts about the world such as (AT*, t_1, JOHN, JOHN'S-HOUSE). But in addition to an ability to record specific facts, a knowledge system needs some facility for grouping sets of similar facts into units, allowing these facts to be represented collectively through some compact sharing mechanism and to be conceptualized as an integrated whole.

This chapter discusses a method for gathering related facts into units called reoccurring patterns and encoding such patterns through general statements. Each general statement (or quantified expression) encodes both a set of similar facts and an indication of how the individual facts are related. This ability to encode generalized information is of considerable importance since it is often impractical (or even impossible) to record the same information by a collection of individual specific statements both because of the very number (possibly infinite) of statements required and because details of particular individuals may not be explicitly known.

9.1 Some structure for contexts

Before proceeding with the discussion of generalized statements, it is necessary to pause and examine the notion of context more closely. As used in this work, a context is a collection of background assumptions. Hence, each context q corresponds to a set A_q of restricting conditions.

In many usages, as exemplified in Section 5.2, it is convenient to isolate certain contextual attributes. By assigning values to these attributes, the restrictions of A_q may be encoded. For example, suppose $A_q = \{a_1, a_2\}$ where a_1 is the condition that the time be t = 3:00 PM, November 4, 1972, and a_2 is the condition that the place be p = Temple, Texas. The background assumptions of set A_q may be completely encoded by creating and assigning appropriate values to contextual attributes "time" and "place."

In studying generalized statements, background conditions of the form "symbol s is assigned the value v" will be of great importance. The assignment of values to symbols, just as the assignment of a fixed time, may be regarded as an attribute of context. The value of this value-assignment attribute is a set G of pairs (s, v) where the fact that (s, v) is an element of V_q is to be a background assumption in context q. Sets G of pairs (s, v) are called <u>binding sets</u>.

The notion of a binding set is closely related to

the LISP 1.5 notion of an a-list. In particular, the binding set

$$G = \{(S_1, V_1), (S_2, V_2), \ldots, (S_n, V_n)\}$$

may be encoded as the a-list

$$A = (S_1 \ V_1 \ S_2 \ V_2 \ \ldots \ S_n \ V_n).$$

The point is that both binding sets and a-lists are used to define a value relation V_q between variables and their bindings.

In what follows, contexts whose only background assumptions concern the assignments of particular values to particular symbols will be of special interest. If G is a set of symbol-value pairs, then let $\underline{C}(G)$ be the context in which

$G \subset V_{\underline{C}(G)}$,

$(*, G) \in V_{\underline{C}(G)}$,

if d is a template which evaluates to #d by using the LISP 1.5 conventions with an a-list encoding exactly G and the binding of * indicated above,

then $(d, \#d) \in V_{\underline{C}(G)}$

and

$\underline{C}(G)$ makes no background assumptions other than those needed to meet the specifications given above.

Intuitively, $\underline{C}(G)$ is the LISP environment in which

* and the left components of pairs in G are the only atoms which have assigned values. Each left component of a pair in G will evaluate to its corresponding right component and the special symbol * will evaluate to G (or, under an alternative interpretation, to the LISP a-list).

For example, if G = {(man, WOLFGANG), (woman, HELGA)}, then

$$\underline{value}_{\underline{C}(G)}([\equiv \text{LOVES man woman}]) =$$
$$(\text{LOVES WOLFGANG HELGA})$$
$$\underline{value}_{\underline{C}(G)}([\equiv \text{BINDINGS } *]) =$$
$$(\text{BINDINGS } \{(\text{man, WOLFGANG})$$
$$(\text{woman, HELGA})\}).$$

9.2 The encoding of reoccurring patterns

The primary application of indirect reference is in indicating reoccurring patterns of relationships through general statements of fact. (The term "reoccurring" is used here rather than "recurring" to avoid the implication that reoccurring patterns are necessarily recursive in nature.) Indeed, the search for such reoccurring patterns and their encoding in general statements is the central business of mathematics.

9.2.1 A familiar pattern

As an example of the use of indirect references in stating a reoccurring pattern, consider the standard quadratic equation. Letting the symbols A, B, C and X be

associated with values drawn from the set of complex numbers, if

$$0 = AX^2 + BX + C$$

then

$$X = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

In standard usage, the value of X is sought given the values of A, B and C. To record explicitly the values of X associated with each combination of A, B and C is an infinite task since there is an infinite number of such combinations. However, as has just been seen, the necessary information may be recorded implicitly by a general statement which gives the value of X for any combination of A, B and C.

9.2.2 The $\overline{\epsilon}_q$ relation and plenary patterns

To formalize the notions of reoccurring pattern and general statement presented intuitively through the quadratic example, it will be convenient to begin by defining the special element relation $\overline{\epsilon}_q$ as follows:

$$s \; \overline{\epsilon}_q \; R \text{ iff } \underline{value}_q(s) \in R \text{ (Note infix notation.)}$$

Thus, for example, if symbol s stands for (has as its value) the n-tuple (t_1, JOHN, MARY'S-HOUSE) in context q, then the statements

$$(t_1, \text{JOHN}, \text{MARY'S-HOUSE}) \in AT^*$$

and

$$s \; \overline{\epsilon}_q \; AT^*$$

are equivalent.

Consider now how $\overline{\epsilon}_q$ may help encode a set of similar facts. In general, any given fact may be encoded by a statement of the form

$$n \in R$$

where n is an n-tuple and R is a relation. A set of similar facts (facts capable of being defined in terms of the same relation set) may be encoded by a set of statements such as

$$n_1 \in R$$
$$n_2 \in R$$
$$\vdots$$
$$n_m \in R.$$

If the n-tuples n may be referenced indirectly by using a single template d in various contexts, then, using the $\overline{\epsilon}_q$ relation, the same information may be encoded as

$$d \; \overline{\epsilon}_{q_1} \; R$$
$$d \; \overline{\epsilon}_{q_2} \; R$$
$$d \; \overline{\epsilon}_{q_m} \; R$$

where each q_i is a context such that

$$\underline{value}_{q_i}(d) = n_i.$$

The template d, of course, contains indirect references

just as the equations in the quadratic example contain indirect references. Loosely, all of the related facts may be rendered in terms of the single statement

$$d \overline{\in} R$$

where this statement is to be interpreted in any of the contexts contained in $Q = \{q_1, q_2, \ldots, q_m\}$.

Formally, a plenary pattern p is a triple

$$p = (d, R, Q)$$

where d is a template, R is a relation set and Q is a set of contexts. The plenary pattern p = (d, R, Q) may be interpreted as meaning that for all contexts q in Q

$$\underline{value}_q(d) \in R.$$

A plenary pattern is plenary in the sense that the set of contexts is supplied. The "pattern" which reoccurs in these contexts is

$$d \overline{\in} R.$$

Since d and R are sufficient to specify this pattern they may be referred to collectively as "the encoded pattern."

All plenary patterns may be collected into a set PP. Hence p is a plenary pattern iff $p \in PP$.

As an elementary example of a plenary pattern, let (d, R, Q) be an element of PP where

$$d = [X \equiv SUN]$$

$$R = ORBITS$$

$$Q = \{\underline{C}(\{(X, \text{MERCURY})\}), \underline{C}(\{(X, \text{VENUS})\}),$$
$$\underline{C}(\{(X, \text{EARTH})\}), \underline{C}(\{(X, \text{MARS})\})\}.$$

Choosing the first context listed in the set Q,

$$d \ \overline{\in} \ \underline{C}(\{(X, \text{MERCURY})\})^{\text{ORBITS}}$$

or

$$\underline{\text{value}}_{\underline{C}(\{(X, \text{MERCURY})\})}(d) \in \text{ORBITS}$$

and thus

$$(\text{MERCURY}, \text{SUN}) \in \text{ORBITS}.$$

In general, for each of the inner planets there is a context $\underline{C}(\{(X, \text{planet})\})$ which is an element of Q. Hence, the plenary pattern (d, R, Q) encodes a set of facts which taken together indicate that all the inner planets orbit the sun.

Since it will often be difficult to explicitly specify a family of contexts Q, it is necessary to pursue an alternative method for encoding reoccurring patterns.

9.2.3 The sets $\forall'$, $\exists'$, $\overline{\forall}$, $\overline{\exists}$ and RP

9.2.3.1 Top level quantification

Assume that there is only one indirect reference in a template d. Let the unique indirect reference in d be in terms of the symbol x which takes on values from set X. Then define two sets $\forall'$ and $\exists'$ as follows:

$$(x, X, d) \in \forall'$$

iff

$$\forall y \in X,\ \underline{value}_{\underline{C}}(\{(x, y)\})^{(d)}.$$

Similarly,

$$(x, X, d) \in \exists'$$

iff

$$\exists y \in X \text{ such that } \underline{value}_{\underline{C}}(\{(x, y)\})^{(d)}.$$

A triple $(x, X, d) \in \forall'$ iff for all assignments of x to values in X the relation statement value(d) is valid. A triple $(x, X, d) \in \exists'$ iff there exists an assignment of x to a value in X such that the relation statement value(d) is valid.

As an example of the relations $\forall'$ and $\exists'$ consider the triple

$$Z = (x, \{\text{MERCURY, VENUS, EARTH, MARS}\}, [\equiv\text{ORBITS } x \equiv\text{SUN}]).$$

According to the definition of $\forall'$,

$$Z \in \forall'$$

iff

$$\forall p \in \{\text{MERCURY, VENUS, EARTH, MARS}\}, \underline{value}_{\underline{C}}(\{(x, p)\})^{([\equiv\text{ORBITS } x \equiv\text{SUN}])}.$$

Hence, $Z \in \forall'$ iff

$$\forall p \in \{\text{MERCURY, VENUS, EARTH, MARS}\}, (\text{ORBITS } p \text{ SUN}).$$

9.2.3.2 General quantification

Moving in the direction of generality, let two new

relations $\overline{\forall}$ and $\overline{\exists}$ be defined as follows where G is a binding set (Section 9.1).

$$(x, X, G, d) \in \overline{\forall}$$

iff

$$\forall y \in X, \underline{value}_{\underline{C}}(G \cup \{(x, y)\})(d).$$

Also

$$(x, X, G, d) \in \overline{\exists}$$

iff

$$\exists y \in X, \underline{value}_{\underline{C}}(G \cup \{(x, y)\})(d).$$

The motivation behind these definitions is to allow nesting of quantification by binding one variable at a time. Each instance of an element of $\overline{\forall}$ or $\overline{\exists}$ will serve to bind one variable. Previous bindings will be recorded through the binding sets G. The simplest usage of $\overline{\forall}$ and $\overline{\exists}$ is the case in which $G = \emptyset$. For this case, note that

$$(x, X, d) \in \forall' \text{ iff } (x, X, \emptyset, d) \in \overline{\forall}$$

and

$$(x, X, d) \in \exists' \text{ iff } (x, X, \emptyset, d) \in \overline{\exists}.$$

To see the utility of relations $\overline{\forall}$ and $\overline{\exists}$ in nested quantification, consider the rather complex template

$$d = [\equiv\overline{\forall} \quad \equiv p \quad \equiv POSTMEN \quad * \quad \equiv[\equiv BITE \quad k \quad p]]$$

and the statement

$$(k, DOGS, \emptyset, d) \in \overline{\forall}.$$

From the definition of $\overline{\forall}$ it follows that

$$\forall x \in DOGS, \underline{value}_{\underline{C}}(\emptyset \cup \{(k, x)\})(d).$$

Performing the value operation (i.e., applying LISP function EVAL to d) and paying close attention to the role of the symbol * in the definition of function C (see Section 4.1),

$$\forall x \in \text{DOGS}, (\bar{\forall}, p, \text{POSTMEN}, \{(k, x)\}, [\exists\text{BITE } k \; p]).$$

Note that value(d) is a relation statement involving $\bar{\forall}$. Expanding further,

$$\forall x \in \text{DOGS}, \forall y \in \text{POSTMEN}, \underline{\text{value}}_{\underline{C}}(\{(k, x)\} \cup \{(p, y)\})([\exists\text{BITE } k \; p]).$$

Hence,

$$\forall x \in \text{DOGS}, \forall y \in \text{POSTMEN}, (\text{BITE}, x, y).$$

In English, the statement thus signifies that every dog has bitten every postman.

The following three generalized relational statements which are similar to $(k, \text{DOGS}, \emptyset, d) \in \bar{\forall}$ further serve to show how the sets $\bar{\forall}$ and $\bar{\exists}$ may be used to encode quantification.

1) $(\bar{\forall}, k, \text{DOGS}, \emptyset, [\exists\bar{\exists} \; \exists p \; \exists\text{POSTMEN} \; * \; \exists[\exists\text{BITE } k \; p]])$
   iff $\forall x \in \text{DOGS}, \exists y \in \text{POSTMEN}, (\text{BITE}, x, y)$.
2) $(\bar{\exists}, k, \text{DOGS}, \emptyset, [\exists\bar{\forall} \; \exists p \; \exists\text{POSTMEN} \; * \; \exists[\exists\text{BITE } k \; p]])$
   iff $\exists x \in \text{DOGS}$ such that $\forall y \in \text{POSTMEN}, (\text{BITE}, x, y)$.
3) $(\bar{\exists}, k, \text{DOGS}, \emptyset, [\exists\bar{\exists} \; \exists p \; \exists\text{POSTMEN} \; * \; [\exists\text{BITE } x \; y]])$
   iff $\exists x \in \text{DOGS}$ such that $\exists y \in \exists\text{POSTMEN}$ such that (BITE, x, y).

In general, the relations $\overline{\forall}$ and $\overline{\exists}$ facilitate the encoding of an arbitrary string of quantifiers followed by an arbitrary proposition stated in terms of the quantified variables. The most global (left-most) quantification is realized by a $\overline{\forall}$ or $\overline{\exists}$ relation at the top level. More localized quantifications (quantifications encoded moving to the right) are realized by embedding $\overline{\forall}$ and $\overline{\exists}$ relations as patterns within the more global $\overline{\forall}$ and $\overline{\exists}$ relations. The innermost pattern encodes the arbitrary proposition.

It is important to realize that the relations $\overline{\forall}$ and $\overline{\exists}$ facilitate the encoding of arbitrary quantification by appealing only to the paradigm of indicating that certain objects (n-tuples) belong to certain sets ($\overline{\forall}$ and $\overline{\exists}$). When n-tuples are replaced by network nodes in Division III, the network counterpart of this feature will allow quantification to be encoded in networks.

9.2.3.3 Clusters and the commutativity of quantifiers

The reader will recall that a list of $\forall$ or $\exists$ quantifications may be commuted. For example,

$$\forall x \in X, \forall y \in Y, \forall z \in Z, [P(x, y, z)]$$

is equivalent to

$$\forall z \in Z, \forall y \in Y, \forall x \in X, [P(x, y, z)].$$

Also,

$$\exists x \in X, \exists y \in Y, \exists z \in Z, [P(x, y, z)]$$

is equivalent to

$$\exists z \in Z,\ \exists y \in Y,\ \exists x \in X,\ [P(x, y, z)].$$

Using clusters, repeating patterns (RPs) may be stated in a way which takes advantage of this commutativity. Let

$$((x_1, X_1), (x_2, X_2), \ldots, (x_n, X_n);$$
$$(y_1, Y_1), (y_2, Y_2), \ldots, (y_m, Y_m);\ G;\ d) \in RP$$

iff

$$\forall x_1' \in X_1,\ \forall x_2' \in X_2,\ \ldots,\ \forall x_n' \in X_n,$$
$$\exists y_1' \in Y_1,\ \exists y_2' \in Y_2,\ \ldots,\ \exists y_m' \in Y_m,$$
$$\underline{\text{value}}_{C(G')}(d)$$

where

$$G' = G \cup \{(x_1, x_1'), (x_2, x_2'), \ldots, (y_m, y_m')\}.$$

Of course, d itself may encode another RP relation.

Note, templates for clusters use a slash to separate components. Thus [1 2 / 3 x] evaluates to the cluster (1, 2; 3, $\underline{\text{value}}$(x)).

As an example of the use of relation set RP, consider the statement

$$\forall i \in I^+,\ \forall j \in I^+,\ \exists k \in I^+,\ k = i + j$$

where I^+ is the set of positive integers. Note that

$$k = i + j$$

may be expressed as

$$(i, j, k) \in SUM.$$

Thus, the repeating pattern may be encoded by the statement

$$((i, I^+), (j, I^+); (k, I^+); \emptyset; d) \in RP$$

where

$$d = [\equiv SUM \quad i \quad j \quad k].$$

9.3 Recording continuous change

Section 6.3 introduced the star relations and showed how they may conveniently be used to record history. Recall that the fact that John owned a yoyo at time t may be recorded as

S1: $(t, JOHN, YOYO) \in OWNS^*$.

Further, if John owns the yoyo from time t_1 until time t_2 the statement

S2: $(t_1, t_2, JOHN, YOYO) \in OWNS^{**}$

may be used to encode the information. Actually, S2 is a type of shorthand notation for the structure

$$(\overline{\mp}, t, I, \emptyset, [\equiv OWNS^* \quad t \quad \equiv JOHN \quad \equiv YOYO])$$

where I is the interval $[t_1, t_2)$.

While star-star relations are convenient for recording lay relationships which remain constant over some time interval $[t_1, t_2)$, they cannot represent relationships which undergo a continuous state of flux over an interval.

Imagine a tightrope walking robot, RBT, which lives along the one dimensional world of the tightrope. Assuming the rope is 100 feet long, the position of the robot at any

time t may be given by relation AT_t, where (RBT, x) $\in AT_t$ implies that RBT is at point x, a number between 0 and 100, at time t. Using a star relation, the fact that the robot is at point 20 at time t_1 might be recorded as

$$(t_1, \text{RBT}, 20) \in AT^*.$$

Suppose that the robot moves from point 0 to point 100, starting at time t_1 and moving at a constant speed of 10 feet/second. This fact may be recorded through the general statement

$$(\bar{\forall}, t, I, \emptyset, d)$$

where

$$I \text{ is the interval } [t_1, t_1 + 10),$$

and

$$d = [\exists AT^* \quad t \quad \exists RBT \quad <\underline{\text{times}} \quad 10 \quad <\underline{\text{difference}} \quad t \quad \exists t_1>>].$$

In a less precise, but more readable form, the pattern may be expressed as

$$(t, \text{RBT}, 10\cdot(t - t_1)) \in AT^*$$

where all literals represent constants except "t" which takes values from t_1 to $t_1 + 10$.

Extrapolating from the example, general statements may capture any sequence of change that may be characterized through a closed mathematical formula in which time is the only variable.

Chapter 10

The Notion of Process

10.1 Introduction

In practice, it is impossible to know all changes which occur from one instant to the next or to explicitly encode the chronoflux over any non-empty time interval I (Section 6.5). Despite the fact that each human has a very limited knowledge of the chronoflux, human beings perform their various tasks rather well. Thus, a complete knowledge of the chronoflux (the sequence of changing conditions through time) is not necessary. (In fact, it may be argued that the future is non-deterministic, and, hence, that "future chronoflux" is indeterminate.)

Humans tend to group certain sets or sequences of related changes into units called processes. Human knowledge relating to time-propelled change is apparently organized around these units. (Hence, the English verb system.) Presumably, every chronofluxion may ultimately be accounted for in terms of some process.

From one point of view, the total chronoflux is the result or by-product of the many processes which are in progress from one instant to the next. From an alternative vantage, a process is simply a description, characterization or explanation of certain sets of chronofluxions.

Under either interpretation, a process accounts for (realizes or interprets) a small portion of the differences occurring in sequential states. Thus, the notion of process allows the chronoflux to be broken up into small understandable pieces.

10.1.1 The automata theoretic interpretation of process

For those familiar with automata theory, the function of process in understanding (or predicting) the chronoflux may be explained through an analogy with sequential, finite state machines, such as Mealy or Moore machines. The universe (as conceived by the author and many others) is a closed system receiving no inputs and producing no outputs. Yet it does step (continuously flow) through a complex sequence of states and hence may roughly be conceptualized as a sequential machine. Many large sequential machines may be decomposed into a number of smaller machines which, working in parallel, produce the same behavior as the original machine. A process is like one of these component parallel machines. It is smaller than the total machine (and therefore more capable of being understood), yet it accounts partially for the results produced by the total machine.

10.1.2 The theatrical interpretation of process

From yet another vantage, a process is like a play

which is enacted again and again. Within the sequential progression of macro-states constituting the chronoflux, various reoccurring themes or plots are discernible. By analyzing the chronoflux in terms of these plots, history is seen as the enactment of a large number of interacting, coordinated plays. Some enactments of plays run simultaneously while others are separated by broad lapses of time. In all, these plays follow a (relatively) limited number of plots, the complexities and variety of history being produced by the interaction of plays and by the casting of different objects in the various "roles" associated with each plot.

In general, there is no particular set of plots which best characterizes the chronoflux. Given a particular plot P, it is often possible to split P into plots P_1 and P_2 where the enactment of plot P is equivalent to the simultaneous enactment of P_1 and P_2. Alternatively, P may be equivalent to the sequential enactment of P_1 and P_2. Either P or P_1 and P_2 may be included in the set of chronoflux characterizing plots. Note that it is possible (if impractical) to combine all plots into a single macro-plot, the plot which characterized the total chronoflux.

10.2 Background for the representation of processes

If chronological change is to be understood in

terms of processes, then a representation of both process plots and the instantiation of such plots must be formulated. Following the analogy with plays, it is necessary to have representations for both play scripts and the notices that such and such a play is performed at such and such place with so and sos playing the various parts. To provide these representations, it will be convenient to build upon the established framework of robot modeling systems such as STRIPS, the Siklóssy-Dreussi-Roach robot and the Winograd system.

10.2.1 State of the world models

The techniques of robot modeling attempt to provide a complete description (to some coarse level of detail) of a very limited micro-world. Knowledge about such a micro-world is divided into two categories: state knowledge and process knowledge. State knowledge relates to knowledge about the micro-world at certain instants in time. Knowledge relating to a particular instant is represented by a state of the world model (SWM). An SWM is simply a highly abbreviated version of the macro-state sets discussed in Section 6.5. That is, a micro-world's SWM is a set of relation statements which are highly pertinent to the description of the micro-world. (This set is a subset (or abstraction) of some macro-state set.)

An example micro-world is depicted in Figure 10.01. Also, included in Figure 10.02 is a set of relation statements which constitute an SWM for the micro-world.

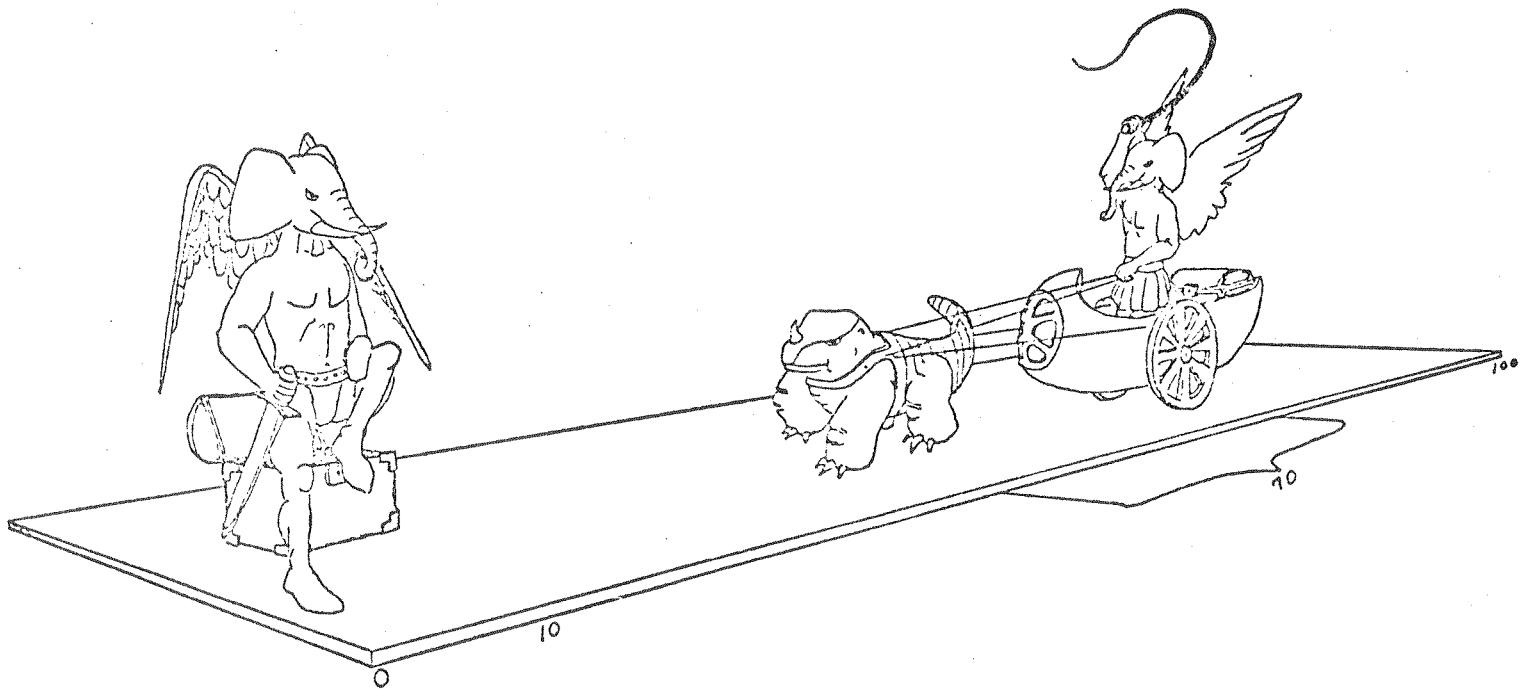
Although this world might be described at any number of levels of detail, for present purposes it is to be described very simply. The one-dimensional world contains five major objects: two winged creatures, two treasure chests and a chariot drawn by a rhinoceros. Despite intuition to the contrary, each of these objects is to be assumed to occupy a single point along the world's single dimension. So that creatures, chariot and chests may move freely, it will also be assumed that any number of these objects may occupy the same point at the same time without necessarily affecting one another.

Going over the SWM line by line (Figure 10.02) using the relation key of Figure 10.03, the first statement indicates that CR1 is an ELEMent of the set CREATURES. Given that CREATURES is the set of all creatures (in the micro-world), CR1 must be a creature. Likewise, CR2 is a creature, TC1 is a (treasure) chest and CHR is a chariot.

It is assumed that the one-dimensional creatures' world is a line segment which contains points 0 through 100. The statement

(AT, CR1, 10)

indicates that creature CR1 is AT location 10, that is, at



A Pictorial Representation of the Creatures' World
(Drawn by Mike Smith)

Figure 10.01

(ELEM, CR1, CREATURES)	(AT, CR1, 10)
(ELEM, CR2, CREATURES)	(AT, CR2, 70)
(ELEM, TC1, TCHESTS)	(AT, TC1, 10)
(ELEM, TC2, TCHESTS)	(AT, TC2, 70)
(ELEM, CHR, CHARIOTS)	(AT, CHR, 70)
(ON, CR1, GROUND)	(MAXGROUNDSPEED, CR1, 15)
(ON, CR2, CHR)	(MAXGROUNDSPEED, CR2, 16)
(ON, TC1, GROUND)	(MAXGROUNDSPEED, CHR, 20)
(ON, TC2, CHR)	(MAXAIRSPEED, CR1, 20)
(ON, CHR, GROUND)	(MAXAIRSPEED, CR2, 25)

SWM of a State of the Creatures' World

Figure 10.02

NOTATION	EXPLANATION
[≡AT o p]	Object o is at place p.
[≡ELEM i s]	Item i belongs to set s.
[≡LESSEQ n1 n2]	Number n1 is not greater than number n2.
[≡LESSTHAN n1 n2]	Number n1 is less than number n2.
[≡MAXAIRSPEED c s]	The top speed at which creature c can fly is s.
[≡MAXGROUNDSPEED c s]	The top speed at which creature c can walk is s.
[≡ON x y]	Object x is on (supported by) object y.
[≡SELECTED c activity ...]	Freewill agent c has selected to participate in the activity [activity ...].
[≡¬relation ...]	Shorthand for ¬[≡relation ...]. E.g., [≡¬AT o p] means object o is not at place p.
[≡CALLED c f]	Creature c has chosen to bet that a coin will land with face f up.
[≡FALLENOFF x]	Object x has fallen off the edge of the creatures' world.
[≡INUSE p]	Phone p is off the hook.
[≡NUMBER p d1 d2]	The two digit phone number for phone p is d1 d2.
[≡UNCOMMITTED c]	Creature c has not yet chosen a face upon which to bet in a coin toss.

Key to Relationships in the Creatures' World

Figure 10.03

point 10 on the number line. Locations of other objects in the micro-world are given by similar statements.

Despite the fact that the world is assumed to be one-dimensional, an object may be ON (on top of) another. CR1, TC1 and CHR are on the ground. Hence, for example,

(ON, CR1, GROUND)

is a statement in the SWM. (Figure 10.01 catches CR1 with one foot on the ground and one foot in the air as he steps down from atop TC1.)

It is assumed that the creatures can both walk and fly. The MAXSPEED relations indicate the creatures' maximum walking and flying speeds. For example,

(MAXGROUNDSPEED, CR1, 15)

indicates that CR1 has a top walking speed of 15 length units per time unit.

Many other aspects of the creatures' world might be added to the list of Figure 10.02. (E.g., statements could be added to indicate that the creatures have wings and trunks, that the chariot has a driver's place in the front and a cargo section in the back, etc.) However, the description as given is complete to the level of detail needed to understand the most fundamental processes which transpire in the creatures' world.

10.2.2 Robotic descriptions of process knowledge

An SWM such as that just presented depicts a

micro-world at a single instant in time. To make the micro-world dynamic, conventional world modeling systems have introduced "operators" which describe how one state may be transformed into another. In what follows, the fluid of kinetics will be injected into the micro-world through "scenarios," a generalization of the "operator" concept (see Hendrix, 1973b).

10.2.2.1 Instantaneous process scenarios

If a creature and a (the) chariot are at the same point at the same time and the creature is on the ground, then the creature may board (climb ON to) the chariot. Assume that the creatures are very skilled at boarding chariots so that the boarding operation effectively requires no time. The boarding process may be encoded at one level of detail by the following instantaneous process scenario.

Scenario name: "BOARD"

Parameters: {c, w, p}

Initiation conditions:

{[≡ELEM c ≡CREATURES],
[≡ELEM w ≡CHARIOTS],
[≡AT c p],
[≡AT w p],
[≡ON c ≡GROUND],
[≡SELECTED c ≡BOARD c w]}

Effects - instantaneous:

delete: {[≡ON c ≡GROUND]}

add: {[≡ON c w]}

This process scenario may be interpreted as follows: The first line of the scenario indicates that the scenario's name is "BOARD". This name is analogous to the title of a play. The next part of the scenario is the parameter set which is analogous to a list of characters. The elements of this set are the symbols which may be used in making indirect reference to the actors which play the various roles. The initiation conditions may be thought of as being analogous to either the play's setting or starting cue. That is, the play will take place (the process will be realized or instantiated) whenever the various initiation conditions are met. The conditions are given as a set of template relation statements containing variables taken from the parameter list. Interpreting these one by one, there must be a creature c and a chariot w. ("w" was chosen for "wagon" to avoid confusion with "c" for "chariot" and "creature".) Creature c and chariot w must be at the same point p. Creature c must be on the ground and, lastly, creature c must have selected to board the chariot. (The SELECTED relation is explained more fully in Hendrix (1973b). In short, the creature has a free will and will activate its muscles to the performance of the task only if

it wants to. In general, the notation

[≡SELECTED agt activity ...]

indicates that a freewill agent, agt, has decided to engage in an activity indicated by

[activity ...].

In this example, a creature c must decide to engage in the activity [≡BOARD c w], the boarding of chariot w by c. For interpretations of other relationships, see Figure 10.03.)

In determining that the initiation conditions hold, the various parameters are "bound" to certain objects in the micro-world. That is, a context g is defined so that if

$$(x, y) \in V_g,$$

then role x (a parameter) is play by cast member y (an object in the micro-world). It is important to realize that V_g is a binding set (Section 9.1). For any particular instantiation of the scenario, V_g is a concise specification of who played what part.

In addition to the parameters in the parameter set, a special parameter, $t_@$, is bound when the initiation conditions are met. (Parameter $t_@$ may be thought of as an implicit member of the parameter set.) The binding of $t_@$

is the time at which the process begins (i.e., the time at which the initiation conditions first hold).

If the initiation conditions are met, then the action of the play begins. The action is quite simple. The template relation statement

S1: [≡ON c ≡GROUND]

ceases to be true while the relation statement

S2: [≡ON c w]

becomes true. That is, in the state in which the process begins, $\underline{\text{value}}_{\underline{C}}(V_g)$ of statement S1 holds and $\underline{\text{value}}_{\underline{C}}(V_g)$ of statement S2 does not hold. The process is performed in only an instant, but in the state succeeding the play's beginning by this brief instant, statement S2 holds while statement S1 does not. The changes in the truth values of S1 and S2 are concrete examples of the chronofluxions defined in Section 6.5.

Looking at the changes in truth value of S1 and S2 in terms of star-star relations, it is known that

$$[\equiv\text{TBEFORE}\ \ t_{@}\ \ c\ \ \equiv\text{GROUND}]\ \overline{\epsilon}_{\underline{C}(V_g)}\ \text{ON}^{**}$$

and

$$[t_{@}\ \ \equiv\text{TAFTER}\ \ c\ \ w]\ \overline{\epsilon}_{\underline{C}(V_g)}\ \text{ON}^{**}$$

where

$t_{@}$ is the special symbol representing the time at which the BOARD process occurred,

TBEFORE is a time before the process began,

and

TAFTER is a time after the process took place.

TBEFORE and TAFTER are unique for any given BOARDing process. The condition of c being on the ground that was terminated by the BOARD process was initiated at time TBEFORE. The condition of c being on w that was initiated by the BOARD process is (was, will be) terminated at time TAFTER.

10.2.2.1.1 Scenarios as relations

The process scenario just presented may be encoded as an n-tuple in a relation set. Let IPS encode the set of all Instantaneous Process Scenarios. An element s of IPS is a 4-tuple of the form

(p, i, d, a)

where

p is a set of parameters,

i is a set of template relation statements constituting the process' initiation conditions,

d is a set of template relation statements to be deleted

and

a is a set of template relation statements to be added.

To indicate that a particular 4-tuple encodes an instantaneous process scenario, the statement

$$(p, i, d, a) \in IPS$$

may be made.

Note that the scenario name is <u>not</u> included in a 4-tuple (p, i, d, a). In the context of this work, a name such as "BOARD" has as its <u>value</u> one of these 4-tuples. Hence, BOARD is the 4-tuple describing the scenario whose name is "BOARD".

```
BOARD = ({c, w, p},
         {[≡ELEM  c  ≡CREATURES],
          .
          .
          .
          [≡SELECTED  c  ≡BOARD  c  w]},
         {[≡ON  c  ≡GROUND]},
         {[≡ON  c  w]})
```

By itself, a process scenario does not describe

changes in the world. A scenario merely describes a type or characterization of change. It is the instantiation of a scenario which constitutes an actual change. For example, if the BOARD scenario is realized at time t_0 with parameter c bound to CR1, w bound to CHR and p bound to 66, then CR1 actually boards CHR at time t_0. Clearly, the members of the cast (that is, the values of c, w and p) are in a definite relationship with each other, with the process scenario and with the time at which the process scenario was instantiated. This relationship might be encoded in any number of ways.

One possible encoding scheme is to define a relation set IIPS, the Instantiated Instantaneous Process Scenario relation, as follows. Let

$$(s, t, G)$$

be an element of IIPS iff the scenario s, an element of IPS, was realized (instantiated) at time t with the various explicit scenario parameters bound in accordance with binding set G.

According to this scheme, the instantiation of BOARD just cited as an example may be encoded as

$$(\text{BOARD}, t_0, \{(c, \text{CR1}), (w, \text{CHR}), (p, 66)\}) \in \text{IIPS}.$$

Pragmatically, it may well be more convenient to define a separate relation for each scenario. Thus, for

scenario BOARD, a relation R^{BOARD} will be defined to encode instantiations of BOARD. R^{BOARD} effectively encodes the set of all boarding events for the micro-world.

Before defining R^{BOARD}, assume that $t_@$ is a special symbol which is an implicit member of the parameter sets of all instantaneous scenarios. This special symbol will take as its value the time of an event instantiation. Then let R^{BOARD} be a set of binding sets where

$$G \in R^{BOARD}$$

iff scenario BOARD was realized at a time t with parameters bound in accordance with G. Further, $(t_@, t)$ must be an element of G.

In terms of the instantiation example,

$$\{(c, CR1), (w, CHR), (p, 66), (t_@, t_0)\} \in R^{BOARD}.$$

In general,

$$G \in R^S$$

iff scenario S was realized at time t with scenario parameters bound in accordance with G, and $(t_@, t) \in G$. (Those familiar with case systems (see Bruce, 1973) may think of an element of R^S as a set of case-value pairs.)

10.2.2.2 Scenarios of duration

Instantaneous scenarios such as BOARD describe processes which transpire in a single instant. For those

processes which transpire over a period of time (i.e., over a time interval) another type of scenario is needed.

As an example of a process which transpires over an interval of time, consider walking. In the creatures' world, there is no need (at the moment) to be concerned with such details as the moving of feet, the bending of knees, etc. Walking in the creatures' world may be equated simply with movement along the ground. For purposes of discussion it will be convenient to view walking toward the right (moving to points along the micro-world's one-dimension which are associated with increasing numbers) as a separate process from walking toward the left (moving to points of lesser number). At one level of detail the WALKRIGHT process may be given by the following duration process scenario.

Scenario name: "WALKRIGHT"

Parameters: {c, cspeed, cfrom, cmaxspeed}

Initiation conditions:

{[≡ELEM c ≡CREATURES],
[≡ON c ≡GROUND],
[≡AT c cfrom],
[≡LESSTHAN cfrom 100],
[≡SELECTED c ≡WALKRIGHT c cspeed],
[≡LESSTHAN 0 cspeed],
[≡MAXGROUNDSPEED c cmaxspeed],

[≡LESSEQ cspeed cmaxspeed]}

Effects - continuing:

{[≡AT* t c <add cfrom <times cspeed
<difference t t_I>>>]}

Terminating conditions:

{[≡¬ON c ≡GROUND],
[≡¬SELECTED c ≡WALKRIGHT c cspeed],
[≡AT c 100]}

This scenario is presented in a form which closely parallels that used by the scenarios of instantaneous processes. Again there is a scenario name, a parameter list and a set of initiation conditions expressed as template relation statements. (See Figure 10.03 for relation key.)

The initiation conditions indicate that there must be a creature c which is on the GROUND and AT a point cfrom. Since the creature c is to walk toward the right, cfrom must be less than 100. Otherwise c would be at the extreme right edge of the micro-world and hence could not move further to the right.

Before creature c actually walks right, it must choose to do so. This allocation of the creature's resources is indicated through the SELECTED relation. In selecting to walk right, the creature also selects a walking speed cspeed. This speed must be greater than zero for any movement to take place toward the right. Further,

cspeed must be less than or equal to cmaxspeed where cmaxspeed is the maximum speed at which c may walk.

In determining that the initiation conditions hold, the various parameters are bound to objects in the microworld just as with instantaneous scenarios. Again, a set V_g describing these bindings is defined.

If <u>all</u> initiation conditions are met, then the effect of the process begins. For the WALKRIGHT process, the effect is to move the creature gradually toward the right. The gradual change in the creature's location (which will be described shortly) will continue until any one of several terminating conditions arises. If the creature is not on the GROUND, then it can no longer walk. If the creature stops selecting to walk, then the creature will no longer walk. If the creature is at point 100, then it can walk no farther to the right. These alternative conditions for process termination are presented in the scenario under the heading "Terminating conditions."

The primary distinction of duration processes is that between the time a duration process is initiated and the time it terminates, a <u>continuous</u> sequence of gradual changes may be operative. For each instantiation of a duration process, let t_I be the time at which the initiation conditions are met and let t_F be the time at which some terminating condition arises. Finally, let I be the

half-open interval $[t_I, t_F)$. The process may be thought to be operative over I.

The gradual effects of a duration process are specified under the heading "Effects - continuing". Any template d listed under this heading is to be interpreted as the template of a repeating pattern

$$(\overline{\forall}, t, I, V_g, d)$$

where t is a variable over I, I (as just defined) is $[t_I, t_F)$ and V_g is the binding set as defined above. Thus, the continuing effects of the WALKRIGHT process are given by

$$(\overline{\forall}, t, I, V_g, [\exists AT^* \ t \ c \ <\underline{add} \ cfrom \ <\underline{times} \ cspeed \ <\underline{difference} \ t \ t_I>>>]).$$

Interpreting this specification in accordance with the definition presented in the chapter on reoccurring patterns (Chapter 9),

$$\forall t \in I,$$

$$(t, \#c, \#cfrom + \#cspeed \cdot (t - \#t_I)) \in AT^*$$

where #x means $\underline{value}_g(x)$. Less precisely, but more legibly, at any time t while the process is in effect, creature c will be AT a point y where

$$y = \#cfrom + \#cspeed \cdot (t - \#t_I).$$

In general, the templates specified under continuing effects may be arbitrarily complex. However, it must be pointed out that in language the exact specification of

a function (such as the position function in WALKRIGHT) is seldom if ever known precisely. For sentences such as

"John walked from his house to town,"

it is only necessary to know that some (unspecified) function exists which gives John's location during the walking process. While the exact nature of the function remains cloudy, certain statements regarding the function can be made. For example, John's location at the beginning and end of the process time interval are known. Further, the range of the first derivative of the function is known. (John cannot change position faster than his walking speed allows.) From this information alone, it is possible to answer such questions as

"Could John have visited point X on his walk to town?"

and

"Could John have been at point X at time t?"

10.2.2.2.1 Duration process relations

Scenarios of duration, just as instantaneous scenarios, may be encoded through sets of n-tuples. Let DPS encode the set of all Duration Process Scenarios. An element of DPS is a 4-tuple of the form

(p, i, f, D)

where

p is a parameter set

i is a set of template relation statements constituting the process' initiation conditions

f is a set of template relation statements constituting the process' termination conditions

and

D is a set of templates d where

$(\bar{\forall}, t, I, V_g, d)$.

Thus,

WALKRIGHT =

({c, cspeed, cfrom, cmaxspeed},

{[≡ELEM c ≡CREATURES],

.

.

.

[≡LESSEQ cspeed cmaxspeed]},

{[≡¬ON c ≡GROUND],

[≡¬SELECTED c ≡WALKRIGHT c cspeed],

[≡AT c 100]},

{[≡AT* t c <add cfrom <times cspeed <difference t t_I>>>]}).

Instantiations of duration processes may be encoded by constructing a set IDPS of 4-tuples (s, t_I, t_F, G) where t_I and t_F are the process initiation and termination times

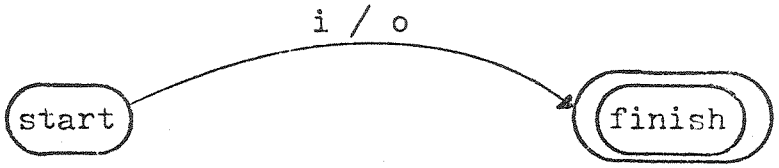
respectively and s and G are as defined in IIPS.

10.3 Process automata

The last two sections have dealt with the two simplest types of processes, instantaneous processes and processes of duration where one set of repeating patterns holds over the entire duration of the process. Many processes are, of course, complex combinations of these simplest types.

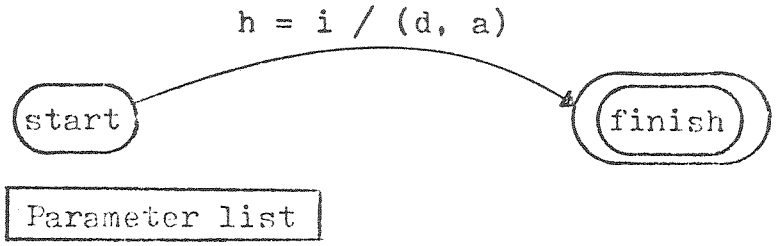
To study complex processes, it will be useful to have some systematic method of stating how various complex processes are built up from the simple types just presented. Toward this end, consider the Mealy machine (1955). One of the simplest Mealy machines is depicted in Figure 10.04. Beginning at control point +start+ (the node labeled "start") the machine may accept an input i. When input i arrives, output o is produced and the machine undergoes a transition to control point +finish+.

(Note: In discussing Mealy machines and process automata, it will be helpful to adopt a few conventions for referring to network diagrams. First, no two nodes in the same diagram will have the same label. The node (control point) labeled "<string>" will be referred to in the text as +<string>+. The information written on an arc (the "inputs" and "outputs") may not be unique. To give a



A Simple Mealy Machine

Figure 10.04



An Instantaneous Process Automaton

Figure 10.05

particular arc a name for use in the text, the sequence "<symbol> =" may appear at the beginning of an arc label. For example, the arc in Figure 10.05 is labeled with information beginning with "h =". Thus, this arc may be referred to as arc h. Actual properties of the arc are indicated by the information to the right of the equal sign.)

10.3.1 Process automata for instantaneous processes

Building on this basic idea, consider the process automaton of Figure 10.05. This machine looks very much like the Mealy machine of Figure 10.04, but has (d, a) in place of output o and contains a parameter box. This process automaton may be interpreted as follows. Beginning at control point +start+, the process automaton (pa) waits for initiation conditions i to be met. Conditions i are stated in terms of templates involving parameters in the parameter set. In attempting to determine whether conditions i are met, a binding set V_g is determined, just as in the processing of scenarios.

Whenever conditions i are met (and thus the parameters bound in accordance with a binding set V_g) the pa undergoes a control point transition from +start+ to +finish+. During the transition, (d, a) is produced as an "output." This output is the deletion of conditions in set d and the addition of conditions in set a. (Both d and a

are sets of template statements.) The result of this output is a change in the state of the world. That is, if i occurs in chronon c_1 and c_2 is the chronon immediately following c_1, then conditions a will exist in the macrostate set of c_2 while conditions d will not exist in that set (unless also included in a).

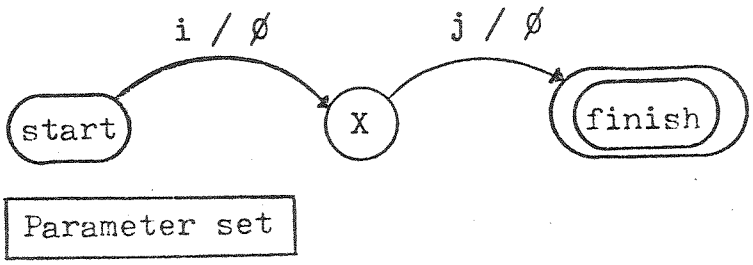
After reaching control point +finish+, the process is completed.

Thus, the pa of Figure 10.05 encodes the scenario (p, i, d, a), an element of IPS, where p is encoded by the parameter set of the pa.

10.3.2 Process automata for simple duration processes

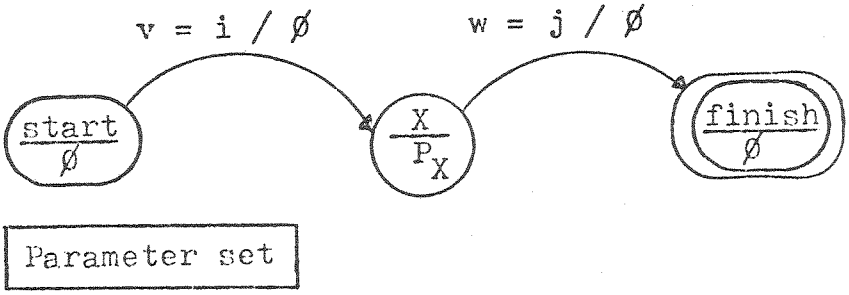
Consider the process automaton depicted in Figure 10.06. Beginning at the control point labeled "start," this pa waits for conditions i. When i occur, the pa steps to control point +X+, producing no (the null) output in the transition. The pa remains at control point +X+ until conditions j occur, moving the machine to +finish+. Again, no output is produced.

This pa would appear to be rather useless since no outputs are produced during control point transitions. However, control points of a pa are intended to be unlike the control points of conventional automata or Mealy or Moore machines. Rather, a control point C is to be



A Three Control Point Process Automaton

Figure 10.06



A Simple Duration Process Automaton

Figure 10.07

interpreted as a state of continuous flux.

The dynamics of conventional finite state (control point) machines are associated solely with the arcs of the machines. In process automata, much of the burden of describing change is carried by the control points themselves. Let I be the interval between the time a control point CP is entered and the time it is exited. The control point CP may describe any number of reoccurring patterns of events which occur over time interval I. These repeating patterns may be encoded by associating certain templates with each control point.

Let D_{CP} be the set of templates encoding the patterns associated with control point CP. Now D_{CP} will be used in the context of control point CP. Let I, as described above, be the interval over which control resides at CP. Further, let V_g be the binding set associated with the pa in which CP is a control point. (V_g is established when certain initiation conditions are met.) Then

$$d \in D_{CP} \text{ implies that } (\overline{\mp}, t, I, V_g, d)$$

where t is a symbol not currently assigned a value.

In the example of Figure 10.06, the control points +start+ and +finish+ are associated with empty sets of pattern specifications while +X+ is associated with a set of pattern specifications P_X. To indicate the associated patterns pictorially, each control point node may be cut in

half with the label in the top half and the associated pattern set in the lower half. Thus, Figure 10.06 may be redrawn as Figure 10.07.

The process automaton depicted in Figure 10.07 may be used to express the various elements of the set DPS, the set of Duration Process Scenarios. The association between Figure 10.07 and any (p, k, f, D) ϵ CPS is as follows:

the parameter set of the pa is p

i = k

j = the singleton set whose sole element is a template encoding the disjunct of the conditions specified by the elements of f.

P_X = D.

The rather awkward correspondence between j and f is due to the fact that the set of scenario termination conditions f is realized in the process automaton as a set of initiation conditions j for crossing an arc. The conditions in an initiation set act as conjuncts while conditions in a termination set act as disjuncts.

As a concrete example of a simple duration process automaton, consider how the WALKRIGHT process of Section 10.2.2.2 might be encoded. Using the automaton structure of Figure 10.07, let

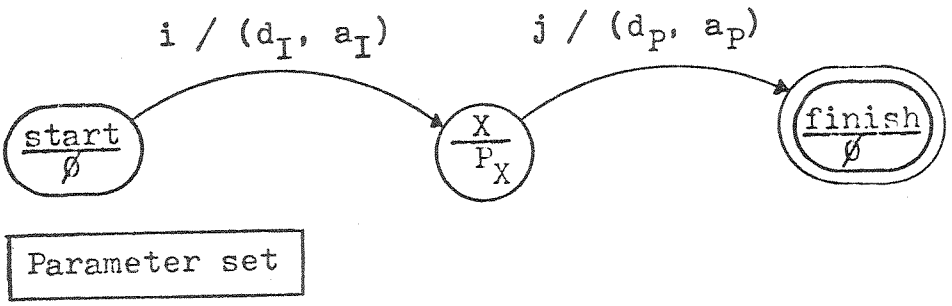
$$i = \{[\exists ELEM \quad c \quad \exists CREATURES],$$
$$[\exists ON \quad c \quad \exists GROUND],$$
$$\vdots$$
$$[\exists LESSEQ \quad cspeed \quad cmaxspeed]\}$$
$$j = \{[\exists OR\ [\exists \neg ON \quad c \quad \exists GROUND]$$
$$[\exists \neg SELECTED \quad c$$
$$\exists WALKRIGHT \quad c \quad cspeed]$$
$$[\exists AT \quad c \quad 100]]\}$$
$$P_X = \{[\exists AT^* \quad t \quad c \ <\underline{add} \quad cfrom \ <\underline{times} \quad cspeed$$
$$<\underline{difference} \quad t \quad t_I>>>]\}$$

where the parameter set is

{c, cspeed, cfrom, cmaxspeed}.

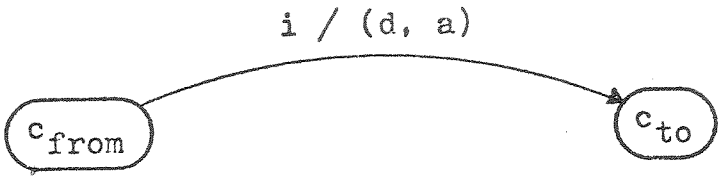
10.3.3 A note on processes with effects sandwiches

In Hendrix (1973b) the author has discussed scenarios with "effects sandwiches." This type of scenario is an amalgamation of two instantaneous scenarios and a scenario of duration which sandwiches a sequence of continuous effects between two sets of discrete effects. The two sets of instantaneous changes accompany the initiation and termination of the continuous effects. This type of scenario is expressed by a pa of the form depicted in Figure 10.08 where d_I and a_I are from the Effects - I (the initial discrete effects) and d_P and a_P are from the Effects - P (the discrete post effects which follow termination of the continuous process). A type of sandwiching also occurs in the example of the next section.



A Process Automaton for "Effects Sandwiches"

Figure 10.08



The Arc $k = (c_{from}, c_{to}, i, d, a)$

Figure 10.09

10.3.4 More complex process automata

In general, more complex process automata may be created by incorporating more control points and more arcs into the design. Let PA be the set of all process automata. PA may be characterized as a set of 5-tuples

$$pa = (C, K, c_S, C_F, P) \in PA$$

where

C is a set of control points,

K is a set of arcs connecting control points,

$c_S \in C$ is the starting control point,

$C_F \subset C$ is a set of final control points

and

P is a set of symbols (parameters).

Each $c \in C$ is a pair of the form

$$c = (N, D_c)$$

where N is the control point "name" and D_c is a set of pattern specifications. No two control points in C may have the same "name" although any number may have the same set of pattern specifications. (The sole purpose of the "name" is to distinguish control points with the same pattern sets.)

Each $k \in K$ is a 5-tuple

$$k = (c_{from}, c_{to}, i, d, a)$$

which represents an arc from c_{from} to c_{to} with "input" i and "output" (d, a). See Figure 10.09.

The process automaton begins with control residing in c_s. The process is said to have come to completion (or to have been interrupted) when control resides in an element of C_F. The elements of $C_F \cup \{c_s\}$ are of the form $(N, \emptyset)$. That is, repeating patterns must not be defined over an interval in which a pa is in either the starting or one of the final states.

If $k = (c_{from}, c_{to}, i, d, a)$ is an element of K, then i, d and a are sets of template relation statements. All variables in these template statements must come from the parameter set P. It is assumed that a pa is activated (at control point c_s) with the elements of P unbound, that is, with $V_g = \emptyset$.

Suppose control of a pa is at a control point c which is the from node of n arcs k_1 to k_n. For $j \in \{1, 2, \ldots, n\}$ let

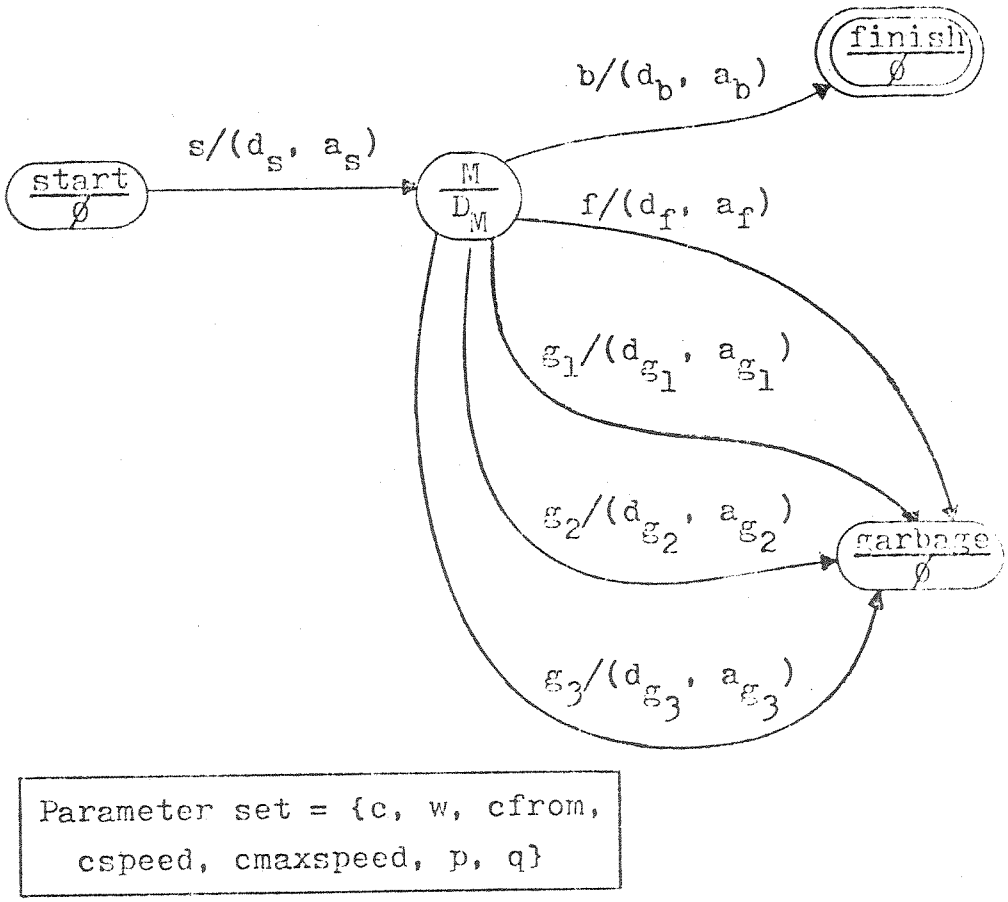
$$k_j = (c, c_j, i_j, d_j, a_j).$$

At each instant the pa "looks" at each set of template conditions i_j. These i_j should be mutually exclusive. Some of the symbols used by template relations in i_j may already be bound. Such bindings may not be altered. For those which are not bound, the pa attempts to find bindings which make all the template statements in i_j hold simultaneously. If this is ever achieved, then the symbols become permanently bound to these new bindings (i.e., the bindings are

recorded in V_g) and control passes to c_j. In the transition from c to c_j, the d_j are deleted and a_j are added. Any symbols used in d_j or a_j must be bound before the transition. According to this procedure, control passes from control point c through that arc k_j whose associated i_j are first to hold. Since the i_j are assumed to be mutually exclusive, no conflicts may arise due to initiation conditions on two arcs holding simultaneously.

As an example of a process automaton with multiple nodes and arcs, consider the automaton for catching a chariot (a process similar to catching a bus) shown in Figure 10.10. In the creatures' world, the process of catching a chariot involves a creature c walking at a constant (but perhaps swift) pace toward the right until its location coincides with that of a chariot w. Chariot w may be moving in either direction or be stationary. The process begins as soon as the creature begins walking to the right. Successful completion is achieved if (and only if) the creature's constant motion causes it to intercept the chariot and the creature boards the chariot at the moment of interception. If the creature ever changes his walking status, or takes to the air, or does not immediately board the chariot when they meet, or walks off the end of the world before intercepting the chariot, the process definition is not satisfied.

PA Name: CATCH-CHARIOT



A Process Automaton for CATCHing CHARIOTs

Figure 10.10

At the beginning of process execution, the control of pa CATCH-CHARIOT is resident at control point +start+. When initiation conditions s (for "start") are met, control moves to +M+ (for "motion"). While at +M+, the templates of set D_M indicate the position of the creature with respect to time.

There are five arcs leaving +M+, representing five mutually exclusive conditions for bringing the movement specified by the templates of set D_M to a halt. The first of these conditions is specified by the set of template statements b. The conditions of b (which are defined more precisely below -- "b" stands for "board") will arise if, at the very moment when the creature intercepts the chariot, the creature changes its activity selection (the relationship SELECTED) from walking to boarding. If conditions b arise while at +M+, the pa undergoes a transition to control point +finish+, a terminal state. In making this transition, the changes specified by d_b and a_b are make in the state of the world, causing c to become "on board" the chariot.

The conditions specified by f (for "fall") will arise if the creature reaches point 100 while still walking along the ground. The creature may or may not have passed the chariot during this walk. When conditions f are met, a transition is made to +garbage+, a nonterminal control

point with no exits. The arrival of control at this non-terminal/nonexitable point indicates the failure of the process. In making the transition triggered by f, the pa alters the state of the world in accordance with d_f and a_f, indicating that the creature has fallen into the great abyss.

The three sets of conditions specified by g_1, g_2 and g_3 indicate other circumstances which might cause the movement of +M+ to be terminated. Each of these alternatives causes the process to pass to +garbage+ and hence to fail.

In terms of the set PA of process automata defined above,

$$\text{CATCH-CHARIOT} = (C, K, (+\text{start}+, \emptyset), \{(+\text{finish}+, \emptyset)\}, P) \in PA$$

where

$$C = \{(+\text{start}+, \emptyset), (+M+, D_M), (+\text{finish}+, \emptyset), (+\text{garbage}+, \emptyset)\}$$

$$D_M = \{[\exists AT^* \; t \; c \; \langle \underline{\text{add}} \; c \text{from} \langle \underline{\text{times}} \; c\text{speed} \langle \underline{\text{difference}} \; t \; t_I \rangle \rangle \rangle]\}$$

$$K = \{((+\text{start}+, \emptyset), (+M+, D_M), s, d_s, a_s), ((+M+, D_M), (+\text{finish}+, \emptyset), b, d_b, a_b), ((+M+, D_M), (+\text{garbage}+, \emptyset), f, d_f, a_f), ((+M+, D_M), (+\text{garbage}+, \emptyset), g_1, d_{g_1}, a_{g_1}),$$

$((+M+, D_M), (+garbage+, \emptyset), g_2, d_{g_2}, a_{g_2}),$

$((+M+, D_M), (+garbage+, \emptyset), g_3, d_{g_3}, a_{g_3})\}$

and

$P = \{c, w, cfrom, cspeed, cmaxspeed, p, q\}.$

The detailed definitions of the various sets of templates used in defining arcs are as follows:

$s = \{[\equiv ELEM \ c \ \equiv CREATURES],$
$[\equiv ELEM \ w \ \equiv CHARIOTS],$
$[\equiv ON \ c \ \equiv GROUND],$
$[\equiv AT \ c \ cfrom],$
$[\equiv LESSTHAN \ cfrom \ 100],$
$[\equiv SELECTED \ c \ \equiv WALKRIGHT \ c \ cspeed],$
$[\equiv LESSTHAN \ 0 \ cspeed],$
$[\equiv MAXGROUNDSPEED \ c \ cmaxspeed],$
$[\equiv LESSEQ \ cspeed \ cmaxspeed]\}$

$d_s = a_s = \emptyset$

$b = \{[\equiv AT \ c \ p],$
$[\equiv AT \ w \ p],$
$[\equiv SELECTED \ c \ \equiv BOARD \ c \ w],$
$[\equiv LESSTHAN \ p \ 100],$
$[\equiv ON \ c \ \equiv GROUND]\}$

$d_b = \{[\equiv ON \ c \ \equiv GROUND]\}$

$a_b = \{[\equiv ON \ c \ w]\}$

$f = \{[\equiv AT \ c \ 100], [\equiv ON \ c \ \equiv GROUND]\}$

d_f = {[≡ON c ≡GROUND]}

a_f = {[≡FALLENOFF c]}

g_1 = {[≡¬ON c ≡GROUND]}

g_2 = {[≡¬SELECTED c ≡WALKRIGHT c cspeed],
[≡AT c p],
[≡LESSTHAN p 100],
[≡AT w q],
[≡¬EQUAL p q]}

g_3 = {[≡¬SELECTED c ≡WALKRIGHT c cspeed],
[≡AT c p],
[≡AT w p],
[≡LESSTHAN p 100],
[≡¬SELECTED c ≡BOARD c w]}

$d_{g_1} = a_{g_1} = d_{g_2} = a_{g_2} = d_{g_3} = a_{g_3} = \emptyset$

In the definitions above, it is assumed that a creature may SELECT to participate in at most one activity at any one time. Conditions s, b and f were discussed earlier. The conditions g_i have the following interpretations: g_1 arises whenever something causes creature c to leave the ground; g_2 arises if the creature chooses to stop selecting to walk at a point that is not occupied by the chariot and is not the end of the world; g_3 arises if the creature chooses to stop walking at a point where it

intercepts the chariot, but does not immediately choose to board the chariot.

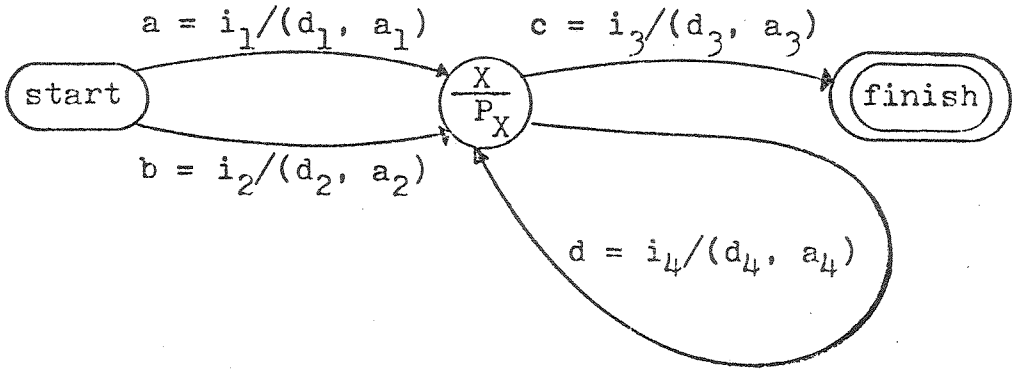
Rather than build up the definition of pa CATCH-CHARIOT from primitive relationships, it might well have been easier to make use of the secondary concepts WALKRIGHT and BOARD. Such a use of secondary concepts is discussed in Section 10.3.6.

10.3.5 Process automata traces

As stated earlier, an instantiation of a process whose plot W is encoded by one of the scenarios of Section 10.2.2 may be specified by citing the plot W and a binding set G. Together plot W and set G specify a unique set of changes.

If the plot W of a process Z is specified by some complex pa, rather than by a scenario, then W and G may be insufficient to express exactly what occurred during a particular instantiation of Z. For example, consider the pa of Figure 10.11. From binding set G alone, it is not clear whether the process started by taking arc a or b. It is not clear how long control remained at +X+ or whether control left +X+ through arc c or d.

To help specify an instantiation of a process, a trace through the process' pa may be used. A trace for the pa of Figure 10.11 is given in Figure 10.12. This trace



A Complex Process Automaton

Figure 10.11

start - t_1 - a - X - t_2 - d - X - t_3 - c - finish

A Process Automaton Trace

Figure 10.12

string is read from left to right and has the following meaning:

1) Control was first located in +start+.
2) Control left +start+ at t_1 through a and arrived in +X+.
3) Control resided at +X+ (from t_1) until t_2 at which time it flowed through d and passed (again) to +X+.
4) Control left +X+ at t_3 through c and passed to +finish+, a terminal node.

Traces can, of course, be constructed to describe the "execution" of pas depicting simple processes. For example, the trace,

start - $t_{@}$ - h - finish

is the only possible trace through the pa of Figure 10.06. Since the binding set G specifies $t_{@}$, G uniquely defines the trace.

The trace

start - t_I - v - X - t_F - w - finish

is the only trace through the pa of Figure 10.07. Since t_I and t_F may be specified by G, G defines the trace.

For much of the work in natural language understanding, traces through complex pas are unnecessary. However, the times at which control leaves the start control point, t_I, and arrives at finish point, t_F, are often important.

10.3.6 Augmenting process automata

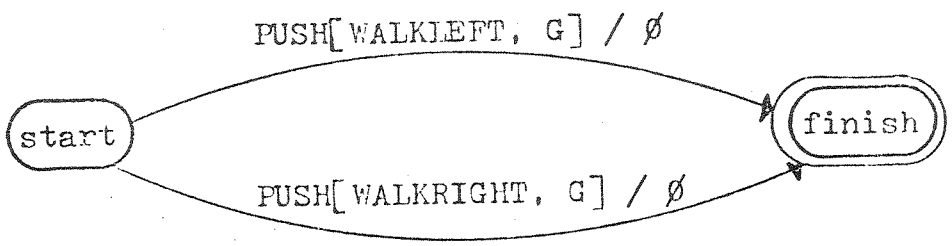
In his AFSTN work, Woods (1970) found that automata provide a very useful framework for describing natural language grammars. Further, he found that his grammar writing was made easier if ordinary automata were augmented with certain control features. Following his example, process automata may also be augmented.

Perhaps Woods' most useful deviation from the standard automaton is the PUSH link (which invokes "subroutines"). The PUSH link may be incorporated in a pa by replacing the i condition of a link k with the construct PUSH[W, G], where W is a reference to another process (i.e., to another pa) and G is a partial binding set. To cross the PUSH link, the process described by W must be successfully fulfilled with the parameters of S bound in accordance with the binding set G.

To see the utility of the PUSH link, consider the process of walking. In Section 10.2.2.2, a scenario was presented which described the WALKRIGHT process. A WALKLEFT process may be defined analogously. The process WALK is a generalization of WALKLEFT and WALKRIGHT which may be defined by the pa of Figure 10.13.

Beginning at +start+, control of this pa may cross either of two PUSH links. The upper is crossed if a WALKLEFT occurs, the lower if a WALKRIGHT occurs. The

pa name: WALK



parameter list: creature

where G = {(c, value(creature))}

A PA for WALK Using PUSH Arcs

Figure 10.13

parameter list of WALK specifies only the creature who walks. Set G indicates that the c parameter of WALKLEFT or WALKRIGHT must be bound to value(creature). Other parameters in WALKLEFT and WALKRIGHT may be bound freely.

Another deviation from ordinary automata concerns the flow of control at a control point. Ordinarily, a control point (cp) receives control whenever control is passed to the node through an incoming link. Further, the conditions ("inputs") on links leaving an ordinary cp are mutually exclusive and hence when control leaves an ordinary cp, it does so through exactly one outgoing link. In addition to ordinary cps, extraordinary cps may be introduced into a pa network to facilitate control.

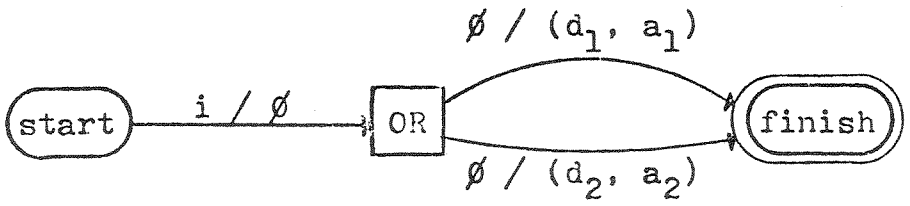
One type of extraordinary cp is the OR cp. Control enters an OR cp just as it enters an ordinary cp. However, control is only transient at an OR. (An OR is not used to encode a state of flux.) Immediately upon entering an OR cp, control is passed through any one of the OR's outgoing links. Since any link may be followed, a process automaton containing an OR is non-deterministic. The conditions or "inputs" of links leaving an OR are always $\emptyset$. Thus, all the arcs leaving an OR node may be taken unconditionally.

As an example of the use of an OR cp, consider the process of calling the throw of a coin in the creatures' world. The two creatures may decide to settle some

controversy by the process of flipping a coin. One of the creatures must "call the toss." That is, one of the creatures must choose a coin side upon which to bet. (If the coin lands with the called side up, that creature wins the toss.) The process of choosing a coin side upon which to bet is called "CALLIT".

The pa of Figure 10.14 is a description of the CALLIT process. The process is initiated when conditions i are met. That is, the process starts when some uncommitted creature c selects to CALLIT. When these conditions are realized, the pa steps to the OR cp (represented by a square). From the OR cp, the pa may cross either of two links to the finish cp. If the top link is crossed, the creature chooses HEADS. If the lower link is crossed, the creature chooses TAILS.

Two other types of extraordinary cps are →& cps and ←& cps, called "right ands" and "left ands" respectively. These cps often appear in pairs. Control enters a →& just as it does an ordinary cp. But the →& splits control into multiple parts and each link leaving a →& is considered simultaneously. A ←& reconsolidates the multiple paths of control created by a →&. Control does not pass from a ←& until control has entered the cp through all incoming arcs. All paths leaving a →& cp N must eventually join at a ←& cp M. No path leaving N may reenter N before passing through



i = {[≡SELECTED c ≡CALLIT],
[≡UNCOMMITTED c]}

$d_1 = d_2$ = {[≡UNCOMMITTED c]}

a_1 = {[≡CALLED c ≡HEADS]}

a_2 = {[≡CALLED c ≡TAILS]}

PA for Calling the Throw of a Coin Using an OR Node

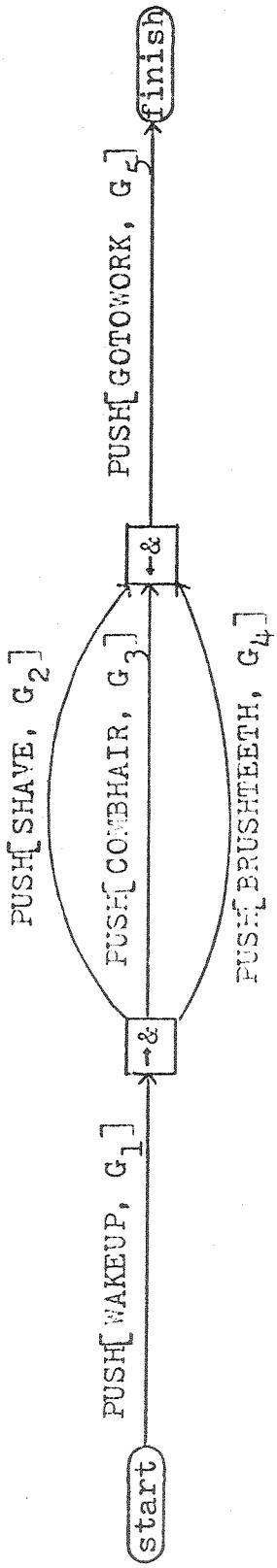
Figure 10.14

M. The use of →& cps in combination with PUSH arcs allows a high level process to invoke multiple, simultaneous subprocesses.

An example of a process automaton using &-type control points and PUSH links is the EARLY-MORNING pa of Figure 10.15. The process described by this pa consists of waking up followed by shaving, combing hair and brushing teeth (in any order or concurrently), followed by going to work. Before the GOTOWORK subprocess is initiated, the ←& cp calls for the SHAVE, COMBHAIR and BRUSHTEETH subprocesses to have all been successfully completed.

Yet another type of extraordinary cp is the Instantaneous (or simply I) cp. Control remains at an ordinary cp until the "input" condition of one of the cp's outgoing arcs is met. For an I cp, one of the outgoing arcs must be immediately transcendable or the process fails. That is, control is not allowed to dwell at an I cp.

As an example of the use of the Instantaneous cp, assume that every point in the creatures' one-dimensional world which is associated with an integer between 0 and 99 contains a telephone. Each phone is associated with a two digit number which, coincidentally, corresponds to the phone's location. Thus, the SWM of Figure 10.02 may be augmented by the statements



The EARLY-MORNING Process Automaton

Figure 10.15

```
(ELEM, PHONE0, PHONES)
(ELEM, PHONE1, PHONES)
  :
(ELEM, PHONE99, PHONES)

(AT, PHONE0, 0)
(AT, PHONE1, 1)
  :
(AT, PHONE99, 99)

(NUMBER, PHONE0, 0, 0)
(NUMBER, PHONE1, 0, 1)
  :
(NUMBER, PHONE99, 9, 9)
```

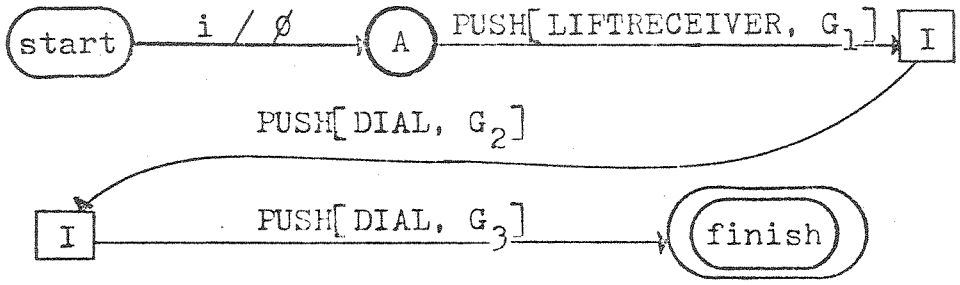
If a creature is at a location associated with an integer, it may use the telephone at that location. To place a call, the creature must dial the two digit number associated with the target phone. The point of interest here is that if the creature does not begin the process of dialing the first digit immediately after lifting the receiver, equipment at the telephone office (not shown) will determine that the phone is not in use and send out an "off the hook" tone. Further, the tone will be sent if the second digit is not dialed immediately after the first. Thus, the process of dialing another phone in the creatures' world is successful only if the dialing of the first digit

begins immediately after the lifting of the receiver and the dialing of the second digit begins immediately after the dialing of the first.

A process automaton for dialing one phone from another is seen in Figure 10.16. The pa for DIALUP makes extensive use of PUSH arcs to invoke the subprocesses of lifting the receiver and dialing a single digit. The binding sets G_1, G_2 and G_3 are used to send variable bindings to these subprocesses. LIFTRECEIVER has parameters for a creature c, a phone p and a location n. DIAL uses a creature c, a phone p, a location n and a single digit d.

The reader may well wonder what happens after the phone number is dialed. Keep in mind that DIALUP is only the dialing process. A MAKECALL process would indicate that either the remote phone begins to ring or a busy signal is returned. (Who could the other creature be talking to? He is arguing with the folks at the phone company (not shown), of course.) The MAKECALL process automaton could be an extension of the DIALUP pa, but more likely it would invoke DIALUP through the PUSH arc mechanism.

pa name: DIALUP



parameter list:

c, d1, d2, p1, p2, n

i = {[≡ELEM c ≡CREATURES],
[≡ELEM p1 ≡PHONES],
[≡ELEM p2 ≡PHONES],
[≡AT c n],
[≡AT p1 n],
[≡¬AT p2 n],
[≡¬INUSE p1],
[≡NUMBER p2 d1 d2],
[≡SELECTED c ≡DIALUP p2]}

G1 = {(c, #c), (p, #p1), (n, #n)}

G2 = {(c, #c), (p, #p1), (n, #n), (d, #d1)}

G3 = {(c, #c), (p, #p1), (n, #n), (d, #d2)}

PA for Ringing One Phone from Another

Figure 10.16

Chapter 11

Categories and Hierarchical Taxonomy

In the last chapter it was shown that the knowledge of change may be organized into processes. Process scenarios and process automata package sets of related changes into units which are meaningful to humans. Dealing with change in terms of these units, an artificial intelligence may become more adept at man-machine interaction.

Another systemization of knowledge which is utilized by humans, but which may easily be lost in an abstract analysis, is the structuring of knowledge about objects into a hierarchical classification system. Such a system provides a mechanism for compressing redundant information, predicting the values of missing data and abstracting.

Briefly, the hierarchical classification system classifies all objects into a number of categories or classes. Objects which are somewhat alike become grouped together, allowing similar objects to be thought about and talked about collectively. The system is hierarchical in that some categories are subcategories of more general classes. The lower a class is in the hierarchy, the more alike its members must be. The "likeness" arises in that

members of each category possess certain common, characterizing properties which are associated with (but not usually possessed by) the category itself. The hierarchy need not be strictly tree-like. A supercategory may have subclasses which are not mutually exclusive. Thus, an object x may be grouped into one subclass because of some aspect a and be grouped into a sister subclass because of another aspect b.

If it is known that a particular object x is a member of some category K, all the special properties associated with K are necessarily possessed by x. If some property p is possessed by each member of K, it is unnecessary to provide a separate, explicit encoding of the possession of p for each x in K.

For example, most humans would agree that baseballs are small round physical objects which are used for play. Thus, if x is known to be a baseball, it is known that x is round, that x is used for play and that x is a physical object. The class of physical objects is a superclass of the class of baseballs. Since x is a baseball and baseballs are physical objects, x must also have the usual properties of physical objects. Thus, x has mass, theoretically could be converted to energy in accordance with Einstein's equation and possibly has a color. The class of physical objects is a subclass of the class of objects. All objects have start times and end times. Thus x was created

(became a baseball) at some time t_1 and will be destroyed (cease to be a baseball) at some time t_2.

11.1 Categories and rules

The central notion of the hierarchical classification system is the notion of category. A category may be defined as a set which has as its members exactly those objects satisfying some rule R (or test T or predicate P). In general, there may exist several rules R which specify a given category (i.e., multiple "intentions" may relate to the same "extension").

One way of formally defining a test or rule is as follows. Suppose d is a template relation statement with indirect symbols $x_1, x_2, \ldots, x_n$. A rule h is an $(n + 1)$-tuple $h = (d, x_1, x_2, \ldots, x_n)$. Let H^h be a set of n-tuples having the property

$$(Z_1, Z_2, \ldots, Z_n) \in H^h$$

iff

$$\underline{\text{value}}_C(\{(x_1, Z_1), (x_2, Z_2), \ldots, (x_n, Z_n)\})^{(d)}$$

is valid. A sequence of objects $y_1, y_2, \ldots, y_m$ is said to obey (follow or satisfy) rule h iff

$$(H^h, y_1, y_2, \ldots, y_m)$$

is a valid relation statement.

It is important to realize that rules and general statements (Chapter 9) are very closely akin. Both use a

template containing variables. A general statement quantifies these variables (and hence implicitly assigns values to the variables) before the template is "evaluated." The general statement asserts the validity of the template statement for each such assignment of variables. A rule neither quantifies nor assigns values to its variables. (Note: The rule does not quantify x_1 to x_n of the definition. However, a rule may use internal quantification as in Section 11.3 where the template involves the special set $\overline{\exists}$.) Rather, the variables of a rule are assigned values by some external mechanism. The validity of the rule's template relation statement under a particular assignment of values y to its variables x determines whether or not the y obey the rule.

Rules used to define categories are characteristically one-place rules. For example, the class K of objects which are the children of some specific person P may be discriminated by the rule h = (d, x) where

$$d = [\exists \text{IS-A-CHILD-OF} \quad x \quad \exists P].$$

An object Z is a member of class K iff (H^h, Z) is valid. In turn, (H^h, Z) is true iff

$$\underline{\text{value}}_{\underline{C}}(\{(x, Z)\})^{(d)}$$

is valid. Thus, Z belongs to K iff (IS-A-CHILD-OF, Z, P) is valid.

There is, of course, a formal relationship

existing between a category and its defining rule (or rules). Let

$$(K, h) \in \text{NSRULE}$$

iff

K is a category,

h is a rule

and

for any object Z, $Z \in K$ iff (H^h, Z) is valid.

Notice that

(H^h, Z) implies $Z \in K$

and

$Z \in K$ implies (H^h, Z).

Thus, h is a necessary and sufficient rule (hence the name "NSRULE") of membership in K. If $(K, h) \in \text{NSRULE}$, h is called an ns-rule of K.

Weaker rules may also be associated with categories (although these weaker rules may not be used to define categories). Let

$$(K, h) \in \text{NRULE}$$

iff

$Z \in K$ implies (H^h, Z).

If $(K, h) \in \text{NRULE}$, h is called an n-rule of K and it is necessary that each $Z \in K$ obey rule h.

Let

$$(K, h) \in \text{SRULE}$$

iff

$$(H^h, Z) \text{ implies } Z \in K.$$

If (K, h) $\in$ SRULE, to prove that Z $\in$ K it is sufficient to show that Z obeys h. If (K, h) $\in$ SRULE, h is called an s-rule of K.

(In addition to ns-, n- and s-rules, certain additional rule types will be discussed subsequently to deal with delineation and abstraction.)

The utility of a categorical system is this: If an object Z is known to be a member of category K, then Z must obey all n-rules and ns-rules associated with K. Further Z must obey the n-rules and ns-rules of all categories of which K is a subset. This ability to encode information at the category level rather than with each individual is of practical importance because it saves computer memory and because all the elements of a category may not be explicitly known. That some Z belongs to a category K may be proven by showing that Z obeys any ns-rule or s-rule associated with K.

For natural language processing, the category system has the important feature that members of the more important categories are expressed by the same set of linguistic patterns. As an elementary example, screwdrivers, wrenches, hammers and saws belong to a category of objects which may be expressed by noun phrases headed by the noun

"tool." Various exchanging events (which belong to the same general category) may be expressed by complete sentences using the words "buy," "sell," etc., as their central verb.

11.2 An example of classification

To illustrate certain aspects of a hierarchical classification system, consider the four categories of geometric figures presented in Figure 11.01. These categories, POLYGONS, CONVEX-QUADRILATERALS, PARALLELOGRAMS and SQUARES, constitute a branch in the hierarchical tree.

No information is included in the figure to define POLYGONS. However, the other three categories are each associated with a supercat (i.e., a superset which is a more general category) and a number of rules. For ease of reading, these rules are stated in English.

From the figure, it may be seen that PARALLELOGRAMS is a subcategory of CONVEX-QUADRILATERALS. There are four ns-rules associated with PARALLELOGRAMS. If any x obeys these rules, then it is a parallelogram. Further any x which is a parallelogram obeys these rules. These four ns-rules could be broken up to produce five n-rules.

1) opposite sides of x are parallel
2) opposite angles of x are congruent
3) opposite sides of x are congruent

Category: POLYGONS

Category: CONVEX-QUADRILATERALS

supercat: POLYGONS

ns-rules: 1) x has four sides and is a convex polygon

Category: PARALLELOGRAMS

supercat: CONVEX-QUADRILATERALS

ns-rules*: 1) opposite sides of x are parallel and x is a convex quadrilateral

2) opposite angles of x are congruent and x is a convex quadrilateral

3) opposite sides of x are congruent and x is a convex quadrilateral

4) any two adjacent angles of x are supplementary and x is a convex quadrilateral

s-rules: 1) all angles of x are 90° and x is a convex polygon

2) the area of x is the product of any two of its adjacent sides and x is a polygon

Category: SQUARES

supercat: PARALLELOGRAMS

ns-rules: 1) x is a convex quadrilateral, all sides of x are congruent and at least one angle of x is 90°

2) x is a polygon and the square of any side of x is the area of x

A Hierarchy of Categories of Geometric Figures

Figure 11.01

*NOTE: The supercategory restriction is repeated in these ns-rules so that each rule constitutes a stand-alone sufficiency test for category membership.

4) any two adjacent angles of x are supplementary

5) x is a convex quadrilateral

However, since these rules are covered by the ns-rules, there is no need to include them in the description of PARALLELOGRAMS. That is, if x a parallelogram implies both that opposite sides of x are parallel and that x is a convex quadrilateral, then there is no need to have a rule that x a parallelogram implies opposite sides of x are parallel.

PARALLELOGRAMS is also associated with two s-rules. Both of these rules imply that x is a rectangle and hence a parallelogram. Of course, all the members of PARALLELOGRAMS do not need to obey the s-rule.

Suppose it is known that some object s is a square (i.e., suppose it is known that s $\in$ SQUARES). Then all n-rules and ns-rules associated with SQUARES and its supercats PARALLELOGRAMS, CONVEX-QUADRILATERALS, and POLYGONS are immediately known to be valid for s. This information need not be explicitly encoded for the single square s since it is implicitly encoded for all squares.

For ease of reading, the rules in Figure 11.01 have been expressed informally in English. To see how one of the rules may be formalized, consider the ns-rule of CONVEX-QUADRILATERALS

"x has four sides and is a convex polygon."

Let SIDECOUNT be a formal relation set of pairs (w, v) where (w, v) $\in$ SIDECOUNT iff w is a geometric figure with v sides. Let CP be the set of convex polygons. Then a formal encoding of the above rule is

$$h = (d, x)$$

where

$$d = [\exists AND\ [\exists SIDECOUNT\ \ x\ \ 4]\ [\exists ELEM\ \ x\ \ \exists CP]].$$

Thus, an object Z obeys the rule if both (SIDECOUNT, Z, 4) and (ELEM, Z, CP) are valid. (ELEM, Z, CP) is valid if Z $\in$ CP.

11.3 Attributes

One of the most important similarities among elements of the same category is that they possess common attributes and are usually distinguishable one from the other by differences in the values of these attributes. In general, attributes are always associated with categories of objects. If K and J are sets of objects such that every element of K is characteristically related to some element of J, then the elements of K are said to possess some attribute @b. Further, @b is associated with a binary relation set B, a subset of $K \times J$. For every $x \in K$ there must necessarily exist $y \in J$ such that

$$(x, y) \in B.$$

If (x, y) $\in$ B, then y is called the #@b of x. Alternatively, y is called the value of x's b attribute. (The symbol "#" may be read as "value of" and the symbol "@" as "attribute." Hence, "#@b of x" means "value of attribute b of x.")

For example, the elements of the set of physical objects K' are characteristically associated with the set of mass measures J'. That is, the elements of K' possess the attribute @b' of mass. Let B' = MASS be a subset of K' $\times$ J' so that

(x, y) $\in$ MASS = B'

iff x has mass y. Note that y is called the #@mass (the #@b') of x or said to be the value of x's mass attribute.

It is hopefully clear that associating attributes with a category is just a special case of associating rules. For example, that every k has a #@b taken from J may be encoded by the rule

"For x in K there exists (unique) y in J
such that (x, y) $\in$ B."

Formally, let

h = (d, z) be a rule

where

$d = [\exists\overline{\exists}\ \exists w\ \exists J\ *\ \exists[\exists^{\in}\ [z\ w]\ \exists^{B}]]$.

Despite the fact that attributes are a special case of rules, they are important in and of themselves because

their relatively simple structure (as compared with arbitrarily complex rules) makes them convenient vehicles for coding and manipulating information. Further, in many instances it is possible to specify an object "in a natural way" simply by noting its superclass and recording the values of certain of its attributes. Indeed, the attributes of many classes of events and situations are in a one-to-one correspondence with the surface language case structures of the verbs which express the events and situations.

Several properties of attributes as defined herein are rather subtle and deserve further discussion. Note that attributes always correspond to "there-exist" rules. In most (if not all) usages, the relation B associated with an attribute b is a mapping. Thus, for example, every physical object has a unique mass.

Since the unique value of an object's attribute is specified by a formal mathematical relation, it is not subject to change. Thus the #@b of any given object is always the same. This implies, for example, that color may not be an attribute (in the sense defined) of toy blocks since the color of a block may be subject to change (by painting). However, physical shape may be an attribute of a block, the block retaining this shape throughout its lifetime. (If a block is cut in half, it ceases to exist and two new blocks

are created. It remains true that the shape of the original block was, say, cubical.)

Perhaps this idea of timeless attributes is better illustrated by considering an event object such as a trading of goods by merchants. Associated with a trading event are certain attributes (in this usage sometimes referred to as "deep semantic cases") which include an @seller, an @buyer, an @thing-bought, an @occurrence-time, etc. A trade (the actual transfer of ownership, not the haggling) may be thought of as transpiring in a single instant (namely #@occurrence-time). Thus the event has a fleeting existence. But before, during and after the event, it is meaningful to talk about the trade's #@seller, etc. Despite the fact that the trade had a brief existence, the relationship between the event and the values of its attributes is constant.

The necessity of unique existence and the constance of attribute values lends credence to the notion that an object's attribute values are really subparts or components of a gestalt which is the object itself.

This general outlook is strongly reflected in the algebraic encoding of situations. For example, to encode the situation of one physical object being on top of another over a certain time period, the relation ON** may be used. The elements of ON** are 4-tuples of the form

(t_1, t_2, A, B).

Each such 4-tuple in ON** corresponds to a situation of an A's being on a B over an interval $[t_1, t_2)$. Note that the situation objects are represented by 4-tuple objects. Each 4-tuple is associated with four attribute values (corresponding to the four positions in the 4-tuple). Observe that for any given 4-tuple these values must exist uniquely and are not subject to change. Further, the 4-tuple is thought of as a composite of these individual values.

While being careful not to confuse the encoding 4-tuple with the encoded situation, it is clear that both have four principal attributes. (There are other attributes. The 4-tuple has a length of 4.) Now in defining a formal relation such as ON** it is convenient to begin with a statement such as

"Let ON** be a subset of $T \times T \times P \times P$
such that ..."

where $T \times T \times P \times P$ is the cross product of four sets. Even without knowing any additional information concerning ON** or the situations it encodes, much can already be said about them. Each 4-tuple in ON** and each modeled situation has four principal participants. Further, it is known that two of these participants must be times (elements of T) while the other two must be physical objects (elements of P). While this information in no way specifies ON** nor

the set of ON situations precisely, it does eliminate a vast array of possibilities and provides a basis for predicting missing participants if some of the participants of a situation are unknown.

In practice, the set of attributes associated with objects of a particular category (along with their corresponding ranges of values) may be used to delineate the category in much the same way that a cross product delineates a formal relation. For this purpose, certain meta-rules called "delineation rules" may be used. For a given category, these delineation rules indicate what attributes may be possessed by category elements and what ranges of values each attribute may assume. A discussion of the encoding of these rules is deferred until Division III.

11.3.1 Using delineations

In natural language processing especially during the parsing phase when surface structures are being translated into a deep semantic representations, the delineation rules discussed in the last section are particularly valuable. Consider, for example, the situation in which the parser has already determined that "bought" is the main verb. Knowing this, an AI system may consult its information concerning trading or EXCHange events to determine the associated attributes and their possible ranges of values.

Some or all of the values of these attributes should be conveyed by the input sentence.

Suppose the sentence fragment

"... bought the girl ..."

is spotted. "The girl" is a noun phrase sitting in a possible direct object position and (through verb paradigm rules) might be assigned to the attribute thing-bought. However, the delineation rules for trading events may reject this parse immediately by determining that "the girl" is not in the range of possible values for #@thing-bought (at least not in the machine's current model of ethics).

11.3.2 Predicting missing information

In the course of its activities, an AI system will often be forced to deal with objects which are only partially specified. If missing information can be guessed or at least delineated, many of the system's tasks become somewhat easier.

An important instance of this phenomenon may be seen by resuming the parsing example of the last section. Assume the parser works with a speech understanding system. As workers in speech understanding know, it is often a difficult task to map sounds onto words. However, it is relatively easy to discover whether or not a given word occurs in the sound pattern. (I.e., the verification of a word

is easier than its original recognition.)

Consider the situation in which the language system has recognized and successfully analyzed the fragment

"... bought candy at the grocery store."

Suppose further that the acoustic system has encountered difficulty in recognizing any words at the beginning of the sentence. Based upon the delineation rules, the system may determine that the missing information specifies the #@buyer of the EXCHange situation. Further, the possible range of buyers is known. Hopefully, context (discourse) considerations may subset this range so that a small number of words may be predicted and tested by the acoustics.

Rather than simply delimit ranges of attribute values, stronger "defaulting" rules may be written to tentatively assign values to attributes. While such default assignments are of little value in parsing, they may be quite useful in planning and understanding.

11.4 Abstraction

> Flower in the crannied wall,
> I pluck you out of the crannies,
> I hold you here, root and all, in my hand,
> Little flower - but *if* I could understand
> What you are, root and all, and all in all,
> I should know what God and man is.
>
> Tennyson, 1869

In his poem "Flower in the crannied wall," Tennyson

ponders the implications of having complete knowledge of a little flower, his conclusion being that to know the flower "all in all" he "should know what God and man is."

This poem has the important message for workers in artificial intelligence that only an omniscient entity could possibly know the "all in all" concerning any object. Faced with this limitation, methods for drawing out and organizing the "most important" aspects of an object must be found. This process of focusing on important aspects while suppressing detail is known as abstraction.

11.4.1 Abstracting by focusing on properties of classes

One kind of abstraction is produced by the classification system directly. If the only information known about some object Z is that it belongs to category C, then the n-rules, ns-rules, attributes and delineation rules associated with C provide an abstraction of Z, the detailed information concerning Z's individuality being unknown. This kind of abstraction is really forced upon Z by the unavailability of additional information.

A second kind of abstraction may be illustrated by the following: Suppose Z is in category C2 which is a subcategory of C1. Some information concerning Z may be explicitly recorded with Z while more general pieces of information may be associated with C2 (which inherits the

information associated at C1, etc.) and with C1. The concept Z may be abstracted by ignoring specific information and considering only the information arising from Z's membership in C2. This information may be further abstracted by considering only information arising from Z's membership in C1. This kind of abstraction is not forced by a lack of information. It is a deliberate method of weeding out the finer, distinguishing details of category members. The information concerning an object Z which is derived solely from Z's membership in category C is called the "abstraction of Z with respect to C."

In the hierarchical taxonomic system, subclasses of a given class C need not be mutually exclusive. Hence certain classes may have both individual elements and complete subclasses in common. Suppose Z is a member of both subclass C1 and subclass C2 where neither C1 nor C2 is a superclass of the other. Then neither A_1, the abstraction of Z with respect to C1, nor A_2, the abstraction of Z with respect to C2, need be an abstraction of the other. To see the utility of such "multiple, parallel abstractions," let object Z be the alternator on the family car. The alternator is in both the class of electrical components and the class of mechanical components (since the alternator receives mechanical power from the engine and converts it to electrical energy). For solving certain problems it may be

useful to think of the alternator as an abstract electrical component while for other problems it may be more convenient to focus on the alternator's mechanical properties. The alternator may be placed in both the mechanical parts subclass and the electrical parts subclass of the automobile hierarchy. It may then be appropriately abstracted for mechanical and electrical considerations.

11.4.2 Abstracting by rule partitioning

The abstraction schemes just presented suffer from a common inflexibility. If an object Z is to be abstracted with respect to class C, all information more specific than the properties associated with C must necessarily be lost while all properties associated with C and its superclass must necessarily be known. The category mechanism records the properties of an object Z at various points along a branch in the taxonomic "tree," with more abstract information recorded near the root of the tree. The type of abstracting presented in the last section corresponds to clipping off a branch in the tree somewhere between the root and the leaves. For many applications, it would be better to split the branch, retaining some of the detail associated with each joint (category) along the path from leaf to root.

An example of a situation in which branch splitting

would be useful is the description of auto alternators discussed above. Recall that the electrical properties of an alternator could be abstracted from a total representation (including mechanical considerations) by viewing the alternator as simply an electrical component. But an alternator is a very special type of electrical component with unique electrical features which are lost by this abstraction. A method of splitting out certain electrical details while suppressing (as much as possible) the mechanical details is needed.

Abstractions of the "split branch" variety may be produced by partitioning the rules associated with each category. For the alternator example, one partition might include rules pertaining to mechanical characteristics while another partition might contain electrical rules. For "complete" knowledge, all the rules could be invoked. For special abstractions, only the pertinent rules need be considered. (There is, of course, a formidable problem in knowing when what abstractions should be used. Conceivably, routines which do specialized question answering and problem solving might be associated with lists of partitions whose rules are pertinent.)

The partitioning of rules for split branch abstraction also has applications in planning. Suppose that for purposes of planning, a (linear) hierarchy of "abstraction

levels" is defined (following Sacerdoti's "abstraction spaces," 1974) by assigning level tags to category information (rules) and to explicit representations. Each "level" corresponds to a level of detail at which objects (including physical objects and processes) are viewed. The planner first solves its problem on the most abstract of these levels, then resolves the problem at each finer level using the higher plan as a guide.

The various levels are encoded by using some type of abstraction method. For this purpose, split branch abstraction has two important advantages over cut branch abstraction. The first advantage may be seen by considering objects x and y at two different levels of detail. At a high level H, some predicate P may apply to both x and y. However, P may be an abstraction of two lower level predicates p_1 and p_2. At a lower level L, p_1 may apply only to x and p_2 only to y. While the high level plan treats x and y alike using P, at the lower level the additional information concerning p_1 and p_2 must be used. With cut branch abstraction, P would still be around cluttering up the model (the new details simply being added to the old.) But the split branch mechanism allows P to be forgotten at level L and allows attention to be focused on p_1 and p_2. That is, the predicate P, while existing at a high level of abstraction, does not even exist on the less abstract

levels. The second advantage is similar to the first. At one level it may be convenient to think of certain properties in terms of a predicate p_1. At another level, the same properties may be more conveniently stated in terms of p_2. Split branching allows predicates p_1 and p_2 to exist on separate levels without interfering with one another.

11.4.3 Answering questions at the appropriate level

Given the ability to represent, consider and answer questions at varying levels of detail, there arises the problem of deciding at which level to respond.

For simple question answering tasks, it is probably appropriate to respond at the level of the question. The "level of the question" is determined by considering how high the concepts expressed in the question are in the taxonomic hierarchy. For example, questions posed in terms of the verb "fasten" are somewhat more abstract than questions posed in terms of the verb "nail."

QUESTION: How do I fasten the brace to the wall?
ANSWER: Nail it.

QUESTION: How do I nail the brace to the wall?
ANSWER: With a hammer.

For consultant systems in which answers are typically instructions, the answer should be rendered at "the user's level of competence." This, of course, assumes a model of the user.

In any event, a user may ask for more or less detail. (E.g., "Tell me more about ...") Such requests may be used to trigger changes in the user model.

Chapter 12

The Integration of Process Automata and the Taxonomy

The last two chapters have shown that the categorization scheme and the notion of process are useful tools for organizing knowledge. Of course, these tools are all the more interesting when they are used together like a hammer and chisel.

12.1 Interactions in definitions

Perhaps the most obvious interplay between these two knowledge organizers is provided by the use of process automata in defining classes of events. For this purpose ns-rules may be defined which state that an event belongs to some event category C iff the event follows (followed) the playscript encoded by the process automaton. Thus the definition of an event class may be posed solely in terms of a process.

Theoretically, even objects which are not normally considered to be events may be partially specified by processes. Being human is to engage in the process of being born, growing up, falling in love, etc. Being an island is to emerge from the sea (perhaps by violent volcanic action), to acquire new life forms, to be visited by pirates, etc.

Conversely, a process may be partially specified by

reference to an event class. The PUSH arcs of pas in effect say "an element of this event class must exist (come into existence) before this arc may be crossed."

12.2 Levels of detail in process

Paralleling the abstraction remarks of the last chapter, an AI system may find it convenient to consider a process at various levels of detail. To understand the meaning of the statement

"Otis traveled from Austin, Texas, to Yarmouth, Nova Scotia,"

an AI entity might consult a process automaton describing traveling events. Such a pa might take into account all the possibilities of travel by land, sea and air which Otis may have used on his journey. The pa might also take into account the purchasing of tickets, the making of reservations, the packing of baggage, lodging and side trips, etc. The very richness of detail which would make such a travel pa suitable for some tasks (such as inferring what Otis might have done along the way) makes it cumbersome for other tasks. For example, for many tasks it may be important only to know that Otis was in Austin at some time t_1, that he was traveling from Austin to Yarmouth from t_1 to t_2 and that he was in Yarmouth at t_2.

Following the notion of abstraction levels

presented earlier, the events belonging to a given class may be presented at different levels of detail. Very simple process automata may be employed by the rules in the higher levels of the rule partition and more complex pas may be used by rules defined for more detailed levels.

By virtue of the PUSH arc mechanism of pas, it happens that many levels of detail may be effectively encoded for most event classes by associating only two levels of pas with each category of events. One of the two pas associated with a category is always very abstract. This pa may be restricted to process descriptions encodable by instantaneous or simple duration scenarios. The more detailed pa is as complex as necessary, but uses PUSH arcs wherever possible. The PUSHes simplify the pa in that some of its detail is specified by the pas describing the subprocesses which are pushed to.

Now the merit of this scheme is as follows: For an abstract statement of an event X, consult the high level automaton. For more details, consult the low level automaton. In consulting the low level automaton for X, PUSHes to several subprocesses Y_1 through Y_n are encountered. To understand X at the current depth, consult the high level pas for each Y. To increase the detail of X, use the detailed pas associated with some or all of the Ys. The detailed Y pas will have PUSHes to Zs which may be evaluated

at either a high or low level of detail, etc.

12.3 A second type of rule involving process automata

At the beginning of this chapter it was seen that a process automaton PA may be used in defining the rule

"x follows the process plot encoded by PA."

A rule h_m of this type is used to define a category whose members are instantiations of the plot encoded by PA. Rule h_m is called a pa-match rule since, to obey the rule, a process instantiation must match the changes encoded by the process automaton.

A second type of rule involving process automata uses a pa to define a process which tests whether or not some x belongs to a particular category. A rule h_t of this type is called a pa-test rule. In general, such rules may be stated as

"the procedure encoded by process automaton PA may be successfully applied to x."

An x obeying this rule need not (necessarily) be the instantiation of a process. In particular, x is not (in the usual case) an instantiation of the plot encoded by PA. Rather, x is some object which may successfully be cast in one of the roles of PA.

To see the difference between pa-match rules and pa-test rules, consider a process J which determines

whether or not a given number is prime. Assume that process J is modeled by a process automaton, PAprime. The parameters associated with PAprime include a number n, a tester t and a result r. Instantiations of PAprime are events in which a tester t performs certain operations involving a number n. (The nature of these operations is specified by the internal structure of the pa.) Depending upon the outcome of the operations, the tester t arrives at a result r (either TRUE or FALSE) which indicates whether or not n is indeed a prime.

Automaton PAprime may be used to define both set I, the set of instantiations of the prime determination process, and set P, the set of prime numbers. Set I is defined by a pa-match rule while P is defined using a pa-test.

12.4 Active participation in processes

The last section alluded to a process J for determining whether or not a given object is a prime number. Based on J, a pa-test rule for prime numbers was discussed which indicates that an object n is a prime number if some tester t has successfully performed the process J on n. A problem arises in that there may be some objects n for which no tester has bothered to perform the process J. Hence, it may be impossible to classify n until some tester

can be found who is willing to perform J on n.

This situation often arises in the academic world when considering whether or not some newly arrived paper is worth reading. Often one can find a tester (a colleague) who has performed a testing process upon the paper (i.e., read and evaluated it). Given the result of the evaluation process, one knows whether or not the paper is worth reading. But more often no one trustworthy has yet read the paper. In this situation, an academician may either wait for someone else to evaluate the paper or perform the evaluation process (by reading the paper) himself.

Analogously, it would be convenient if an artificially intelligent entity could itself perform tests coded by pas. In general, it would be convenient for an AI entity to be able to perform both testing and non-testing processes. (Most processes required of a natural language system are test-like processes. E.g., the parsing of a sentence is a test to see if the string of characters does indeed constitute a sentence.)

Further, it would be convenient if the processes which an AI entity does indeed perform could be encoded as pas which are accessible for inspection (and introspection).

The relationship between a process automaton and an actual performance of the process modeled by the pa is rather subtle. In particular, a pa does not necessarily

constitute a program or algorithm for the performance of a process by the computer. For example, there may be a pa in the data base which describes the process engaged in by a bird during flight. Assuming that the AI entity is not a bird robot, the entity cannot itself engage in the process of bird-like flight.

However, the bird-flight pa may be investigated by the AI entity to determine the nature of flight by birds. An examination of the pa would indicate that the process requires a bird to flap its wings and results in the bird moving through the air. The AI entity might use this information to construct a sequential simulation of the flight of a bird and exhibit the simulation as moving pictures on a cathode ray tube. The pa might indicate that the bird first jumps into the air. (The JUMP pa may be called upon for details.) Then the bird flaps its wings. The flapping of wings causes the bird to move forward, etc. Such a simulation may be a model of either a past, present, future or imaginary flight.

Using a process automaton, an AI entity may simulate any kind of process at varying levels of detail. In particular, the entity may simulate activities in which it itself is a participant. But simulating itself performing processes (even if the simulation is of a process taking place in the present) does not necessarily mean that the

entity does indeed perform the process. For example, at this very moment the reader (if he so chooses) may imagine (internally simulate) himself basking in the sun on some exotic tropical island. Further, he need not imagine himself in this situation yesterday or tomorrow, but may imagine himself involved in it right now. But this simulation involving himself does not cause him to actually sunburn.

On the other hand, the reader may imagine (internally simulate) himself reading this sentence. Despite the fact that the reader can read the sentence, he cannot fully simulate (i.e., simulate down to the last detail) the process of reading since he does not himself know the precise details of the working of his mind and eyes. Happily, the reader does not need to know exactly how he reads. It is usually enough to know that one sequentially scans the words of the sentence and a short time later the meaning of the sentence registers upon the mind. The point is that although he cannot completely simulate his own reading process, he can switch on certain internal black boxes which cause the reading to take place. The result of switching on the black boxes is the enactment (as opposed to simulation) of the reading process. This real reading process may follow the reader's model of reading (as far as the model goes), but by virtue of being real will contain intricacies and details not accounted for in the reader's

model of his reading.

The important thing to note here is that the black boxes provide the link to external reality. If an AI entity is to affect the external world, it too must possess such boxes. These boxes (bodies of code and appropriate hardware) for realizing certain basic processes may be associated with categories of events. To engage in a particular category of event, the entity turns on the corresponding box.

Now black boxes need be furnished only for primitive processes. Executable higher level processes may be described by pas using PUSH arcs which (at some level) call only these most basic primitives. The high level pas may then be "interpreted" using the primitives.

Certain categories of events may have both a black box and a pa. Under these circumstances the pa describes the black box process. The pa might describe the process in terms of subprocesses which are at such a low level that they cannot consciously be activated. Such a pa may be used in a simulation, but not interpreted. A second possibility is that the pa may indeed be interpreted. Activation of the black box would perform some learned task such as riding a bike. Interpreting the pa would cause the bike to be ridden, but the AI entity would have to concentrate its attention on the details of pumping the pedals and

balancing.

12.5 Relating process automata to the work of Woods and Schank

Before closing this division, it is important to note the work on representing process knowledge of Woods and Schank.

There is a strong resemblance between process automata and the AFSTN systems of Woods (1970b). There is, however, a fundamental difference in their intent. AFSTN was developed as a programming language to describe the process of parsing. The process automaton, on the other hand, was developed as a data structure for describing the processes that are cited by language. As seen in Section 12.4, under certain conditions a process automaton may be interpreted (i.e., executed) in much the same way that an AFSTN is interpreted. Indeed, there is no reason why a pa may not describe the process of parsing or of generation and be interpreted to produce the desired results. Further, since pas may describe continuous as well as discrete processes, they may be better suited than AFSTNs to represent the processes of continuous human speech (although no experiments in this area have been attempted).

As an alternative to the method of recording process knowledge through operators or process automata, Roger

Schank and his associates (Schank, 1972 and 1973a; Rieger, 1974) have described change in terms of a small number (sometimes 12, sometimes 14) of primitive actions, with instantiations of complex events being represented by composite structures encompassing several primitives. This scheme is attractive in that a program using the scheme need encode information about only a small number of event types.

The process automaton view of change appears to have at least three advantages over the primitive action scheme. First, process automata allow even the simplest actions to be decomposed into their initiation conditions and effects. Further, this decomposition information is given in terms of the situations (static relations) which define states of the world. Second, it is possible to define complex event types which have many sub-events as subprocesses. (Such complex events may include loops of sub-events such as the continuous alternation of leg movement during walking.) Third, processes may be viewed at multiple levels of detail rather than always in terms of primitives. This allows complex event types to be encoded and manipulated at higher levels of abstraction. Thus, for example, the details of leg movement may be suppressed in the description of walking events when such details are not needed.

The second advantage cited above makes a critical difference between the ability or inability to represent the meaning of certain English sentences. Consider, for example, the sentence

"Mary went shopping."

To replace the concept of shopping with primitives such as physical movement (PTRANS) and change of ownership (ATRANS) would require a knowledge of how many stores Mary visited. That is, one would need to produce a sequence such as

MOVED Mary from HOME to SHOP1

MOVED Mary from SHOP1 to SHOP2

BOUGHT Mary ITEM1

MOVED Mary from SHOP2 to SHOP3

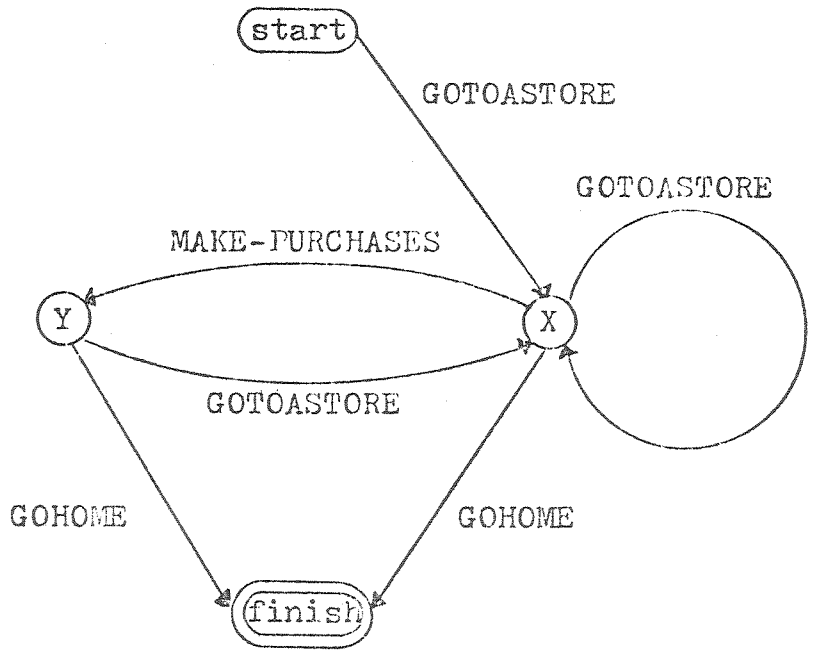
BOUGHT Mary ITEM2

MOVED Mary from SHOP3 to HOME.

But the simple statement "Mary went shopping" does not supply this detailed information. Only the existence of some sequence like the one above is implied.

Using process automata, a category of shopping events may be defined in terms of a pa-match rule which calls upon a process automaton, say SHOP, to define shopping. Mary's shopping is then recorded as an instance of this event category in which the shopper is Mary.

A possible encoding of pa SHOP is shown in Figure 12.01. A shopping is herein defined as any sequence of



Sketch of Process Automaton for SHOP

Figure 12.01

change which causes automaton control to pass from +start+ to +finish+. Hence, if all we know is that "Mary went shopping," then we know only that there is some path from +start+ to +finish+ with Mary bound as the #@shopper. Looking at the automaton, it is seen that Mary must have gone to a store and then eventually gone home. But the automaton contains optional loops which allow Mary to have made numerous purchases and called at other stores.

The sequence of MOVED and BOUGHT events shown above is only one possible trace through the automaton. Using the approach just described, the fact that some path exists through the pa for Mary may be asserted without commitment to any particular path.

DIVISION III

Considerations for Computer Based Modeling Systems

Chapter 13

Goals of Division III

Divisions I and II have shown how information may be encoded mathematically and how it may be usefully organized. Building upon this base, Division III discusses the encoding of knowledge in computer data structures.

The division begins with a search for an economical, theoretically adequate and computationally convenient encoding scheme. This search leads ultimately to networks. The balance of the division is devoted to formalizing network notation and to encoding rules, categories, processes and other objects in network structures.

Chapter 14

Selecting a Data Structure

One of the first considerations for encoding information in a computer is the selection of a suitable data structure. In considering possible structures for representing information pertinent to natural language understanding, two considerations are paramount. First, the data structure must be capable of representing the complete range of pertinent data without distortion of meaning. Second, stored information must be organized in such a way that facts pertinent to the answering of questions are easily accessible. A third consideration, as always, is economy.

An early consideration in selecting a basic representation structure for a modeling scheme is the supposed choice between procedural and assertional representations. For use in computer systems (as opposed to static applications), this is really not so much a choice between one or the other as it is a choice of balance between the two alternatives. Procedural systems such as Hewitt's (1971) encode only a portion of the system's knowledge procedurally. This procedural portion corresponds to a set of theorems which may be "evoked" in clever and heuristically

guided ways to manipulate knowledge maintained in an assertional cache. Thus, a portion of a procedural system is assertional. Conversely, a portion of an assertional system is procedural. For example, part of the knowledge in implementation of Robinson's resolution system (1965), namely the knowledge of modus ponens, is procedural.

The argument just cited concerning the necessary coexistence of both assertional and procedural information in computer systems does not carry over to representation systems per se. To simply represent information (without manipulating it or tying it to "reality" through sensory or motor devices), assertional encoding of knowledge is adequate. For example, the knowledge in resolution systems may be totally represented assertionally by including an assertional description of modus ponens. The knowledge of a system using procedural encodings may be made "assertionally complete" by encoding assertionally (and hence redundantly) those theorems which have been encoded procedurally. This redundant encoding may have practical benefits. For example, the procedural encoding of the knowledge "HUMAN(x) implies MORTAL(x)" will be adequate to answer the explicit and specific question "Is Socrates, who is human, mortal?" but can it answer "Does to be human (existentially) imply to be mortal?" which appeals to the meta information encoded by the theorem? The assertional statement of this

theorem may be used to answer both queries.

Since even procedural systems use assertional representations and since procedural information may be encoded assertionally, it seems reasonable to adopt an assertional encoding as the basic representational paradigm. (This does not mean that a computer system based on this paradigm will not wish to encode some information procedurally for efficiency purposes.) Given that knowledge is to be represented assertionally, there still remains a wide variety of choices. English or some other natural language could be used to describe assertionally the knowledge in a system. Other alternatives include the formal languages of logic such as predicate calculus and the formalisms of Division I, tabular representations following Codd (1970), and semantic networks.

One has only to consider the multitude of documents (book, letters, notes, etc.) recorded in natural language to see that adequate systems for knowledge representation have existed for centuries. However, since the semantics of natural language is so poorly related to its syntax and since most of what is communicated by natural language is not explicitly recorded, this medium is, unfortunately, unsuitable for the internal representation of knowledge in computers.

Turning to the other alternatives, tabular systems

are out because, among other defects, they have no provisions (in themselves) for encoding quantified statements. (Tables are, however, very efficient devices for encoding specific information. Typically, the elements of a formal relation may be stored as rows in a table representing the relation set. Since a table may encode the relation of links in a net, a table may encode a semantic network which in turn encodes arbitrary information. This is, of course, an indirect use of the tabular mechanism.)

With tables eliminated, only formal languages and semantic networks remain as viable options from the limited set of known representation structures. It has been argued that the constructs discussed in Division I provide an adequate (if sometimes clumsy) encoding of knowledge. Thus, the search for a data structure meeting the criteria of completeness, accessibility and economy may be launched by investigating methods for encoding formal constructs such as sets and n-tuples. As will be seen, this search will lead ultimately to the adoption of semantic networks.

14.1 A first attempt

As a first attempt at the computer encoding of knowledge, consider how the statement

(DOOR, ROOM1, ROOM2) $\in$ CONNECTS

might be represented. It is possible to encode this

statement by the string of characters

"(DOOR, ROOM1, ROOM2) ∈ CONNECTS".

A sequence of character codes representing this string may be kept in an array in a computer's memory. However, it becomes very difficult to find the string when needed. That is, there is no simple, inexpensive way to find all records which concern the relation set CONNECTS or the objects DOOR, ROOM1 and ROOM2. (For special cases this fault may be partially corrected by sophisticated hash coding.) Given the sequence of character codes, it may still be difficult to interpret what the string means. That is, the character code strings for "CONNECTS", "ROOM1", "ROOM2" and "DOOR" convey no direct information concerning objects CONNECTS, DOOR, ROOM1 or ROOM2. To interpret the codes, it is necessary to look up each character code sequence in a directory. A directory entry for the character code sequence "CONNECTS" indicates where information concerning the relation set CONNECTS may be found.

14.2 From strings to lists of pointers

The statement (DOOR, ROOM1, ROOM2) ∈ CONNECTS might also be represented by a list such as

(-CONNECTS- -DOOR- -ROOM1- -ROOM2-)

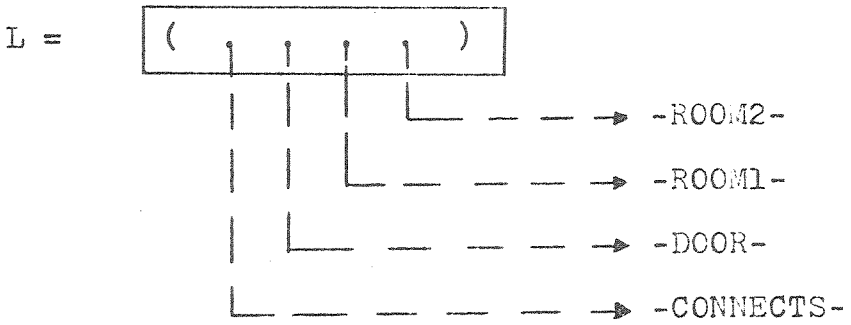
where -<string>- designates a LISP-type atom. This list corresponds to a relation statement n-tuple. In a language

such as LISP, this statement could be stored in a construction such as that shown in Figure 14.01. Figure 14.01 shows only "content" links (various CARs). Certain links used to produce the list structure (for LISP, the CDRs) have been omitted. The broken lines used to represent the links indicate that links may be followed only in the direction of the arrow.

This list structure is superior to the string notation above in that each component of L points directly to an atom (or, in other instances, to an S-expression) which models the associated n-tuple component. For example, the first component of L is a link to the atom -CONNECTS-, the model of CONNECTS. Information regarding CONNECTS may be accessed through this atom (perhaps through a property list). This structure still suffers from the inability to easily find L given DOOR, ROOM1, ROOM2 or CONNECTS.

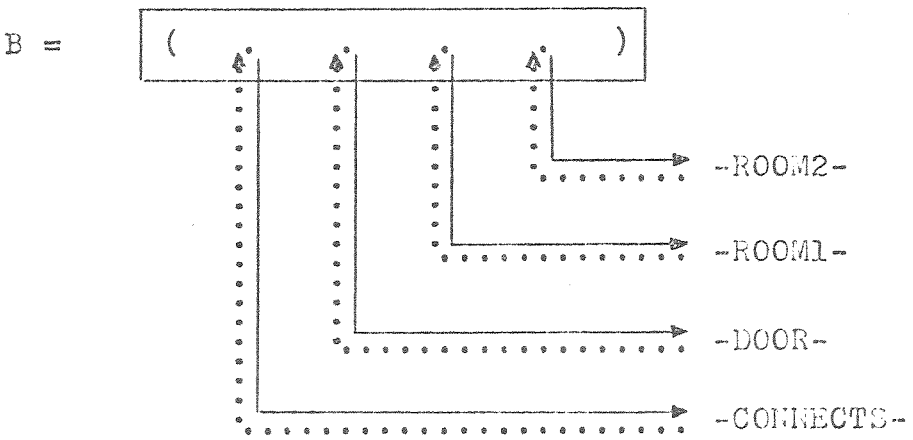
14.3 Backtrackable links

If the links from L to its various components were backtrackable, they could be followed from the components back to L. This situation is depicted by the backtrackable structure of Figure 14.02 where solid arrows depict backtrackable links. The links are backtrackable in that they may be followed in either direction. However, the links remain directed. To backtrack is to trace a link in its



A One-Way List Structure

Figure 14.01



A Backtrackable List Structure

Figure 14.02

reverse direction (shown here by dotted lines).

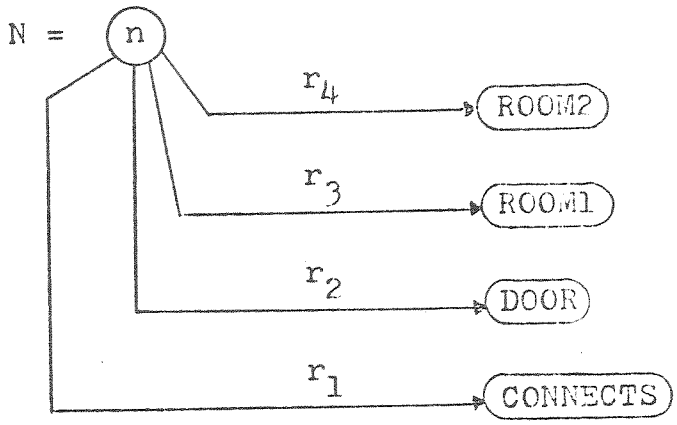
(The term "backtrack" as used here may be interpreted rather broadly. The backward pointing might be accomplished explicitly (by, for example, storing with each LISP S-expression (including atoms) a list of all S-expressions in which the given expression appears (the inverted file approach)) or indirectly (by, for example, a QLISP-like discrimination net). For natural language usages, the explicit mechanisms are probably superior since they are the most efficient for handling the frequent problem of finding all known information concerning some object X.)

With its backtrackable links, the data structure exemplified in Figure 14.02 meets the important criterion of being easily accessible.

14.4 Networks

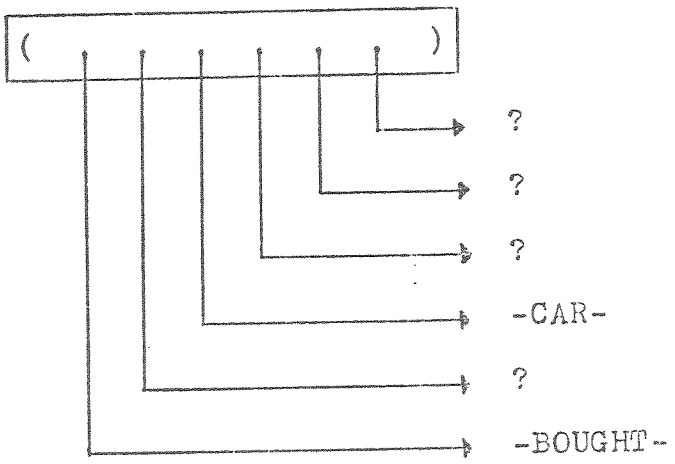
Notice that the links leaving structure B are necessarily ordered. (Clearly (-CONNECTS- -DOOR- -ROOM1- -ROOM2-) is not the same as (-ROOM1- -CONNECTS- -DOOR- -ROOM2).) Rather than order the links, one link may be distinguished from another by labeling. Consider, for example, the structure shown in Figure 14.03.

The structure of Figure 14.03 is a network. The node N labeled "n" represents the relation statement n-tuple (CONNECTS, DOOR, ROOM1, ROOM2). Backtrackable arcs



A Net Structure with Labeled, Backtrackable Links

Figure 14.03



A Backtrackable List Containing Unknowns

Figure 14.04

connect N to nodes representing CONNECTS, DOOR, ROOM1 and ROOM2. Since the arcs are labeled, they need not be given in any particular order.

Except for the fact that structure N uses labeled backtrackable links (called arcs) while structure B uses ordered backtrackable links, these structures are really identical. It is probably fair to say that the ordering of links is more economical than arc labeling, thus making ordered backtrackable linked lists appear the more economical data structure. However, arc labeling has some important advantages over ordering. The primary advantage of labeling is that different labels may be assigned meanings. Rather than being shackled to position numbers, designers may use such labels as "agent", "goal", "from-location", etc. These meaningful labels may be used to facilitate efficient, intelligent associative retrieval operations.

Now a mathematician might well argue that meanings could just as easily be assigned to positions within n-tuples. The problem with assigning meanings to positions is that dozens of separate meanings may be desired. Assigning each of these meanings a separate n-tuple position would make all the encoding n-tuples very long. Further, since for any given object encoded by an n-tuple most of the meanings would not be appropriate, most positions in most n-tuples would contain NIL markers. And adding one

new meaning would force the updating of all the n-tuples. One of the advantages of arc labeling is that inappropriate arcs may be omitted without confusion.

Another facet of easy arc omission is seen when considering missing (i.e., unknown) information. In natural language usage, the ability to efficiently omit pieces of information from the recording structure becomes very important. For example, the relationship BOUGHT may be defined as a set of 5-tuples. Thus

(JOHN, CAR, TOM, $500, TUESDAY-MORNING)

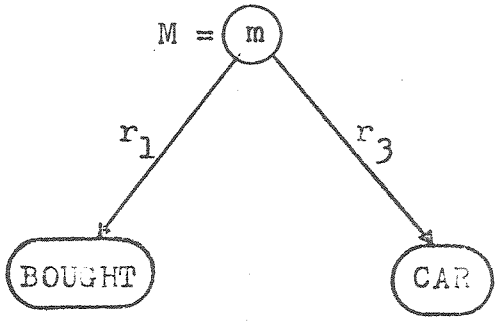
$\in$ BOUGHT

iff JOHN bought a/the CAR from TOM for $500 on TUESDAY-MORNING. If a natural language machine is told only that "JOHN bought a CAR" or worse, that "The CAR was sold," it is inconvenient to have to explicitly record "question marks" for all of the unknown components of the transaction.

To see this pictorially, consider Figure 14.04 which encodes "The CAR was sold" by a backtrackable list. The structure must save places for four unknown components of the transaction.

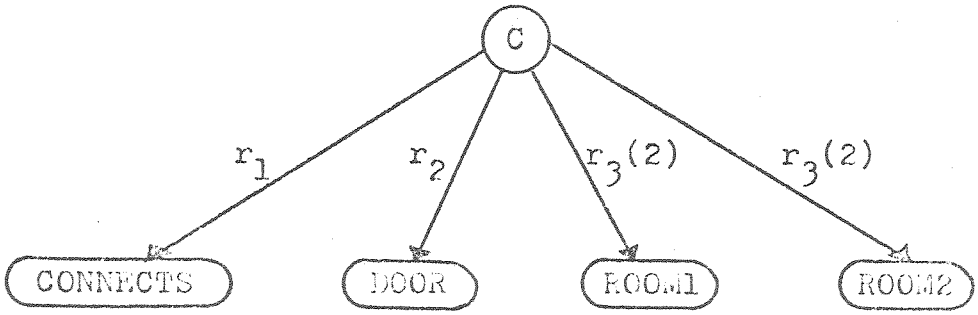
The same information is encoded in Figure 14.05 using network notation. Node M encodes the partially specified n-tuple statement

(BOUGHT - CAR - - -).



A Net Structure with Missing Information

Figure 14.05



The Network Encoding of a Cluster

Figure 14.06

That the first component of M is BOUGHT is indicated by the r_1 (first component) arc from M to the node modeling BOUGHT. That a/the CAR was bought is indicated by the r_3 arc. Unknown information is not considered. Note that the network structure uses two links while the backtrackable list uses six. (In fairness to the list structure, back pointers would probably not be implemented for the marker "?".)

14.5 Networks encoding clusters

An additional advantage of the labeled arc scheme is realized when n-tuples are replaced by "clusters." In an ordinary n-tuple $N = (x_1, x_2, \ldots, x_n)$ there is a unique i^{th} component x_i. If this requirement is lifted, the resulting structure is called a cluster. The cluster

$$C = (x_{1,1}, x_{1,2}; x_{2,1}; x_{3,1}, x_{3,2}, x_{3,3})$$

has two first components, one second and three third components. By using clusters rather than n-tuples, the network notation becomes more efficient.

For example, with regard to the CONNECTS relation, it is obvious that the following two statements are both true.

$$(\text{DOOR}, \text{ROOM1}, \text{ROOM2}) \in \text{CONNECTS}$$

$$(\text{DOOR}, \text{ROOM2}, \text{ROOM1}) \in \text{CONNECTS}$$

Using ordinary n-tuple notation there is no easy way to

combine these statements. However, the cluster relation statement

(CONNECTS; DOOR; ROOM1, ROOM2)

(which is identical to (CONNECTS; DOOR; ROOM2, ROOM1)) does combine the two statements. Ordered backtrackable linked lists may not be used to (directly) encode this cluster since the cluster's third components are inherently unordered.

The cluster may, however, be encoded by the network of Figure 14.06. Arcs labeled "$r_3(2)$" indicate that there are two third components. The two objects specified by these arcs are distinguishable, yet unordered. Ordinary arcs labeled "r_1" and "r_2" indicate the unique first and second components. Hence the network encodes a cluster statement which carries the combined meaning of the two n-tuple statements cited above.

14.6 Summary

Considerations presented above indicate that networks show substantial promise as a data structure in which to encode a model of meanings for natural language utterances. They are of sufficient power in that they may mimic the formalisms of Division I or the predicate calculus. (The encoding power of nets will be extended subsequently by introducing node space partitioning.) They are

inherently associative. Thus related pieces of information are interlinked (perhaps through a chain of arcs) and are accessible one from the other. Further, arc labeling allows meaning to be assigned to certain links (which facilitates processing efficiency) and unused or unknown arcs may be omitted without confusion.

The chief rivals to semantic networks are encodings of formal language assertions through LISP-like S-expressions. While such assertions in formal languages are easily manipulated once in hand, a computational problem arises in finding those assertions which are relevant to the execution of specific tasks. This problem of cross-indexing has been partially solved by matching techniques (as exemplified by QLISP) which find all assertions in a given data base which follow a particular pattern. This matching is accomplished by appealing to a discrimination net or an inverted file which serves as an index into the primary set of assertions. Unfortunately, techniques based on discrimination nets are inept at finding <u>all</u> information referring to a particular item. (The ability to find all information about an item is of importance in natural language generation.) Systems using inverted files as indices into the cache of assertional statements may be viewed as a restricted class of semantic networks in which arc labels are restricted to be numbers (which correspond to positions in

n-tuples).

In comparison, semantic networks, by virtue of their labeled, bidirectional linkage between nodes, provide a cross-indexing as an integral and inherent part of their structure. This cross-linkage is not established through semantically unregulated mechanisms (as in the case with discrimination nets), but those nodes are interlinked (and hence cross-indexed) which are most closely interrelated semantically.

Thus, semantic networks have the potential for combining the expressive power of formal logics with a "natural" indexing scheme and, therefore, appear to be the best choice for the basic representational paradigm of a semantic system. It must be emphasized that while networks potentially have the expressive power of formal logics, numerous network schemes have been devised which do not have this power or which achieve it only through cumbersome mechanisms. Beginning with the next chapter and continuing through the remainder of this work, conventions will be presented for encoding information in networks. Through these conventions, networks may be constructed with both the assertional expressive power of predicate calculus and the procedural expressive power of the Turing machine.

Chapter 15

The Building Blocks of Network Notation

Arguments of the previous chapter indicate that networks provide a reasonable data structure for the encoding of knowledge. However, before attempting to use networks, the formal structure of nets must be stated more precisely.

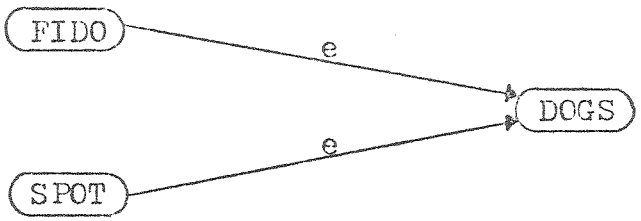
15.1 Nodes and arcs

The basic building blocks of semantic networks are, of course, nodes and arcs. Each node in a network represents an object, an element of U. Such an object might be a piece of matter, a situation, a formal relation, an event, a set or any other member of U. Arcs also encode objects, but are restricted to the encoding of biparticipant, omnichronic situations.

In drawing diagrams which depict networks, a printed symbol will often appear as a label within a circle which represents a node. For example, consider the single node network depicted in Figure 15.01. This network consists of a single node. This node is labeled by the symbol "3". In this paper, no two nodes with the same label will ever appear in the same figure. The (unique) node labeled "<string>" is referred to as node '<string>'. Thus, the

A One Node Network

Figure 15.01



A Network Containing e-Arcs

Figure 15.02

node of Figure 15.01 is node '3'. A node '<string>' is typically used to represent the concept conveyed by <string>. Thus, node '3' represents the number 3. A node 'JOHN' might represent a particular person, JOHN.

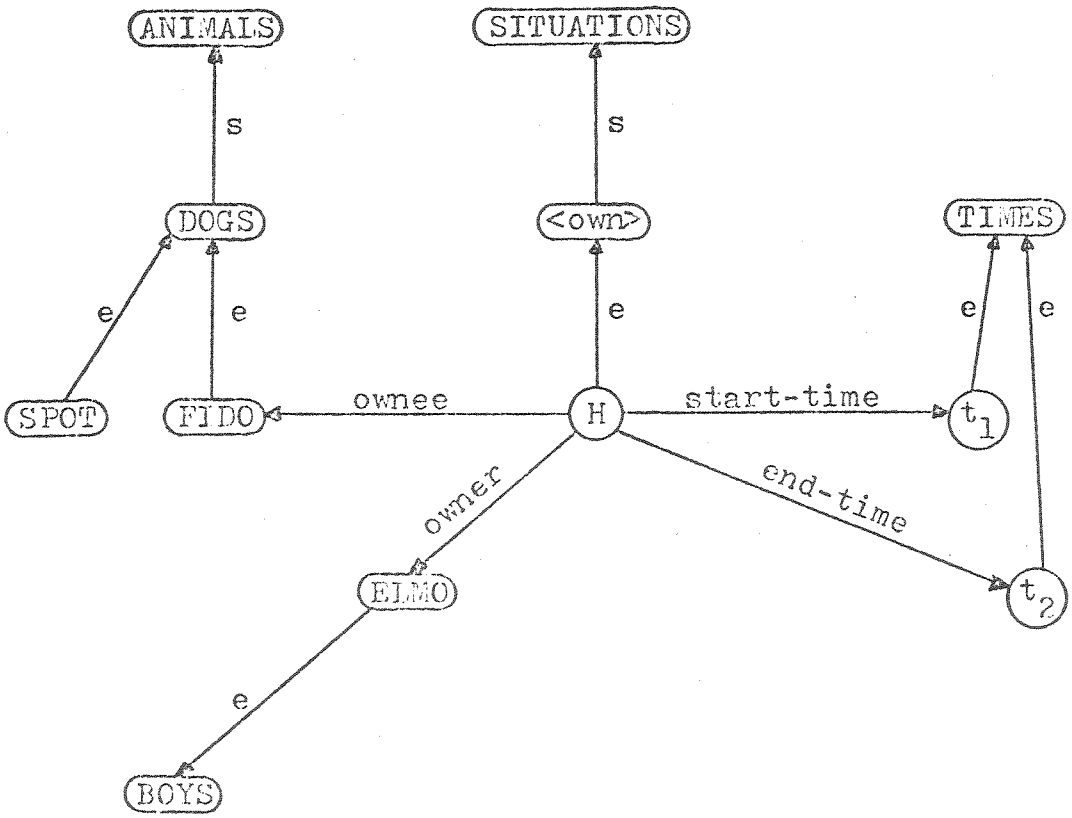
In the diagrams, all arcs will be labeled. Unlike nodes, many arcs with the same label may appear in the same diagram. For example, the network of Figure 15.02 contains two arcs with the label "e". The label of an arc indicates the type of biparticipant, omnichronic situation that the arc encodes. Arcs labeled with "e", e-arcs, represent situations in which one object is an element of a second object that must be a set. Hence, in Figure 15.02, FIDO and SPOT are indicated to be elements of the set DOGS.

Now the reader may be tempted to believe that arcs are used to represent binary relations, but this is not really true. Arcs represent the same information represented by elements of binary relations. The formal relation describing set membership consists of a set of ordered pairs, i.e., a set of n-tuples. FIDO's being a member of the set of DOGS has nothing to do with n-tuples. It is rather a situation in itself which may be captured in a model involving either n-tuples or arcs. To interpret an e-arc as encoding an instance of a relation (an element of a relation set) which in turn encodes a situation existing between an object and a set which contains it is to

introduce an unnecessary level of indirection. It is perfectly acceptable, however, to say that e-arcs model situations which are also modeled by the formal relation between elements and sets. Using the more informal term "relationship," it is also acceptable to say that an arc represents a relationship between two objects.

A feeling for how nodes and arcs are organized to represent various facts may be gained by considering the network of Figure 15.03. In this network the node 'ANIMALS' represents the set ANIMALS, the set of all animals. Likewise, node 'DOGS' represents the set of all dogs. The arc labeled "s" from 'DOGS' to 'ANIMALS' represents the situation of DOGS being a subset of ANIMALS. Similarly, the e-arc from 'FIDO' to 'DOGS' represents the situation of FIDO being an element of the set DOGS. Hence, FIDO is shown to be some particular dog.

Node 'H' encodes an owning situation, an element of the situation set <own>. Since owning situations may be initiated and terminated by exchange events, they are not omnichronic and hence are never represented by arcs (in this system). The particular owning situation H is the owning of the dog FIDO from time t_1 until time t_2 by boy ELMO. H may be thought of as a fairly complex situation which is itself a participant in situations. In particular, there are special situations existing between H and its



Network for "Elmo Owns Fido"

Figure 15.03

participants. For example, the thing owned in the owning situation is in the situation of being the "ownee" of H. This relationship between H and its "ownee" participant is indicated by an ownee-arc from 'H' to 'FIDO'. Notice that the relationship between a complex situation and each of its participants is both omnichronic and biparticipant.

15.2 Reasons for the restrictions on arcs

Since arcs are clearly well suited for modeling the same entities as may be modeled by formal binary relations, the reader may well wonder why arcs have been restricted to the modeling of only biparticipant, omnichronic situations. The rationale behind this restriction is as follows: Both arcs and formal relations attempt to model situations. Since arcs are associated with exactly three pieces of information (a from-node, a to-node and a label) they may adequately model only those situations which may be fully specified by three items. Every situation is associated with a situation type and exists over an interval of time. Further, every situation involves participants. Since an arc label is used to indicate the situation type, only two positions remain for representing the time interval and the situation participants. If arcs are restricted to the representation of omnichronic situations, then the time interval of the situation's existence is always the universal

span of existence and need not be recorded explicitly. This leaves the two positions free to designate situation participants. If non-omnichronic situations are allowed, the two positions must be used to indicate the start and end times of the situation, leaving no provisions for the encoding of situation participants.

In Figure 15.03, the start-time and end-time of owning situation H are clearly indicated in the network. How could these important facets of the existence of H be encoded if the situation were recorded as an own-arc from 'ELMO' to 'FIDO'?

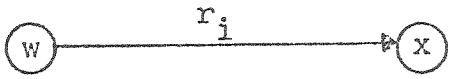
15.3 Using networks to represent algebraic formalisms

Since algebraic objects are both well behaved and well understood, they provide convenient examples for discussing facets of encoding information in networks. The network encoding of the often used mathematical relation "is an element of the set" uses an arc with label "e", an e-arc, as illustrated previously by Figures 15.02 and 15.03. Similarly, s-arcs indicating set membership were used in Figure 15.03.

Among the interesting mathematical objects in U are the n-tuples. Let "N-TUPLES" be the label of a node which corresponds to the set of all n-tuples. Then if w is an n-tuple, a network such as that shown in Figure 15.04 may

Object w is an N-Tuple

Figure 15.04



The i^{th} Component of w is x

Figure 15.05

be constructed. In general, an n-tuple w has a structure $w = (x_1, x_2, \ldots, x_n)$. Each x_i, a component of w, is in a special relationship with w. Let (w, x_i) be an element of R_i iff x_i is the i^{th} component of w. Since R_i is a formal binary relation, a designator corresponding to R_i may appear as an arc label. In particular, let the arc label "r_i" designate instances of the relation R_i. Thus, the fact that x is the i^{th} component of w is shown in Figure 15.05.

Using e-arcs and r_i-arcs, instances of any formal relation may be recorded. For example, consider

(DOOR, ROOM1, ROOM2) $\in$ CONNECTS,

This statement is encoded by the network of Figure 15.06. The e-arc from 'Z' to 'CONNECTS' indicates that Z is an element of CONNECTS. The r_1-, r_2- and r_3-arcs indicate that Z is the triple (DOOR, ROOM1, ROOM2). As emphasized in Chapter 16, this structure does not directly encode the fact that DOOR connects ROOM1 and ROOM2. That is, DOOR's connecting ROOM1 and ROOM2 is not modeled by any single node in the network. Rather, the network structure indicates that the triple (DOOR, ROOM1, ROOM2) is an element of relation set CONNECTS. This condition in turn encodes the fact that DOOR connects ROOM1 and ROOM2.

15.4 Expansion definitions for primitive arc relations

The arcs labeled "e", "s", "r_1", "r_2", etc.,

indicate binary relationships (not relations) among objects. Hence, these arcs may be reencoded through n-tuples and relation sets. For example, Figure 15.07 may be rewritten as Figure 15.08. Figure 15.08 indicates that $(X, Y) \in E$, where E is the formal relation set defining the "is an element of the set" relation. Since E is a primitive network relation, it must be expected that e-arcs will be defined in terms of other e-arcs.

Likewise, r_i-arcs as seen in Figure 15.09 may be rewritten as in the network of Figure 15.10. Figure 15.10 indicates that $(X, Y) \in R_i$, where R_i is the set defining the relation "an i^{th} component of X is Y." Notice that for $i > 2$, r_i is not defined in terms of itself.

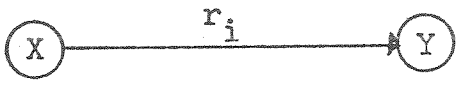
The subset arcs (s-arcs) may also be expanded. For example, Figure 15.11 may be reencoded as Figure 15.12 where S is the subset relation. That is, $(X, Y) \in S$ iff $X \subset Y$. Note that the s-arc is not defined in terms of itself.

15.5 Encoding clusters

Clusters, like their n-tuple brethren, may be encoded using e- and r_i-arcs. For example, the cluster

$$C = (x_1, x_2; y_1, y_2, y_3; z)$$

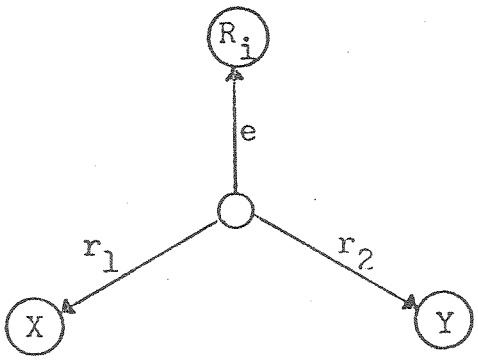
may be encoded as shown in Figure 15.13. A number in parentheses following an "r_i" in an arc label indicates



Encoding of "Y is the i^{th} Component of X"

Using r_i-Arc

Figure 15.09



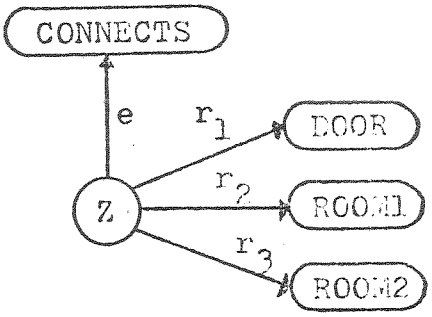
Encoding of "Y is the i^{th} Component of X"

Using Relation R_i

Figure 15.10

The s-Arc Encoding of $X \subset Y$

Figure 15.11

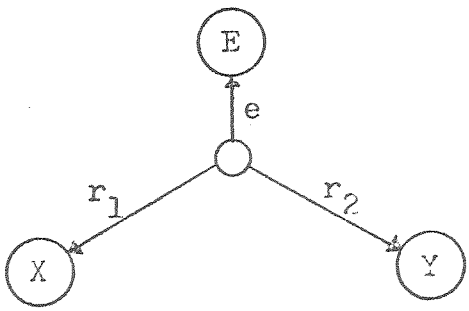


Network Encoding of (DOOR, ROOM1, ROOM2) $\in$ CONNECTS

Figure 15.06

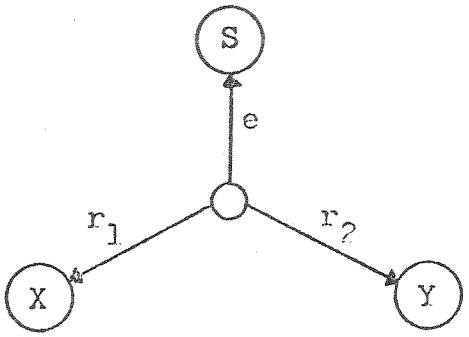
Network Encoding of X $\in$ Y Using e-Arc

Figure 15.07



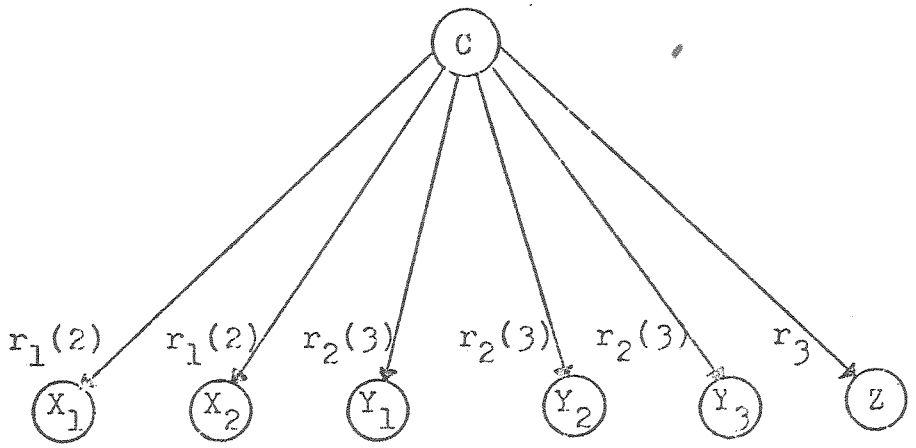
Network Encoding of X $\in$ Y Using E Relation

Figure 15.08



Encoding of $X \subset Y$ Using Relation S

Figure 15.12



Network Encoding of a Cluster

Figure 15.13

that there are multiple r_i components in the cluster. Thus, the $r_2(3)$-arcs all point to the second components of the cluster and indicate that there are three such components.

Chapter 16

Direct Network Encoding

Since networks are to form the fabric of the representation scheme, it is important that they be utilized in the most direct and efficient way possible. Simultaneously, it is important to maintain the representative power and the mathematical legacy of standard logical encoding schemes. While the whole of Division III is concerned with these issues, it is the purpose of this chapter to concentrate on the transition from a set-theoretic to a network-oriented perspective.

16.1 A reevaluation of relation sets and n-tuples

Chapters in Division I placed heavy emphasis on relation sets and n-tuples as media through which knowledge about the world may be encoded. However, it is necessary to bear in mind that these artifacts of modeling are simply the vehicles for encoding. While set-theoretic structures may at some times be of interest in their own right, their principal utility is in encoding information concerning other entities.

In particular, situations and conditions which in some way interrelate a group of objects have, in previous chapters, been encoded through formal relations. Situations and conditions outside the sphere of set theory are

usually neither relation sets nor n-tuples. Set-theoretic constructs may encode these entities, but are not equivalent to them.

For example, the fact that John was at home between times t_1 and t_2 has been encoded by the statement

$$(t_1, t_2, \text{John}, \text{John's-house}) \in \text{AT}^{**}.$$

John's being at home between time t_1 and t_2 is an event E. Now any event is a thing and hence an object. But apparently neither event E nor a model of E appears explicitly in the encoding statement above.

It might be argued that at least the essence of E is encoded by the 4-tuple

$$N = (t_1, t_2, \text{John}, \text{John's-house}).$$

To see the fallacy of this notion, consider

$$N \in \text{OWNS}^{**}$$

which denotes the event F of John's owning his house (the object called "John's-house") from time t_1 to t_2. Clearly, the events expressed by

$$N \in \text{AT}^{**}$$

and

$$N \in \text{OWNS}^{**}$$

have very little to do with one another. Since N is used in the encoding of both events, N clearly does not encode "the essence" of either event. The n-tuple N is simply an artifact of the mathematical formulation of the situations

called "at" and "owns" as captured in formal relations AT** and OWNS**.

Note also that neither the set AT** nor the set AT is a set of "at" events of which E is a member. Both AT** and AT are sets of n-tuples, not sets of "at" events.

In an attempt to find a construct which models the event E above, consider M, the relation statement n-tuple

$$(AT^{**}, t_1, t_2, \text{John}, \text{John's-house}).$$

This statement references both the n-tuple N (implicitly) and the formal relation AT**. Yet M is not the event E; it is a 5-tuple which makes a statement which "describes" E. In general, such a statement might even be false.

While the event E may be described by an n-tuple such as M, it should be clear that E itself is not an n-tuple. In general, many objects have some type of internal structure which may be described in terms of n-tuples (or clusters), but this does not make the objects themselves n-tuples.

The 5-tuple M in an appropriate context may be assigned any value. For purposes of encoding information in a computer, it may be very desirable to assign E as the value of M. Let q% be the context in which a statement

$$(x_0, x_1, x_2, \ldots, x_n)$$

has as its value the situation or condition described (or encoded or modeled) by the statement

$$(x_1, x_2, \ldots, x_n) \in x_0.$$

Then $\underline{value}_{q\%}(M) = E$. Several natural language systems (e.g., Winograd's) and robot modeling systems (e.g., STRIPS) have used this convention (implicitly).

Another way of interpreting the equation

$$\underline{value}_{q\%}(M) = E$$

is to say that the meaning of M in context q% is E. That is, in context q%, M is intended to signify the event E.

In considering computer constructs for modeling various aspects of knowledge, it seems clear that the values assigned to various construct building blocks should not be mathematical artifacts (e.g., n-tuples), but should be the actual objects such mathematical formalisms are intended to encode. That is, computer data structures should not model formalisms such as n-tuples and relation sets which in turn model objects of interest. Rather, data structures should model the objects of interest directly. Formal mathematical constructs may, however, be used to formalize the relationships between data structures and their associated modeled objects.

16.2 Direct network encoding of non-mathematical entities: Part 1

Section 15.3 presented network methods for recording the fact that

(DOOR, ROOM1, ROOM2) $\in$ CONNECTS.

That is, constructs were outlined for representing the fact that the triple (DOOR, ROOM1, ROOM2) is an element of the relation set CONNECTS. While this fact may be of some importance, pragmatically the fact that DOOR connects ROOM1 to ROOM2 may be of more immediate concern.

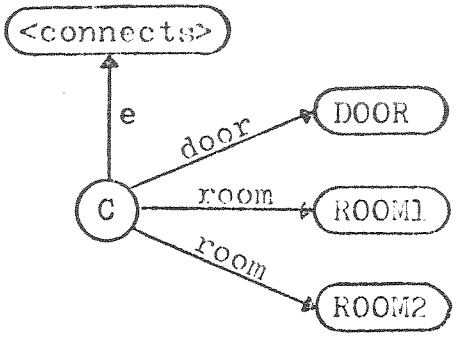
DOOR's connecting ROOM1 to ROOM2 is a condition (or situation or state of being) C which is easily modeled by the algebraic constructs (DOOR, ROOM1, ROOM2) and CONNECTS. Yet condition C is not itself a standard algebraic construct. It is neither an n-tuple nor a relation set nor a relationship between an n-tuple and a relation set. Rather, C is a particular static situation, a particular specimen from a general category (Chapter 11) of objects which may be characterized by the general statement "a door x connects rooms y and z."

C may be modeled by the network of Figure 16.01. Here C is encoded as an element of the category <connects>, the set of all instances of some door x connecting two rooms y and z. Notice that

<connects> $\neq$ CONNECTS

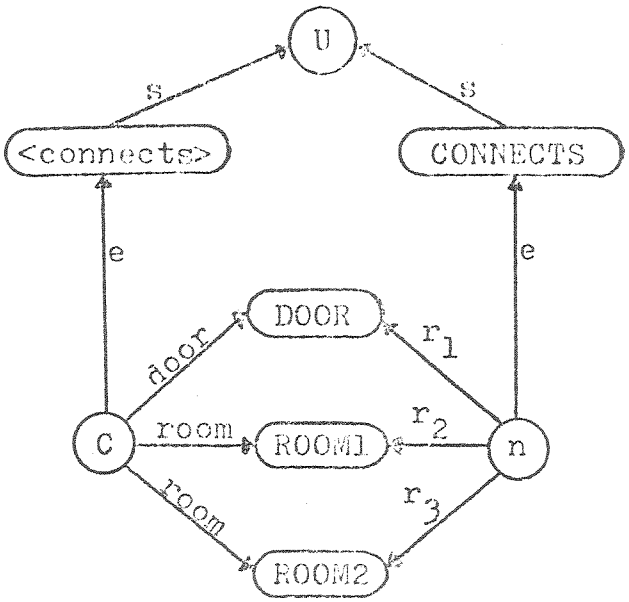
where CONNECTS is the set of all triples (x, y, z) such that x connects y and z.

Unlike an n-tuple, C has no precisely defined internal mathematical structure. However, by their nature,



Direct Network Encoding of
"DOOR connects rooms ROOM1 and ROOM2"

Figure 16.01



The Sets <connects> and CONNECTS

Figure 16.02

the elements of <connects>, being members of a category, possess certain attributes or deep semantic cases. It is through these attributes that the DOOR and rooms ROOM1 and ROOM2 are associated with C.

The attributes characterizing members of <connects> are @door and @rooms. The value of @door, #@door, is always an element of the set of doors while #@rooms is a set or rooms with cardinality two. The #@door of a member of <connects> connects its #@rooms.

Referring again to Figure 16.01, a door-arc exists from 'C' to 'DOOR'. Formally, this means that

$$(C, DOOR) \in DOOR.OF.A.CONNECTS$$

where DOOR.OF.A.CONNECTS is a mathematical relation between connects situations and doors. More directly, the door-arc from 'C' to 'DOOR' indicates that DOOR is the value of C's @door attribute.

Rather than have rooms-arcs leading from 'C' to a set of two rooms, Figure 16.01 uses two room-arcs to link 'C' to nodes 'ROOM1' and 'ROOM2'.

In general, the value of an attribute @a will be pointed to in a network by an 'a' arc.

To emphasize the relationships between <connects> and CONNECTS and between C and (DOOR, ROOM1, ROOM2), consider Figure 16.02. Here the node 'U' represents the universal set U. The s-arcs indicate that <connects> and

CONNECTS are subsets of U. However, <connects> is a set of situations in which a door connects two rooms while CONNECTS is a set of triples (which may be used in encoding connections). The e-arc from 'C' to '<connects>' indicates that C is an example of a door's connecting two rooms. Namely, C is the situation of DOOR connecting ROOM1 and ROOM2. The e-arc from 'n' to CONNECTS indicates that n = (DOOR, ROOM1, ROOM2) is an element of CONNECTS.

16.3 The arcs as attribute designators

In the preceding section, the arcs leaving node 'C' (Figures 16.01 and 16.02) were associated with certain attributes possessed by C. It seems reasonable to assume that any arc may be associated with some attribute possessed by the entity modeled by the arc's from-node.

The 'r_i' arcs presented in Section 15.03 were not described as modeling attributes. However, it is clear that the various i^{th} components are each characteristically associated with an n-tuple. Thus, any n-tuple has the attributes of i^{th} component for $1 \leq i \leq n$. The i^{th} component attribute is $@r_i$.

16.4 Direct network encoding of non-mathematical entities: Part II

The example of network encoding using CONNECTS presented in Section 16.2 was taken from a robot-world

application (STRIPS). For a more linguistic second example of direct network encoding of situations, reconsider the events E and F of Section 16.1. Recall that E is John's being at John's-house from time t_1 to time t_2 and F is John's owning John's-house from time t_1 to time t_2.

The n-tuple

$$N = (t_1, t_2, \text{John}, \text{John's-house})$$

may be used in the set-theoretic encoding of events E and F through

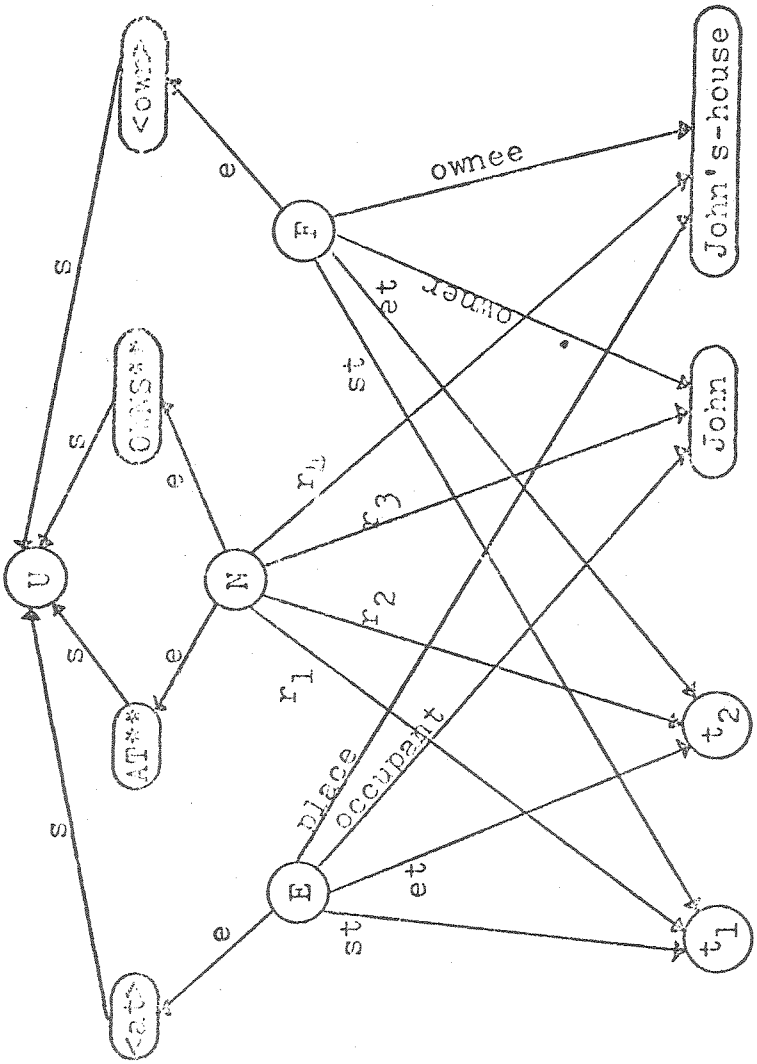
$$N \in AT^{**}$$

and

$$N \in OWNS^{**}$$

respectively.

Consider the network of Figure 16.03. The top node in this network is 'U', the designation of the universal set. Subsets of U (as denoted by s-arcs) include <at>, AT**, OWNS** and <own>. Set <at> is the set of events characterized by some one or some thing occupying a particular place. AT** and OWNS** are sets of 4-tuples. The set <own> is the set of events characterized by some one or some thing owning another object. Note that <at> intersects neither AT** nor OWNS** nor <own>. Likewise, <own> intersects none of the other sets. However, AT** and OWNS** are both sets of 4-tuples and hence their intersection is at least feasible. Indeed, AT** and OWNS** both



The Sets <at>, AT**, <own> and OWNS**

Figure 16.03

contain the 4-tuple N.

E, an "at" event and hence an element of <at>, has certain characteristic attributes @st, @et, @occupant and @place. The @st and @et attributes of any object indicate the time at which the object comes into existence (start time) and the time at which the object ceases to exist (end time), respectively. In the case of events such as E and F, the #@st and #@et are the times at which the events start and stop. (Note: U, <at>, AT**, OWNS**, <own> and N also have the attributes @st and @et. For convenience's sake, however, they are not shown in Figure 16.03.) The @occupant attribute of an "at" event indicates an object which occupies the #@place of the "at" event from the event's #@st to its #@et.

F, an "owns" event and hence an element of <own>, has the attributes @st and @et as explained above. F also has a #@owner and a #@ownee. From F's #@st to its #@et, F's #@owner owns F's #@ownee.

Arcs leaving node 'N' indicate that it represents the n-tuple

$$(t_1, t_2, \text{John}, \text{John's-house}).$$

Further, N is an element of both AT** and OWNS**. Left unindicated are N's @st and @et attributes which, if present, would record N's omnichronic existence.

Chapter 17

The Network Encoding of General Statements

A central theme of Division II was that a conversational artificial intelligence should use general statements, processes, rules and categories to organize knowledge into convenient, meaningful bundles. Since networks have been selected as the basic encoding structures, there now arises the problem of how to snare these bundles in the net. This chapter focuses upon the encoding of general statements while the encoding for categories and rules will be presented in Chapter 18 and the encoding of process in Chapter 20.

In Chapter 9, reoccurring patterns of situations were collectively encoded by general statements employing the special relation sets $\overline{\forall}$, $\overline{\exists}$ and RP. It is now desirable to develop network representations of general statements. Once developed, these general network statements may be used to economically record sets of related situations. By slight alterations, general statements may be converted to rules for use in connection with the category system. Further, a network representation of general statements may be used to model general statements made in English (such as "all cars have wheels").

In searching for a network encoding of general statements, it will be helpful to have some particular statement in mind. Let this particular statement be

"Every house has an owner."

Like most statements in English, this statement is open to multiple interpretations. The one which will be used in what follows is

> "For the particular time T (the current instant), for every house h existing at T there exists a person p such that p owns h at T."

Using this example statement, the search for a network encoding may begin.

17.1 Network encoding of the set theoretic encoding

One possibility for encoding the sample statement in the network is to first formulate it in terms of the templates and relation sets discussed in Chapter 9 and then encode these structures in the net.

The predicate calculus formulation of the general statement is

$$\forall h \in HOUSES_T,\ \exists p \in PERSONS,$$
$$(OWNS^*,\ T,\ p,\ h)$$

where $HOUSES_T$ is the set of houses existing at T, and PERSONS is the set of all persons. (That the selected p

exists at time T is implicit in the ownership relation.)

Converting the predicate claculus statement to an n-tuple statement involving sets $\bar{\forall}$ and $\bar{\exists}$ yields

$$(\bar{\forall},\ h,\ HOUSES_T,\ \emptyset,\ [\equiv\bar{\exists}\ \equiv p\ \equiv PERSONS\ *\ \equiv[\equiv OWNS*\ \equiv T\ p\ h]]).$$

This formulation of the general statement involves a template (the $[\equiv\bar{\exists}\ \ldots]$) and a template within the template (the $[\equiv OWNS*\ \ldots]$). Both templates are in shorthand notation. In terms of pure n-tuples, the statement is

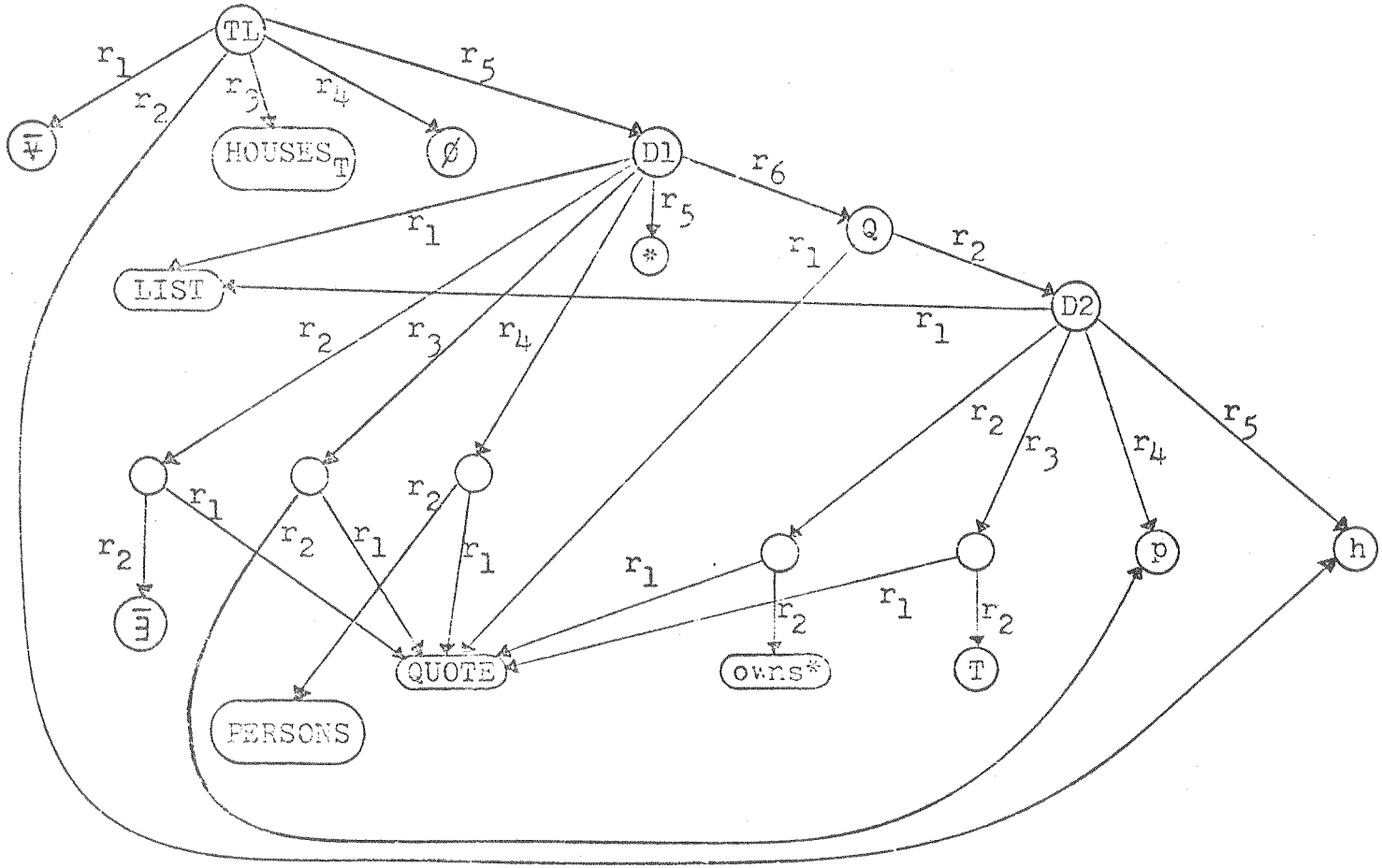
```
(∀̄, h, HOUSES_T, ∅,
    (LIST (QUOTE ∃̄) (QUOTE p) (QUOTE PERSONS)
     * (QUOTE (LIST (QUOTE OWNS*) (QUOTE T)
                    p h))))
```

which may be encoded by the network of Figure 17.01.

In Figure 17.01, the top node, 'TL', represents the top level n-tuple of the general statement. Node 'D1' encodes the $\bar{\exists}$ template while 'D2' encodes the innermost template using OWNS*. Using the notions developed in Division I, it is possible to interpret this statement, but the complexity and inelegance of the structure is overpowering. And at best it is an encoding of an encoding. A better and more direct formulation is needed.

17.2 The direct network encoding of a simple general statement

Clearly, it is important to preserve in network



Network Encoding of N-Tuple Encoding of Sample Statement

Figure 17.01

formalisms the encoding power of n-tuple statements, but this encoding power need not be achieved by merely encoding n-tuples in nets. Indeed, the awkwardness of the network developed in the last section arose from attempting to mimic an n-tuple formulation. Taking full advantage of facilities provided by networks, the example statement may be encoded in the much more concise network of Figure 17.03.

Before pondering how the sample general statement

$$\forall h \in HOUSES_T, \exists p \in PERSONS, \text{ p owns h at T}$$

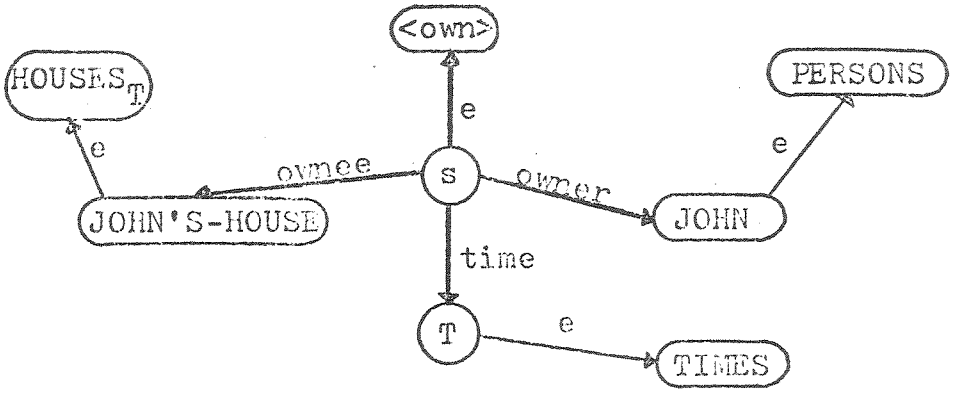
may be efficiently encoded by a net, consider the network encoding of the specific instance of this general statement

"JOHN owns JOHN'S-HOUSE at time T"

which is depicted in Figure 17.02. The owning of JOHN'S-HOUSE by JOHN at time T is a situation s which is encoded by node 's'. Situation s is an element of <own>, the category of situations in which a #@owner owns a #@ownee at some time #@time. For the example situation, JOHN'S-HOUSE is the #@ownee, JOHN the #@owner and T the #@time.

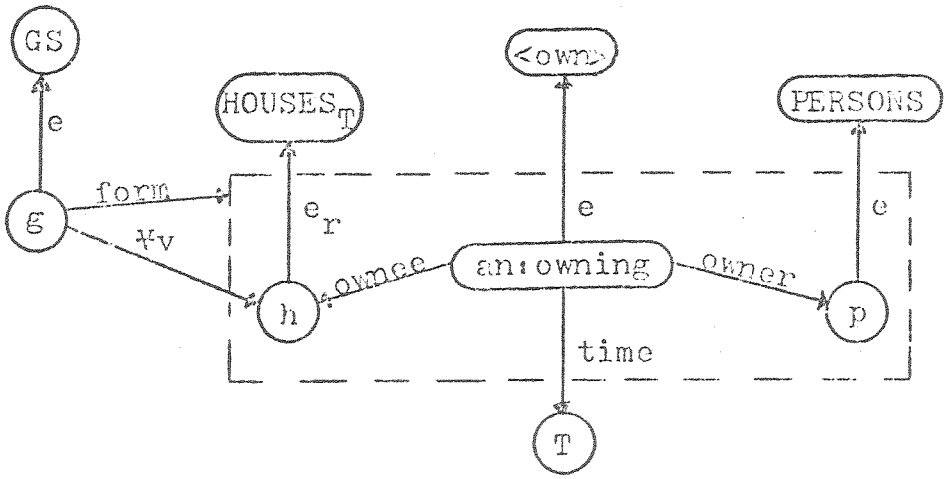
Note that the general statement is intended to encode the fact that instances of <own> similar to the one just cited exist for each house in $HOUSES_T$.

The network of Figure 17.03 encodes the sample general statement. The node 'GS' represents (models) the set of all general statements while node 'g' represents the particular statement under discussion. Like any complex



JOHN Owns JOHN'S-HOUSE at Time T

Figure 17.02



Every House in HOUSES$_T$ Has an Owner at Time T

Figure 17.03

object, g has an "internal" structure. This structure is indicated by the arcs leaving 'g'.

The most important attribute of any general statement is its #@form. In Figure 17.03, the form-arc is drawn from node 'g' to an area of the network enclosed by a broken line. This area (and others like it) may be interpreted, used and implemented in a variety of ways. It is helpful to think of this area as a supernode which has a subnetwork as its label. (The defining of special areas in a network is the subject of Chapter 19.) The #@form of g is used to indicate the type of reoccurring situations which g records and is the network analog of the template

[≡∃̄ ≡p ≡PERSONS * ≡[≡OWNS* ≡T p h]]

used in the n-tuple encoding of the general statement.

The #@form is not itself a situation or condition which reoccurs. Rather, it is a structured variable (just as templates are structured variables whose values (in appropriate contexts) reflect the internal n-tuple structures of the templates themselves). It is the values which the #@form may assume which are the actual situations and conditions of the reoccurring pattern.

The @∀v attribute of g is involved with quantification. The ∀v-arc links 'g' to node 'h'. Node 'h' represents a quantified variable whose range is given by its #@e_r, the arc label "e_r" meaning "element of the range."

The ∀v link from 'g' to 'h' indicates that h is a "for-every variable." That is, the situations represented by the general statement reoccur for every assignment of h to a value in its #@e_r (in its range).

The #@∀v and #@form of g work in combination to indicate that for every assignment of the #@∀v (h) to a member of its #@e_r (its range, here $HOUSES_T$) there exists an assignment of the #@form which is consistent with the #@form's internal structure. (Note that the #@∀v of g is included in the structure of the #@form of g.)

Less cryptically, for every assignment of h to a value in $HOUSES_T$, there exist objects and relationships between objects which parallel the structure of the #@form (the structure within the broken lines). That is, there exist objects x_h, $x_{an:owning}$ and x_p paralleling h, an:owning and p. Further, there exist situations (other objects) paralleling those encoded by the arcs of the #@form. Namely, the following situations exist:

$x_{an:owning}$ is an element of <own>.

x_p is an element of PERSONS.

x_h is an element of $HOUSES_T$.

(e_r-arcs are a special type of e-arc.)

x_h is the #@ownee of $x_{an:owning}$.

T is the #@time of $x_{an:owning}$.

x_p is the #@owner of $x_{an:owning}$.

In short, each reoccurrence of the situation encoded by the general statement might be recorded by a subnet which is isomorphic to the #@form. An inaccurate, but intuitively appealing statement of the interaction of #@¥v and #@form is that for each assignment of the #@¥v of g, the #@form is copied. Note that if the #@form is "copied" for h assigned to JOHN'S-HOUSE, the network of Figure 17.02 is produced (with the #@owner indicated to be JOHN).

It is difficult to overemphasize the practical importance of having the #@form portion of a general statement "look like" the representation of actual instantiations of the statement. After all, information must be retrieved from networks by structure matching. To answer a question, a network system might first look for a specific record containing the answer. Failing that, the system must look for more general statements which cover the specific situation of interest. These general statements may be found more easily if their #@forms parallel the structures of the questions which they answer.

17.3 Some fine points on the network encoding of general statements

In an effort to introduce the network encoding of general statements without bogging down in details or tangential considerations, several important points were

omitted which must now be considered.

17.3.1 On #@form conventions

The #@form of general statements are represented pictorially by areas surrounded by broken lines. Clearly, all nodes falling within this boundary belong to the #@form and are "copied" for each reoccurrence of the statement. Further, all arcs which lie completely within the boundary are "copied." But arcs often cross the boundary. Such arcs may or may not belong to the #@form. By convention, if an arc's label is written within the boundary, it is to be interpreted as belonging to the #@form (and is "copied"). Otherwise the arc belongs to the higher level (partition space).

17.3.2 The e_r-arc

Notice in Figure 17.03 that node 'h' is connected to '$HOUSES_T$' by an e_r-arc. Note also that 1) other nodes in the #@form have e-arcs to indicate from what sets their values must be taken and 2) when the #@form is "copied" (as in Figure 17.02), the e_r-arc is replaced by an e-arc. So why couldn't 'h' simply have an e-arc also to specify its range?

The answer to this question may be found by considering the general statement

"Every bird is an animal which can fly"

or

$\forall b \in$ BIRDS, b $\in$ ANIMALS and b can fly.

This statement is encoded (at a coarse level of detail) in the network of Figure 17.04. Notice that 'b' has both an outgoing e_r-arc and an outgoing e-arc. If e_r-arcs were replaced by e-arcs, the network of Figure 17.04 might be interpreted to mean either that

Every bird is an animal which can fly

or

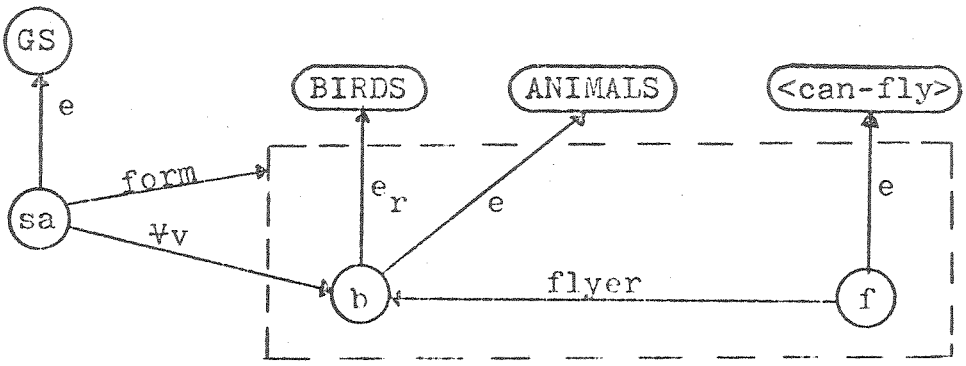
Every animal is a bird which can fly.

If for some application no node ever appears within a form with more than one e-type arc (either e or e_r), then e-arcs may be substituted for e_r-arcs. This condition turns out to be fairly easy to achieve if network designers plan ahead. By establishing an s-arc from 'BIRDS' to 'ANIMALS' it follows that birds are necessarily animals, eliminating the need for the e-arc of Figure 17.04.

17.3.3 Existential quantification

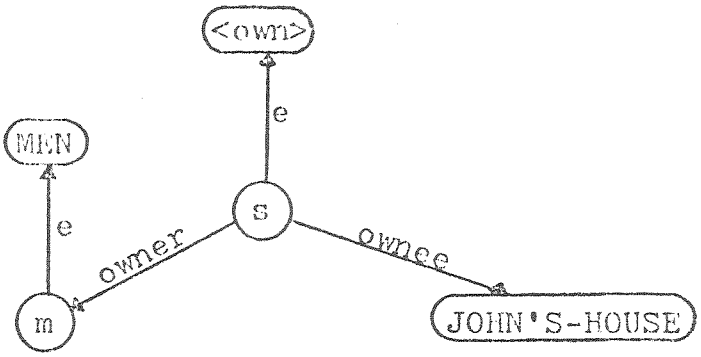
The $\forall$v-arc of Figure 17.03 is used to express a universal or "for-every" quantification. Of course, other quantifiers also exist and provisions must be made to encode these in the net.

Perhaps even more important than the universal quantifiers is the existential. So important is the



Every Bird is an Animal Which Can Fly

Figure 17.04



There Exists a Man Who Owns JOHN'S-HOUSE

Figure 17.05

"there-exists" quantification that it need appear only implicitly in the network. For example, consider the existentially quantified statement

There exists a man who owns JOHN'S-HOUSE
(at time T).

This fact is encoded by the network of Figure 17.05. The existence of the man is asserted by the very appearance of the node 'm' in the network.

Other nodes and arcs in the network imply existence also. Thus, the node 'JOHN'S-HOUSE' implies the existence of JOHN'S-HOUSE, 's' implies the existence of an owning situation, the ownee-arc implies the existence of a relationship between an owning situation and an object, etc.

(Lest the reader be concerned that such implicit existence will prevent the encoding of facts relevant to the mating habits of unicorns or the singing abilities of the sirens, let him be reassured that existence in alternative or hypothetical worlds will be discussed subsequently.)

A more interesting usage of the existential quantifier is provided by those instances in which it appears within the scope of a universal. (It is this quantification structure which makes necessary the Skolem functions of resolution theorem proving.) The sample general statement of Section 17.1 exemplifies this usage:

$\underline{\forall}h \in HOUSES_T$,

$\underline{\exists}p \in$ PERSONS and a situation in which

p owns h.

Again, existence is handled implicitly. From Figure 17.03, for each assignment of variable h to a value in $HOUSES_T$, a "copy" of all objects recorded in the #@form of statement g implicitly exists. In other words, g may be interpreted as meaning

"for every h in $HOUSES_T$, there exist objects (including relationships) matching the structure of #@form."

An interesting side issue arises here. Suppose there were no houses at time T. (Let T = 10^9 B. C.) Now the general statement is still true, there just aren't any h's in $HOUSES_T$ to imply the existence of ownership situations and owners. But there are still nodes in #@form implying the existence of something. But these nodes in the #@form of g are <u>variables</u>. The variables exist even though there are no values to assign them. Even when assignments do exist, the nodes in the #@form must not be confused with actual situations. This point is a subpart of a much broader issue, the isolation problem, which will be discussed in Section 19.2.

17.3.4 <u>Other quantifiers</u>

Natural languages provide a spectrum of quantifications between the universal and existential extremes. (Negative quantifiers are also provided. "No vampire eats garlic.") The "¥v" quantification corresponds to the English quantifiers "all," "every," "each," "both" and, in some usages, to "any." Other types of quantification may be encoded in networks by replacing ¥v-arcs with arcs denoting other quantification types.

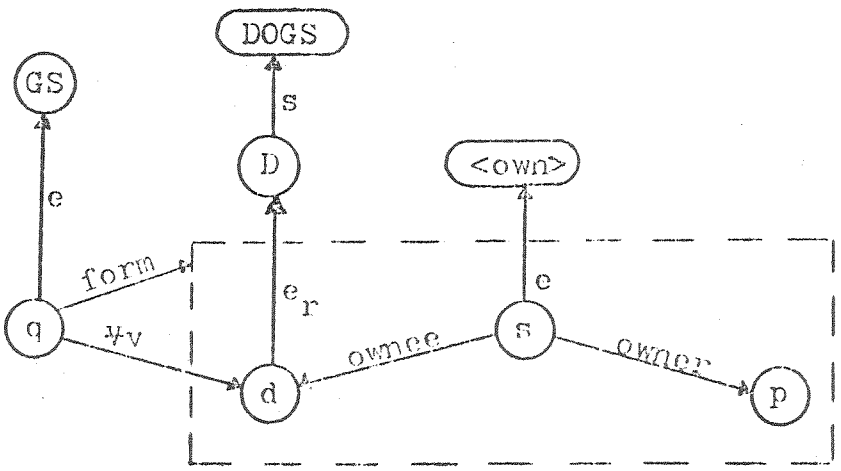
A quantification type which appears frequently in natural language is "some" quantification. This quantification is expressed in English by such words as "some," "few," "many" and "most." Each of these might be represented by a separate arc type. For example, most-v-arcs might be defined.

Rather than represent "some" quantification by special arcs, subsetting may be used. Consider the encoding of the statement

"Some dogs have owners"

which is encoded in the network of Figure 17.06. Really, the network of Figure 17.06 states that every d an element of D has an owner (p). But D is a subset of DOGS. That is, D is a set of <u>some</u> of the dogs from the set of all dogs. All the dogs in this subset have owners.

In one respect, the "some" quantifier is more powerful than "for-every," since "some" implies that the range



Some Dogs Have Owners

Figure 17.06

set is not $\emptyset$. To catch this fact in the network, information concerning set D may be included to show that it is non-empty. For "many" and "few" quantifications, information must also be included to indicate the relative size of D in comparison to its superset.

17.3.5 Nested quantification: $\forall x\ \forall y$

More complex statements sometimes require nesting one universal quantification inside the scope of another. For example, consider the statement

"There is a line connecting any two points"

which may be restated as

$\forall x \in$ POINTS, $\forall y \in$ POINTS, $\exists L \in$ LINES,

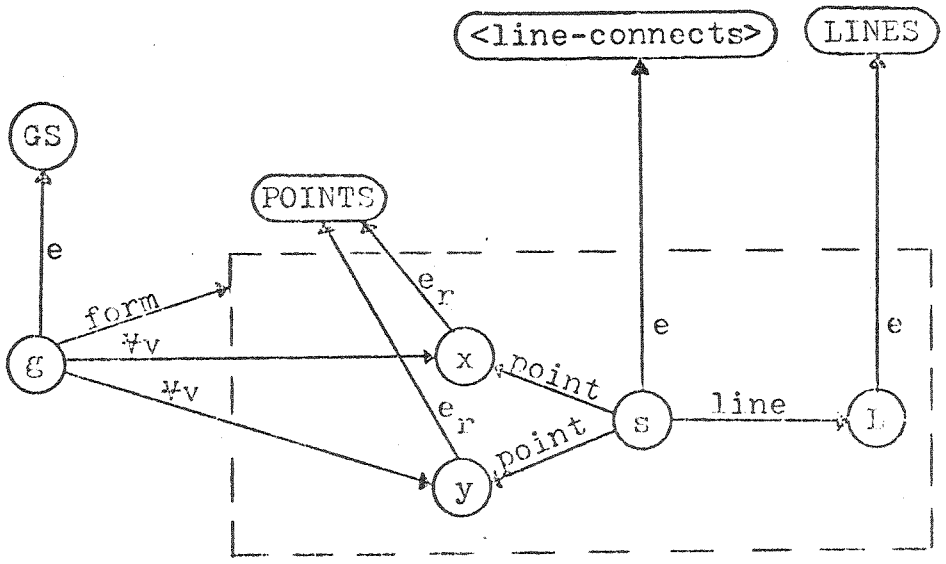
L connects x and y.

As is well known, universal quantifiers are commutative. Hence, the statement may also be given as

$\forall y \in$ POINTS, $\forall x \in$ POINTS, $\exists L \in$ LINES,

L connects x and y.

Following the example of Figure 16.02 in which identical arcs link the two rooms connected by a common door, $\forall$v-arcs from the same general statement node may point to two universally quantified variables as in Figure 17.07. This usage of multiple $\forall$v-arcs provides the network analog of general cluster statements involving the set RP of Section 9.2.3.3.



Every Two Points are Connected by a Line

Figure 17.07

17.3.6 Nested quantification: ∀∃ ∀∃

Some statements become so complex that one #@form must be nested within another. Consider the abstract statement

$$\forall a \in A,\ \exists b \in B,\ \forall c \in C,\ \exists d \in D$$
$$[\text{predicate } (a, b, c, d)].$$

This statement may be encoded by the double form network of Figure 17.08.

A more interesting network, Figure 17.09, is used to encode

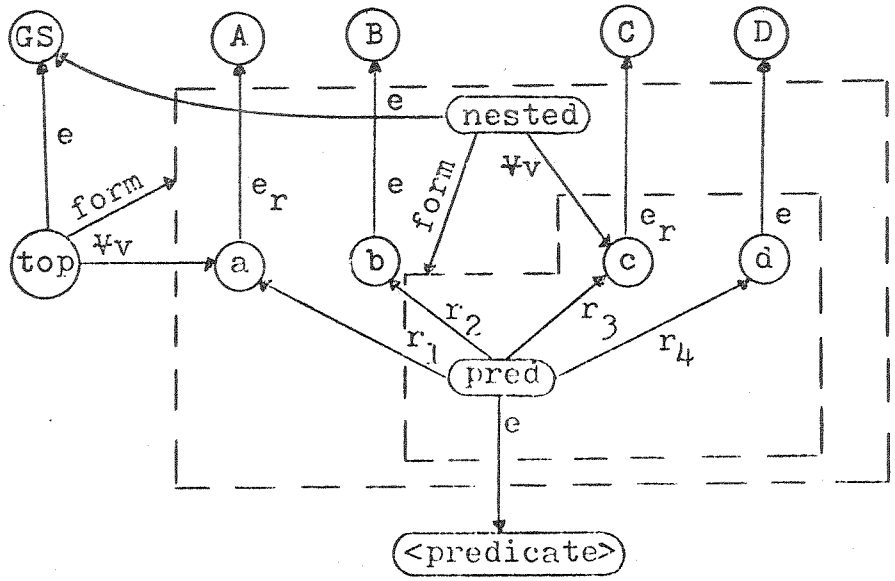
$$\forall a \in A,\ \exists b \in B,\ \forall c \in b,\ \exists d \in D$$
$$[\text{predicate } (a, c, d)].$$

As statements become more and more complex, it may be necessary to nest forms to greater depths. While it may be hoped that complex nesting will seldom be used, it is reassuring to know that it is available and may be pressed into service when needed to encode even the most complex quantifications.

17.3.7 Form recycling

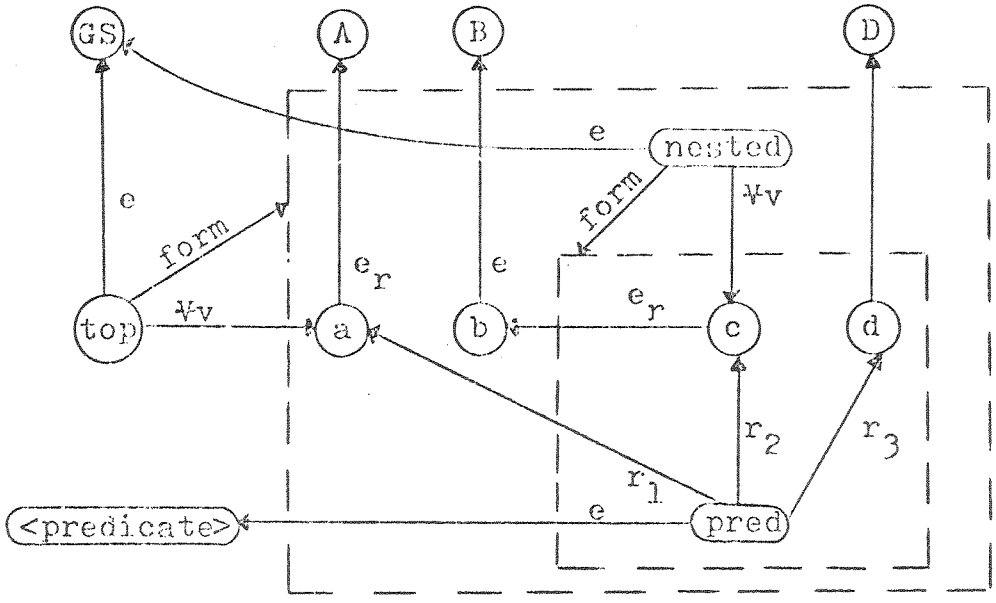
In network terms, the most costly component of a general statement is the #@form. When multiple statements use the same #@form, network efficiency is improved.

Two statements which may be recorded using the same #@form are



$\forall a \in A, \exists b \in B, \forall c \in C, \exists d \in D\ [\text{predicate}\ (a, b, c, d)]$

Figure 17.08



$\forall a \in A, \exists b \in B, \forall c \in b, \exists d \in D\ [\text{predicate}\ (a, c, d)]$

Figure 17.09

S1: Every man loves a woman.

and

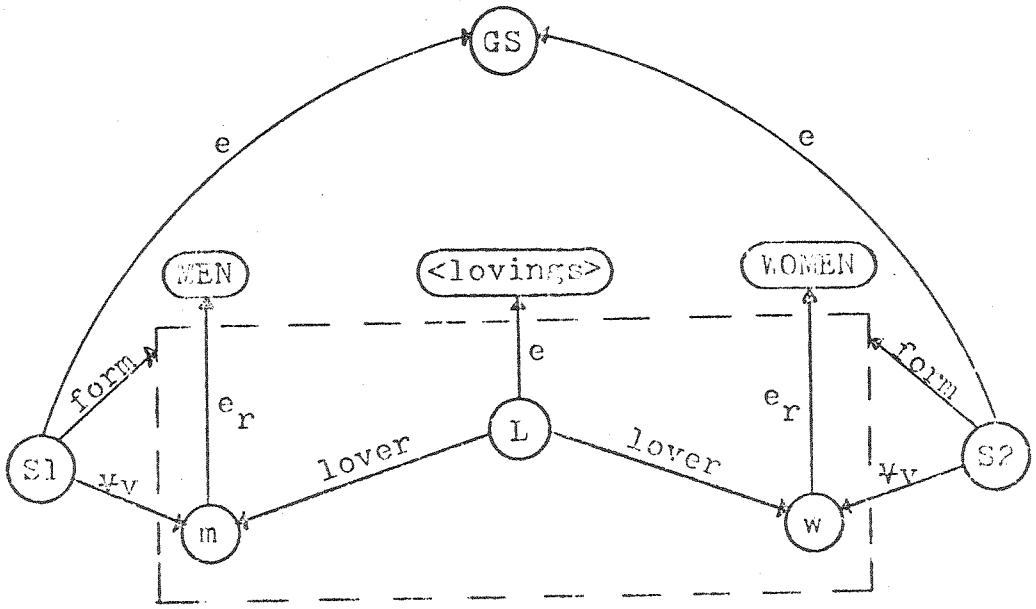
S2: Every woman loves a man.

(As usual, these English statements are ambiguous. Assume all men don't love the same woman and all women don't love the same man.) These statements are recorded in the network of Figure 17.10. Note that in considering S1, the e_r-arc from 'w' to 'WOMEN' must be interpreted as an e-arc. Conversely, for S2, the e_r-arc from 'm' to 'MEN' must be interpreted as an e-arc. As described in Section 17.3.2, an e_r-arc may generally be treated as an e-arc except when quantified range ambiguities arise.

17.4 Cross-references

The encodings of many types of complex general statements require the use of constructs which have not yet been presented. A general statement involving a category definition is presented in Section 18.3.3 while constructs for encoding disjunction, implication and negation are discussed in Section 19.6.

The problems of encoding quantification in semantic networks, as discussed in Woods (1975), has plagued workers in this area for some time. Woods was able to cite only three network schemes which are logically adequate for encoding quantification. (There are many which are logically



Every Man Loves a Woman - Every Woman Loves a Man

Figure 17.10

inadequate.) The first of these, Shapiro's (1971), is discussed in Section 19.3. By far the best known of the logically adequate mechanisms, this scheme is a more or less direct network encoding of the predicate calculus.

The second scheme is that of Martin Kay (1973) which involves Skolem functions. (Schubert, 1974, describes a related approach.) Essentially, the idea is for every existentially quantified entity to include special pointers to all the universally quantified entities upon which the existential depends. This technique will, of course, lead to a proliferation of pointers. (Such pointers are implicitly carried by the #@form construct since all nodes and arcs within a form are known to depend upon the associated ∀vs.) Further, negation is a well-known nightmare in systems using Skolem functions.

The third scheme has been proposed by Woods himself, but apparently has not been implemented. Described in terms of lambda abstractions, the scheme is essentially the same as that developed in Section 9.2 using special sets $\overline{\forall}$ and $\overline{\exists}$ and templates. Network encodings in this scheme resemble both Shapiro's and optimizations of the structures of Section 17.1. So, while the philosophy underlying the scheme differs from Shapiro's, the resultant networks appear to be little more than notational variants on Shapiro's.

Chapter 18

The Network Encoding of Rules and Categories

Continuing the theme of capturing organizational structures in networks, this chapter focuses upon the network encoding of categories and rules.

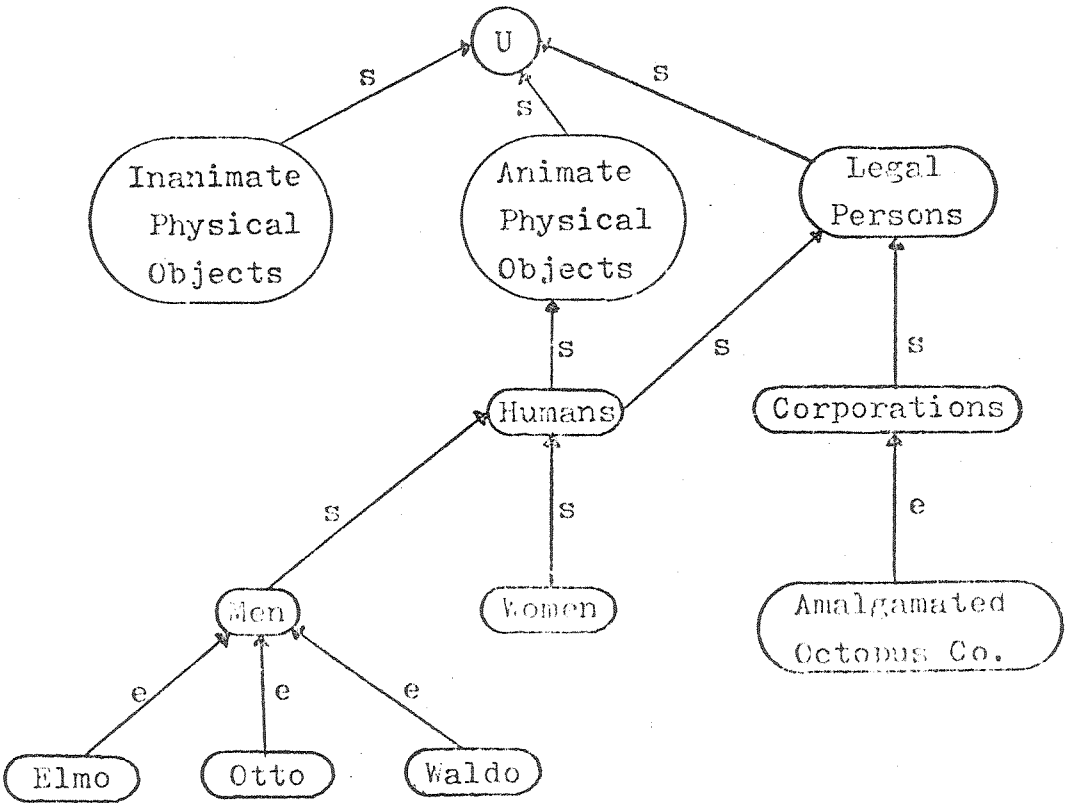
18.1 The skeleton of hierarchical classification in nets

The bare bones of hierarchical classification are trivially implemented in nets. Each category may be modeled by a single node with the hierarchical relations between categories and subcategories represented by s-arcs. A simple hierarchy is encoded in the network of Figure 18.01 which is hopefully self-explanatory.

18.2 The network encoding of rules

The more interesting problems of classification involve putting meat on the bare bones discussed above, by associating rules with the categories. Of course, for a network based system, rules must first be encoded in network structures.

As noted in Section 11.1, rules are closely akin to general statements, differing principally in the quantification and assignment of variables. Hence, it is reasonable to expect the network encoding of rules to follow that



Network Representation of a
Hierarchical Classification

Figure 18.01

of general statements.

In their most general form, rules may be defined over multiple variables. For example, two objects #x and #y represented by variables x and y may be said to obey rule h iff

#x owns #y at time T.

In terms of n-tuples,

$h = (d, x, y)$

where

$d = [\exists OWNS^* \quad \exists T \quad x \quad y]$.

Further, JOHN and JOHN'S-HOUSE obey rule h iff

$(JOHN, JOHN'S\text{-}HOUSE) \in H^h$

and, hence, iff

(OWNS*, T, JOHN, JOHN'S-HOUSE)

and iff

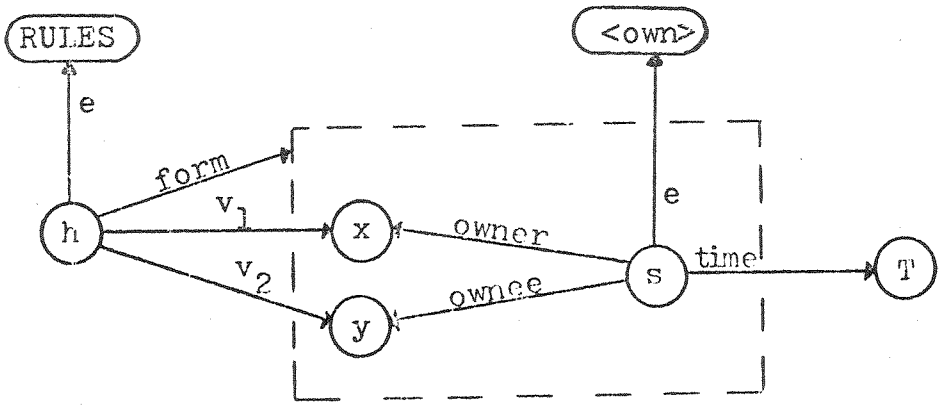
(T, JOHN, JOHN'S-HOUSE) $\in$ OWNS*.

A network encoding of rule h is depicted in Figure 18.02. As expected, this network bears a strong resemblance to that of Figure 18.03 (a reproduction of Figure 17.03) which encodes the related general statement

$\forall q \in HOUSES_T$, $\exists p \in PERSONS$,

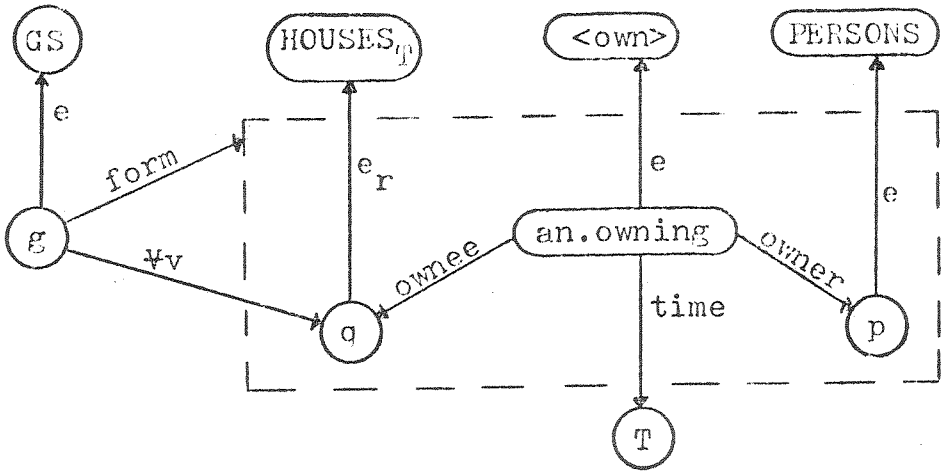
p owns q at time T.

Both g and h use a #@form. Statement g uses a #@∀v for quantifications. The range of the universally quantified statement variable q is $HOUSES_T$, the value of q's $@e_r$



The Rule "x owns y at T" in Net Form

Figure 18.02



Every House in $HOUSES_T$ is Owned at T by a Person

Figure 18.03

attribute. Rule h employs two unquantified variables which must be distinguished by separate arcs labeled "v_1" and "v_2". ("JOHN owns JOHN'S-HOUSE" is not the same as "JOHN'S-HOUSE owns JOHN".) Rules involving more variables have additional @v_i attributes.

The #@forms of both g and h are related to existence. Existence for statements has been discussed elsewhere (Section 17.3.3). The #@form of h denotes existence in that two objects #x and #y obey h iff there exist situations relating #x and #y in the same way that variables x and y are related in the #@form of h. That is, #x and #y obey h iff an object #s and the following relations exist.

#s is an element of <own>.

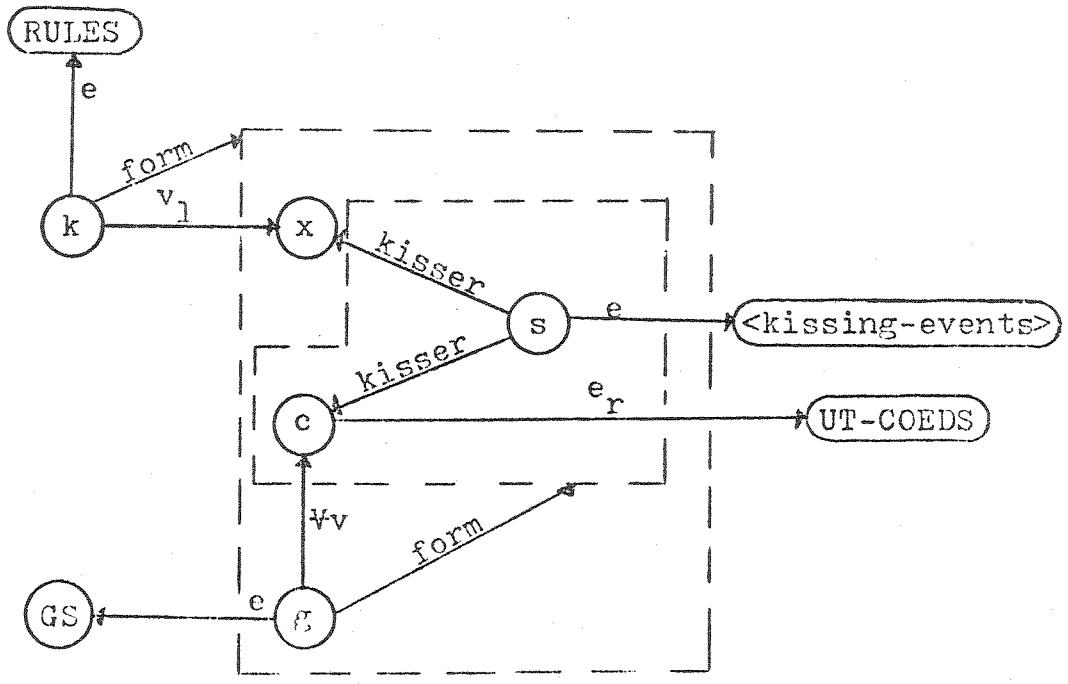
#x is the #@owner of #s.

#y is the #@ownee of #s.

T is the #@time of #s.

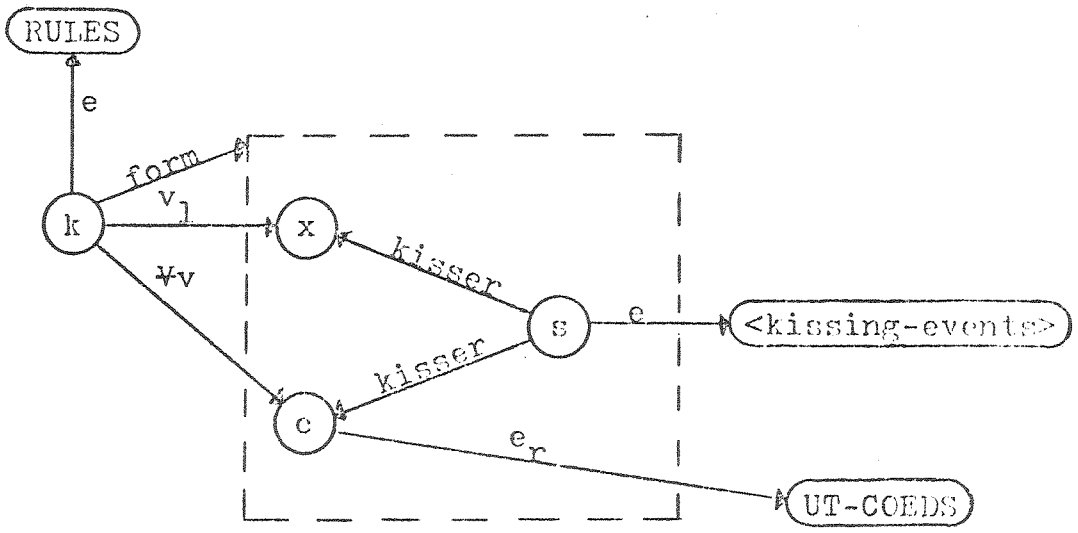
It is important to emphasize that unlike many statements, rule h does not in itself imply the existence of anything. That is, h does not imply the existence of any #x, #s, etc. However, if objects #x and #y are known to obey h, then h implies the existence of an #s and of the four relations cited above.

Most of the fine points for general statements discussed in Chapter 17 apply equally to rules. However, two points are worth mentioning before closing this section.



The University of Texas Playboy Rule: Version One

Figure 18.04



The University of Texas Playboy Rule: Version Two

Figure 18.05

as shown in Figure 18.05. "Assignment of values" proceeds by first "assigning" the v_i, then the ¥v and finally the unmarked existentials within the #@form.

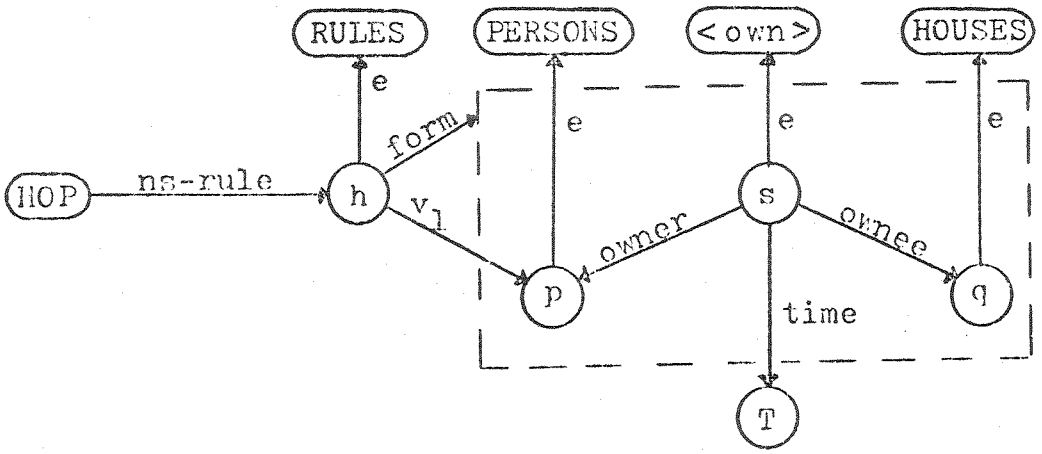
18.3 Linking rules and categories

In Chapter 11, it was shown that the very concept of category is inextricably bound to the notion of the rule. Categories are defined by their ns-rules and realize their utility by compressing information into rules of several types. Clearly then, it is important for the network data base to provide easy linkage from a category to its various rules.

18.3.1 A straightforward link

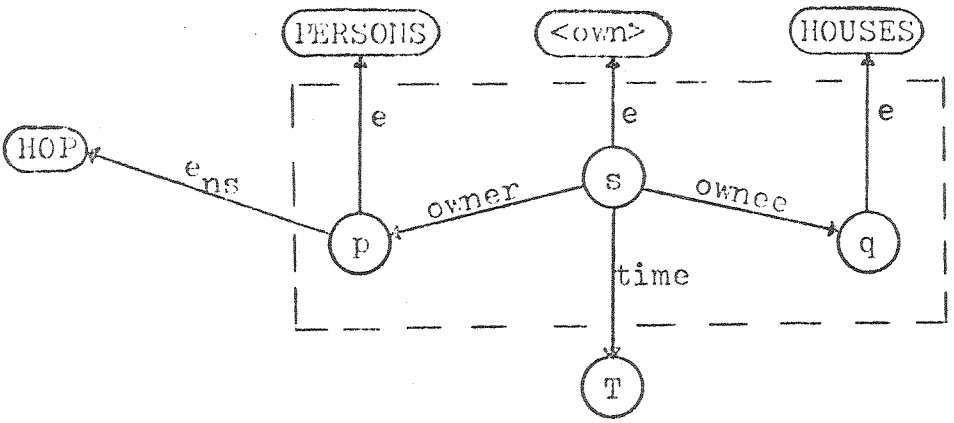
From a theoretical vantage, rules may be thought of as attributes which are possessed by categories. The chief attributes possessed by categories are the attributes @ns-rule, @n-rule, @s-rule and @d-rule which link a category to its ns-, n-, s- and d-type rules respectively. In terms of networks, these attribute links may be implemented by arcs labeled "ns-rule", "n-rule", etc.

As an example of the link between a category and its ns-rule, consider the category HOP of persons who own houses at time T. (HOP = House Owning Persons.) HOP and its ns-rule (= its #@ns-rule) are encoded in the network of Figure 18.06. (Arcs connecting HOP with the hierarchical



The Category of House Owning Persons
and its Rule: Version One

Figure 18.06



The Category of House Owning Persons
and its Rule: Version Two

Figure 18.07

structure are omitted.) Category HOP is represented by node 'HOP' with its ns-rule being represented by node 'h'. An ns-rule-arc links 'HOP' to 'h'. From discussions in previous sections, it is hopefully clear that rule h (in conjunction with the ns-rule-arc from 'HOP') indicates that an object #p obeys h and hence belongs to HOP iff there exist objects #s, #q and the relations (objects):

#p is an element of PERSONS.

#s is an element of <own>.

#q is an element of HOUSES.

#p is the #@owner of #s.

T is the #@time of #s.

#q is the #@ownee of #s.

Rules of other types may, of course, be linked with categories in much the same way.

18.3.2 Shortening the linkage

Category rules, as opposed to rules in general, are somewhat restricted in that they are always one-place rules. That is, they have exactly one ("external") variable which (when the rule holds) assumes a value from the category membership. Taking advantage of this fact, the network representation of a rule may be simplified as illustrated in Figure 18.07.

In this network, category node 'HOP' is linked to

rule variable node 'p' by an e_{ns}-arc. Being an arc, the e_{ns}-arc encodes a relationship which exists between HOP and p. Namely

(HOP, p) $\in$ ENS

where ENS is a mathematical relation set. In general, (x, y) $\in$ ENS iff

there exists h $\in$ RULES such that

h is a one-place rule,

h is an ns-rule of x and

y is the #@v_1 of h.

It is assumed that y will be the #@v_1 of at most one rule h. This means that y determines h uniquely. Thus the e_{ns}-arc implicitly carries all the information concerning h which was encoded explicitly in the network of Figure 18.06.

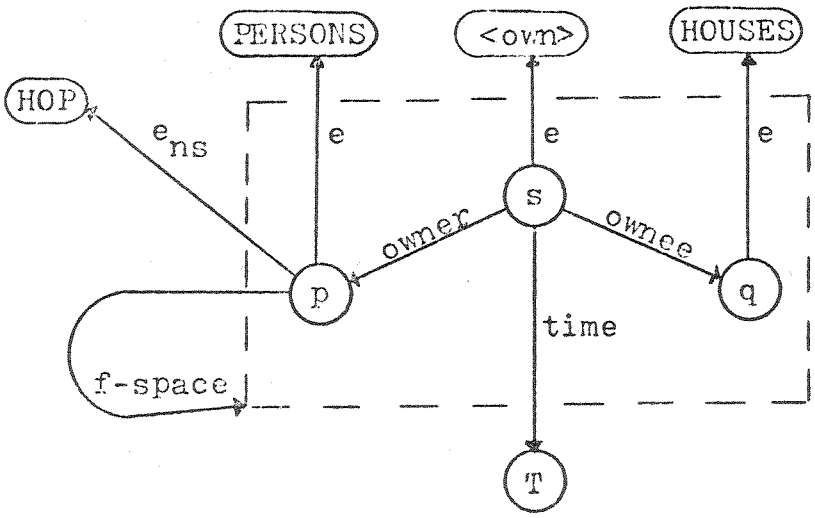
The use of e_{ns}-arcs (and corresponding e_n-, e_s- and e_d-arcs) has two advantages over the explicit encoding scheme. First, networks using e_{ns}-arcs are more economical in that they eliminate one node and two arcs from the net structure. (This advantage is not so great if several categories use the same rule.) Further, this abbreviated structure makes the rule variable only one link away from the category rather than two, producing some efficiency in processing time.

A second advantage of e_{ns}-arcs is that they may be

thought of as special e-arcs. In many applications, it is convenient to think of the rule variable as an archetypal element of the associated category. For example, suppose the validity of some predicate P concerning x is sought and it is known that x belongs to a category C. If P is valid for the "archetypal element" of C, then P must also be valid for x.

There are also disadvantages to the shorthand scheme. Since rules are not explicitly encoded, some computer process must be able to interpret the special e-type arcs. (But some process must interpret rules also.) Further, there must exist some mechanism for finding the #@form of a rule given only its variable. For certain implementation methods (see Section 19.9), the space in which a node lies is available as an integral part of a node's definition in the system. However, to make the information explicitly available, an f-space-arc may be drawn from a rule variable to the #@form of its rule as in Figure 18.08. Note that the label of this arc lies outside the broken line and hence does not belong to the #@form and is never "copied."

The shorthand scheme also makes it more difficult to discuss rules. When rules are explicitly encoded in a network, the label of the encoding node serves as a reference to the rule. This reference is lost when the



An Explicit #@f-space

Figure 18.08

shorthand is used. To overcome this difficultly, let "$RULE_v$" denote the rule using variable v. Thus, for example, it may be said that the #@f-space of p is the #@form of $RULE_p$.

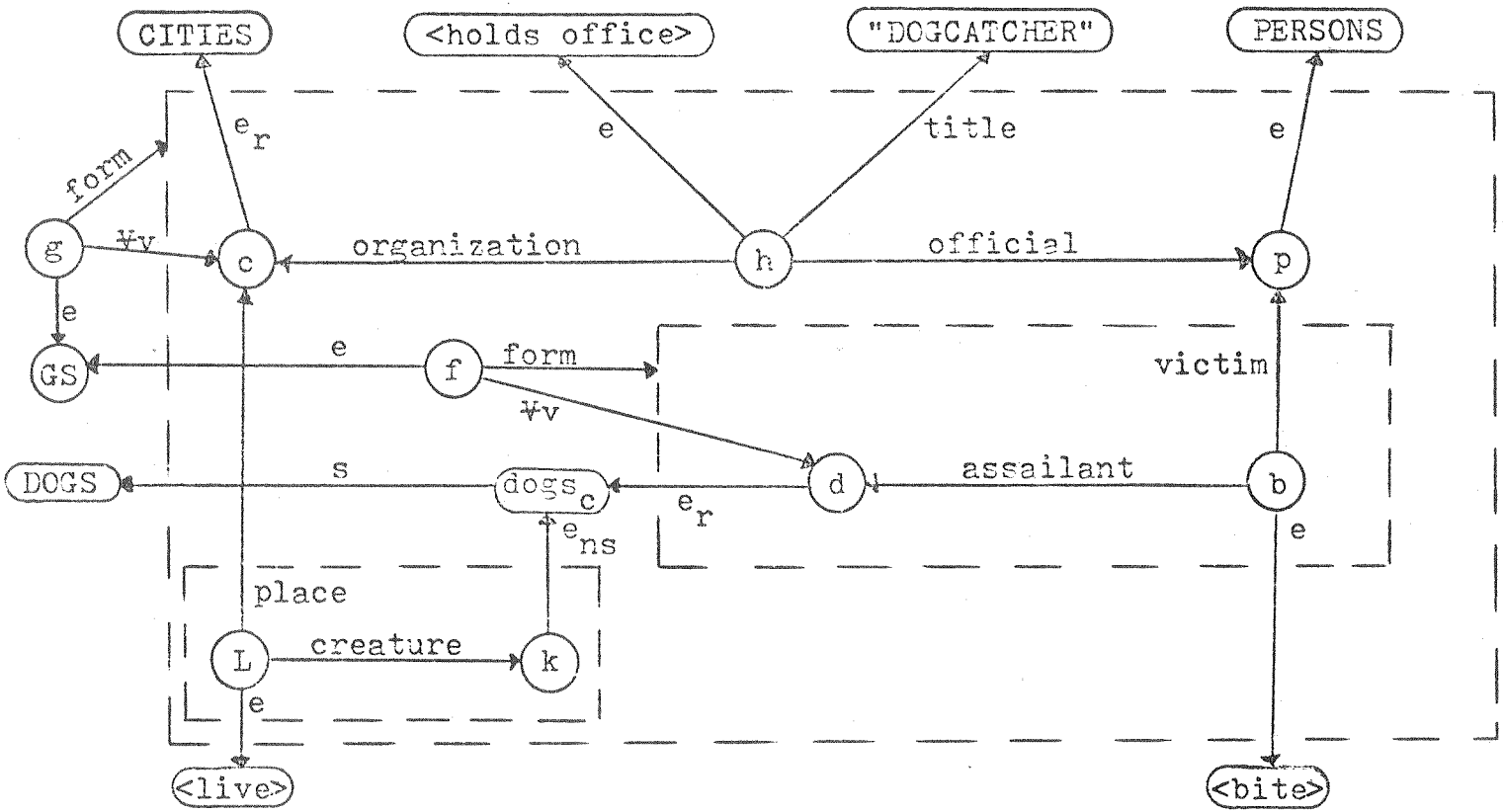
18.3.3 A general statement involving a category definition

The network encoding of category definitions is sometimes needed to record reoccurring patterns of situations in general statements. For example, the network of Figure 18.09 uses the encoding of a category definition in the representation of

"Every city has a dogcatcher who has been bitten by every dog in town."

A restatement of this sentence which makes explicit the various internal variables and quantifications is

For every city c there is a person p and
a (possibly empty) set $dogs_c$ of dogs
such that
p holds the office of dogcatcher in c,
$dogs_c$ is the set of all dogs which
live in c, and
for every dog d in $dogs_c$ (if any),
there exists a biting event b in
which d is the assailant of victim
p.



Every City Has a Dogcatcher Who Has Been Bitten by Every Dog in Town

Figure 18.09

As represented in the figure, the total statement corresponds to node 'g'. The #@¥v of g is c, a variable which ranges over the set CITIES. For every city c there exist entities corresponding to nodes 'p', 'h', '$dogs_c$' and 'f'. Node 'p' represents a person determined by the selection of city c while node 'h' encodes the restricting situation that p must be the office holder with title "DOGCATCHER" for city c.

Since '$dogs_c$' lies within the scope of c, the selection of a city c determines a subset of DOGS. The exact definition of this category of dogs is indicated by the necessary and sufficient rule linked to '$dogs_c$' through the e_{ns}-arc. This rule states that $dogs_c$ contains exactly those elements of set DOGS which live in city c.

Each city c also determines a reoccurring set of situations (or a general statement) f. The interpretation of f is that for every d in $dogs_c$, there exists a biting event b whose #@assailant is d and whose #@victim is p.

18.4 Spotlight on delineation rules

Of the various rule types, delineation rules are (in current usage) the most valuable aids to language processing. Particularly during the parsing phase when surface structures are being translated into nets and when the semantic well formedness of sentences and sentence

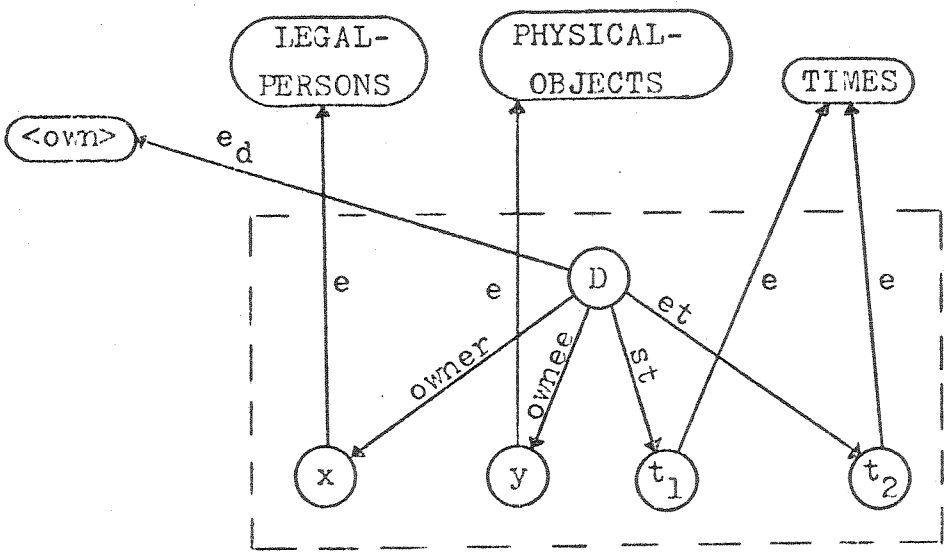
fragments is being tested, it is important to know what attributes are associated with certain categories of objects (especially with event and other verb-like categories) and what range of values each attribute may assume. This information is of utility because attributes indicate the types of participants which are involved in particular categories of situations and because there is often a direct mapping from syntactic cases (including prepositional phrases) to the attributes. Knowing the correspondences between surface cases and attributes and knowing the ranges of values for each attribute allows some parses to be rejected on macro-semantic grounds and provides a facility for predicting the citing of certain situation participants in the surface language. (This prediction ability is especially important for speech understanding.)

The attribute-range information for a category, collectively referred to as the category's delineation, may be associated with the category through a delineation rule (or d-rule). A delineation rule is an n-rule which (necessarily, but sometimes only implicitly) includes range information about every attribute of the delineated category. (As noted in Section II.3, knowing a category delineation is roughly equivalent to knowing that a mathematical relation is taken from the cross product of certain specified sets.) It should be understood that the inclusion of range

information is not a requirement of n-rules in general.

As an example of a delineation rule, consider the delineation of category <own>. Modeled previously by the relation set OWNS**, <own> is the set of situations (or events) in which something is owned over a time interval $[t_1, t_2)$. Each member of <own> has four attributes: a #@owner, a #@ownee, a #@st (start time) and a #@et (end time). (Note: The attribute @time is also associated with <own> and other events. The #@time of an event is any element of the interval [#@st, #@et).) The #@owner of an <own> owns the #@ownee over the time interval [#@st, #@et). To delineate <own> is to specify each of these attributes and its possible range of values. The #@owner of an <own> is always taken from the set LEGAL-PERSONS (the set of humans, corporations and organizations). For present purposes, assume that the #@ownee must be a physical object, an element of PHYSICAL-OBJECTS. Both the #@st and #@et must be TIMES.

A delineation of <own> is encoded by the network of Figure 18.10. In this network, '<own>' is connected to a node 'D' by a rule shorthand e_d-arc. Thus, D is the #@v_1 (the variable) of <own>'s delineation rule, $RULE_D$. Encoded within the #@f-space of D is the necessary information concerning each of the four attributes possessed by members of <own>. (Note: The #@f-space of D (the #@form of $RULE_D$) is



Delineation of Owning Situations: Version One

Figure 18.10

connected to D implicitly and hence no explicit f-space-arc has been included in the figure.)

Category <own> is also delineated by the network of Figure 18.11. However, in this net the delineation is distributed between two rules. Rule $RULE_D$ (the rule with variable D) is local to category <own> and encodes information concerning the attributes @owner and @ownee which are peculiar to situations of owning.

The attributes @st and @et are (theoretically) possessed by all objects. Hence, they may be specified at the top level of the taxonomy and inherited by lower levels. This top level specification is achieved by the delineation rule $RULE_u$ of category U. Note that $RULE_u$ encodes not only simple attribute and range information, but also the restriction that #@st must be "not later than" the #@et. (Objects which exist for a single instant have equal #@et and #@st.) This inclusion of an additional restriction within a delineation illustrates that while the n-rules realizing d-rules must include all pertinent attribute-range information, they are not constrained against encoding additional information.

If needed, extra information may be ignored by parse algorithms. To find the attributes associated with a category C, the system may backtrack a (the) e_d-arc entering 'C' to reach a "delineating element" 'd'. Outgoing

First, note that nodes 'x' and 'y' of Figure 18.02 do not have the e-type arcs which are associated with their counterparts 'q' and 'p' of Figure 18.03. This is because statement g constrains q to be in $HOUSES_T$ and p to be in PERSONS while h specifies no such constraints. In rule h, the very fact that x and y are the #@owner and #@ownee of s implies something about the possible values which x and y may assume. This constraint is encoded by a delineation rule associated with the category <own> (see Section 18.4).

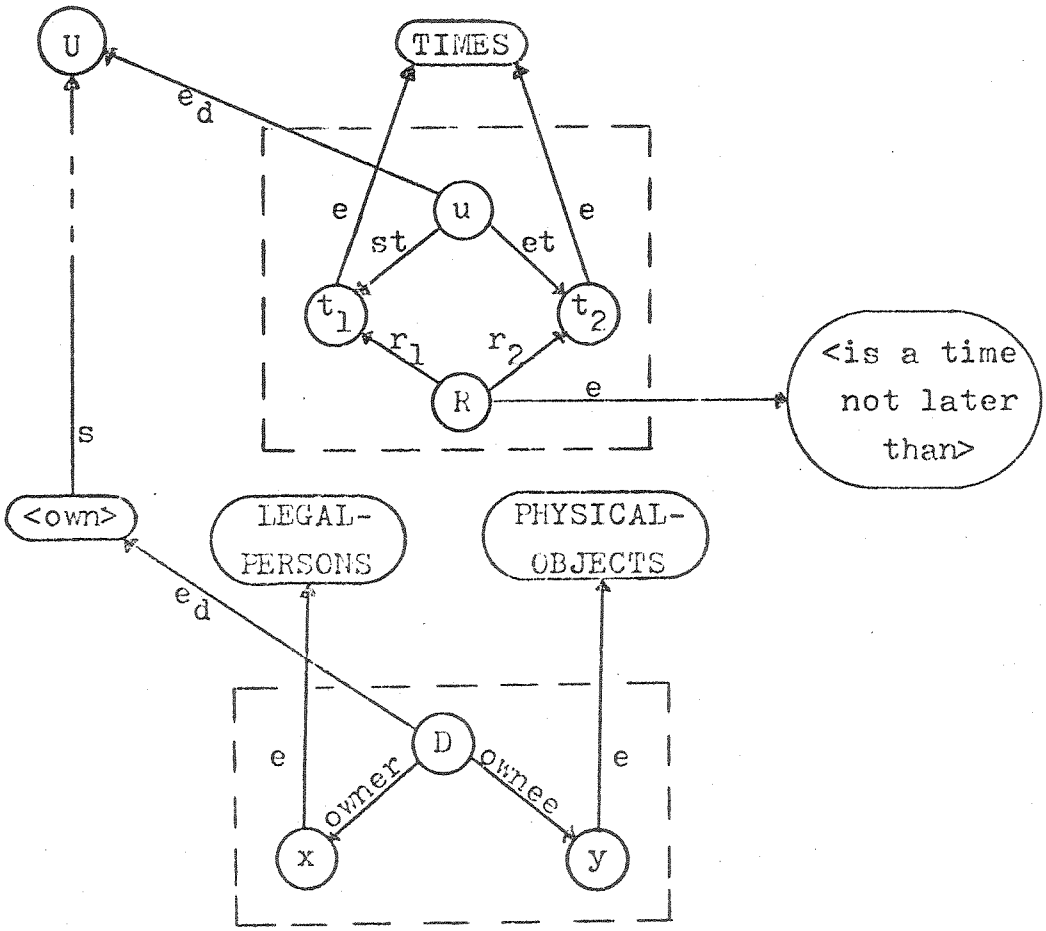
Second, more complex rules may use internal variables with internal quantification. Really, the s used by rule h above is an internal variable with existential quantification. A more enlightening example of an internal variable is provided by the rule k

"x has kissed every University of Texas coed"

which is used to define the category of Texas playboys.

Rule k is encoded in the network of Figure 18.04. Note how this structure embeds a general statement within the #@form of the rule. In general, any complexity of quantification may be encoded by this means. For k, a value of x is first established in the outer #@form, then (in the inner form) kissing events ranging over all UT coeds are considered.

For efficiency purposes, the top level internal universal quantification may be associated directly with k



Delineation of Owning Situations: Version Two

Figure 18.11

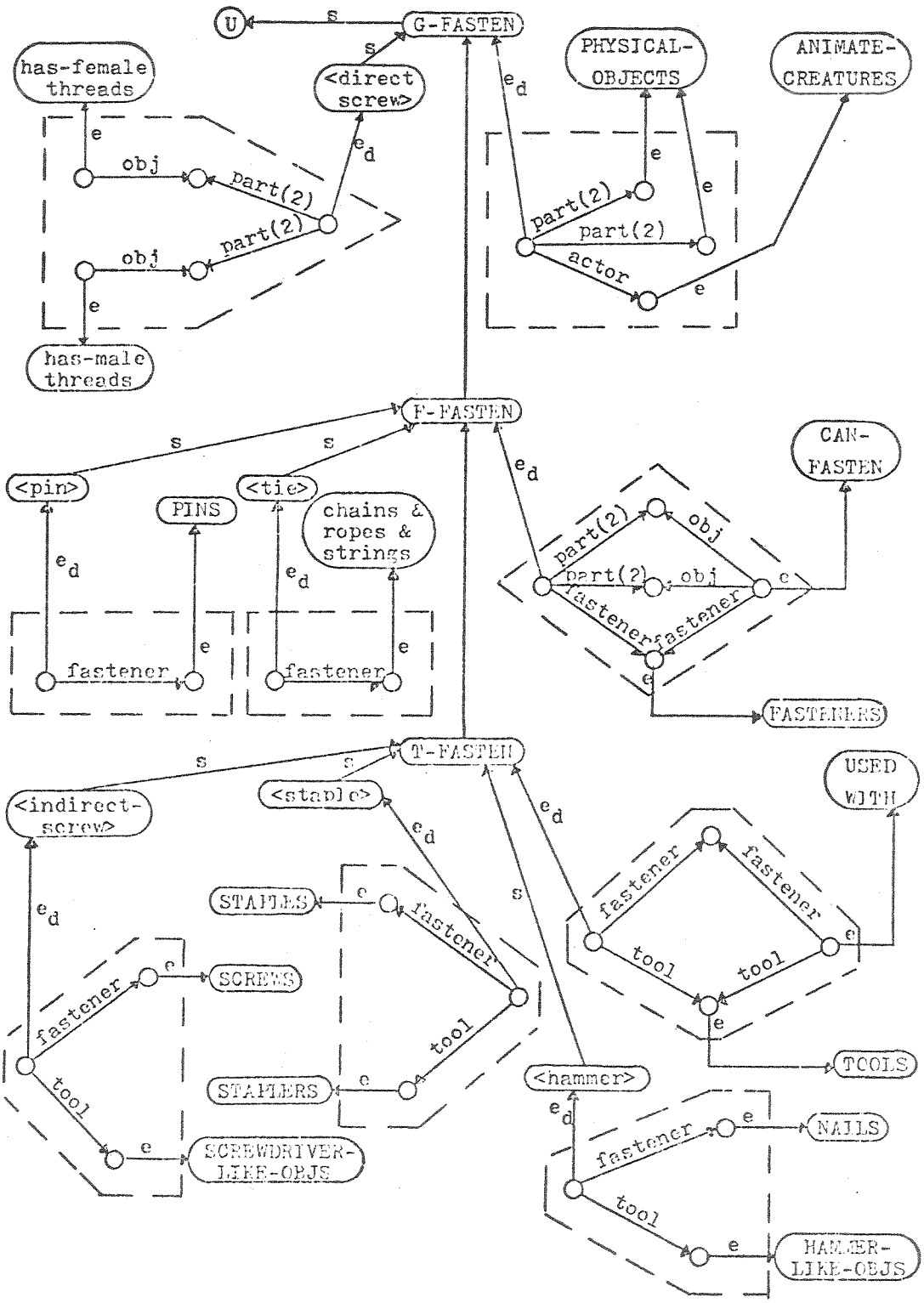
arcs from 'd' to nodes 'x' indicate the associated attributes. Attribute ranges are specified by outgoing e-arcs from the nodes 'x'. No other information in the d-rule need be investigated. (Distribution of information in the hierarchy somewhat complicates this procedure.)

18.5 A family of rules

Previous sections of this chapter have shown how individual rules may be encoded in networks and associated with categories. To really see how a semantic network might appear in practice, it is necessary to back away from individual rules and consider a more complete branch of the category taxonomy. Using this branch, relationships between rules for categories on different levels of the hierarchy may be observed.

Figure 18.12 presents a branch in the category taxonomy dealing with fastening events. Since this example is to emphasize the general structure of branches, individual rules in this network have been somewhat simplified to forestall a discussion of irrelevant detail.

The top category in this branch of the taxonomy is G-FASTEN which is shown to be a subset of U. G-FASTEN is the general category of fastening events. The d-rule associated with G-FASTEN indicates that category members have three participants: two distinguished #@parts (a set of



The Fastening Family

Figure 18.12

parts in an unsimplified version) and an #@actor who fastens the #@parts together.

Category G-FASTEN may be associated with English verbs "fasten" and "attach." Thus, for example,

"John attached the pulley to the shaft"

might be encoded as an element of G-FASTEN.

One of the subsets of G-FASTEN is <direct-screw> the category of events in which one object is screwed into another. The d-rule of <direct-screw> does not need to mention that the #@parts must be physical objects or even that <direct-screw> events involve a #@actor since this information is encoded at the higher level. The rule does, however, indicate (very coarsely) that both the #@parts must be threaded in a way which allows them to be screwed together.

Notice that <direct-screw> subsets G-FASTEN by placing more stringent requirements on the ranges of certain attributes. Category F-FASTEN (fasten with a fastener) subsets G-FASTEN by requiring the presence of a new attribute, @fastener. Very coarsely, the d-rule of F-FASTEN further states that the #@fastener must be capable of fastening the two parts. (Category CAN-FASTEN must have a very complex ns-rule.)

Subsets of F-FASTEN include <pin> and <tie>, etc. Category <pin> restricts the #@fastener to be a pin. Thus,

"Alma pinned the flower to her dress"

is an instance of the <pin> category while

"Simon tied Mary to the railroad track
(with a rope)"

is an instance of the <tie> category.

T-FASTEN is a subset of F-FASTEN which introduces a #@tool to aid in the fastening operation. T-FASTEN is further subset into <indirect-screw> (a fastener called a screw fastens the two parts together), <staple> and <hammer> by restricting the kinds of fasteners and allowable tools.

Also encoded for most categories, but unspecified in the network of Figure 18.12, are the process automata which indicate how the various fastenings are accomplished. These process descriptions must be very abstract for an abstract category such as G-FASTEN, but may be more precise for a category such as <hammer>. The pa for <hammer> might include information concerning how the #@tool is to be used.

18.6 Default rules

Artificial intelligence systems working in any but the most toy-like of worlds must have the ability to cope with missing or only partially specified information. Delineation rules are of considerable help in that they

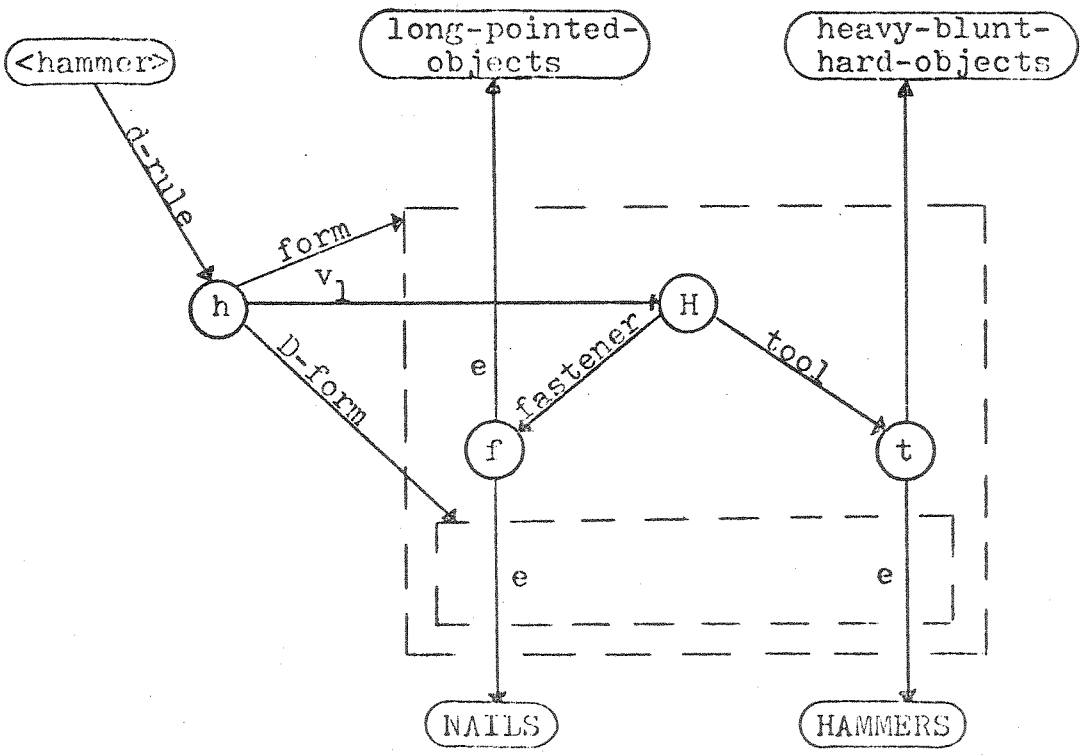
constrain the types and possible ranges of missing pieces of data. But for a variety of purposes it may be desirable to guess the values of missing arguments, making tentative assignments until some concrete evidence presents itself. Such default assignments may be accomplished by augmenting d-rules.

As an example of default value specification, consider the network of Figure 18.13 which delineates category <hammer>. According to this rule (and ignoring the inner form) a <hammer> event has two participants specified at this level of the category taxonomy. One of these is a #@fastener which must be taken from the set of long-pointed-objects and the other is a #@tool which must be taken from the set of heavy-blunt-hard-objects. (In actual practice, these range sets would be encoded as PHYSICAL-OBJECTS with appropriate restrictions.) Thus, a hammering event may involve an ice pick (the #@fastener) and the heel of a shoe (the #@tool) as in

"John hammered the ice pick into the boards
with the heel of his shoe."

But, normally, and in the absence of conflicting information, the #@fastener is assumed to be a nail and the #@tool is assumed to be a hammer. These assumptions are encoded by the #@D-form (the default form) of rule h.

A more complex default rule is encoded by the



A Default Rule for <hammer>

Figure 18.13

network of Figure 18.14. This network delineates and defaults the category of <wash> events. Participants in <wash> events include a #@washer, a #@CA (cleansing agent type - type is used here to avoid the mass noun problem), the #@washees (a set of similar things (such as dishes) to be washed) and the #@location at which the washing takes place. Possible ranges for each of these <wash> attributes are indicated by the #@form of d-rule h.

Washing events turn out to be very rich in default implications. For example, the statement

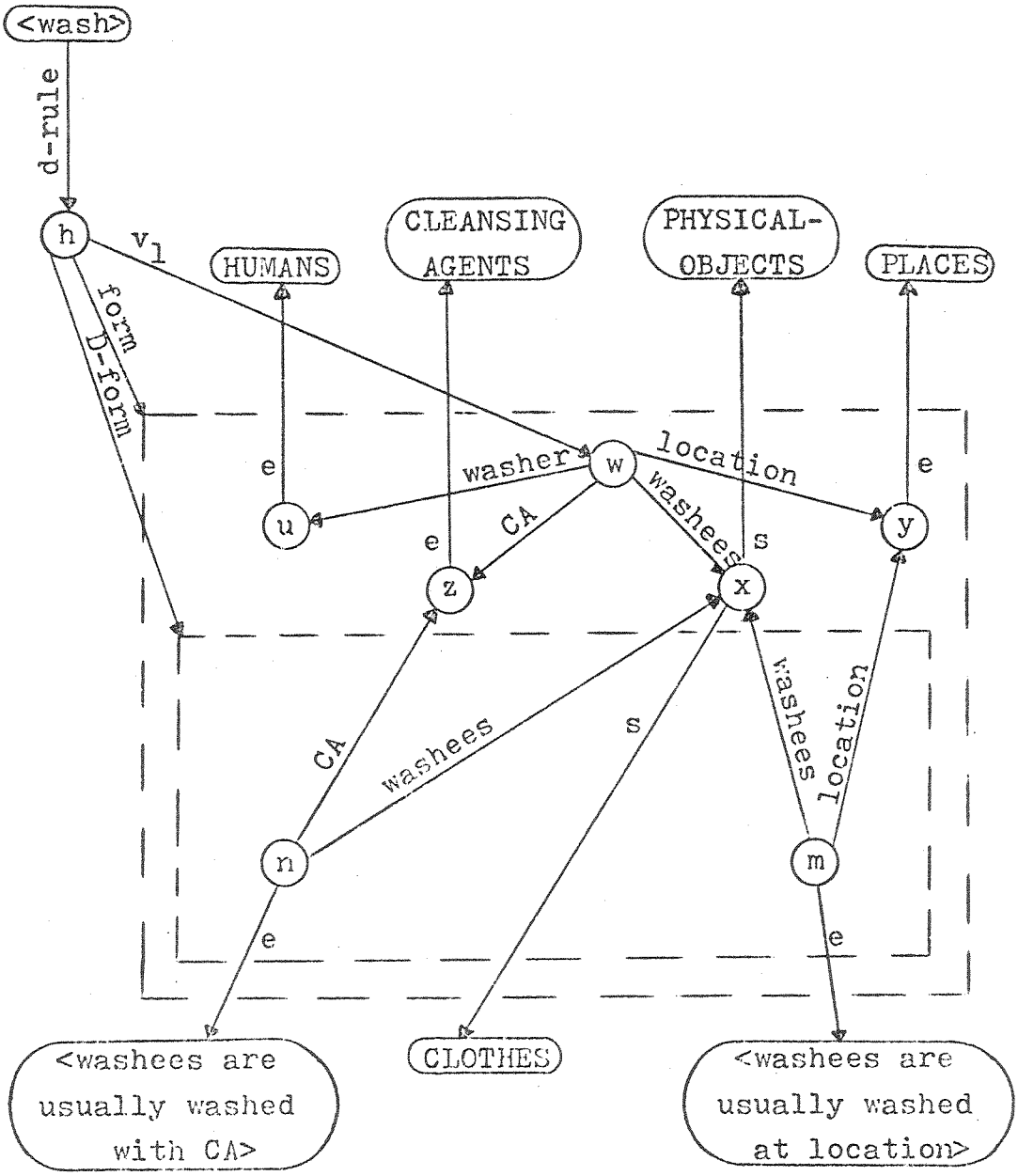
"Genie washed on Monday"

(as opposed to "Genie bathes every day") may be taken to mean that Genie washed clothes on Monday. In the network, node 'x' represents the things which are washed. In the absence of other information, the #@D-form of h indicates (tentatively) that x is a subset of CLOTHES.

Given the type of articles washed, it is possible to make an intelligent guess concerning what type of cleansing agent was used and where the washing took place. Consider

"Genie washed (clothes) (with laundry detergent) (in the utility room)."

"Genie washed the dishes (with dishwashing detergent) (in the kitchen)."



A Default Rule for Washing

Figure 18.14

"Genie washed her face (with facial soap)
(in the bathroom)."

"Genie washed her hair (with shampoo)
(in the bathroom)."

To encode these guesses in a default rule, special "usual" relations are defined. In short, the network of Figure 18.14 indicates that in the absence of more concrete information, it is to be assumed that #@washees are washed at their usual washing place using the usual cleansing agent.

For example, it may be known, as shown in the network of Figure 18.15, that Grandma's crystal compote is usually washed in Heirloom Crystal Cleaner. Thus, from

"Genie washed Grandma's crystal compote"

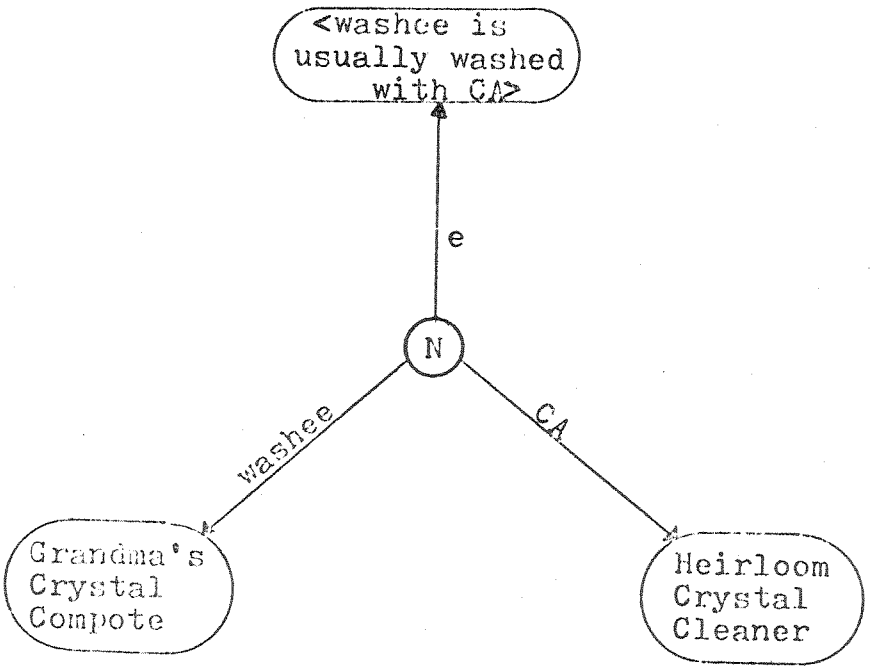
the <wash> default rule would imply that Heirloom Crystal Cleaner was used as the cleansing agent.

For general categories such as dishes, a rule such as that of Figure 18.16 may be encoded to indicate that dishes are usually washed in dishwashing detergent in the kitchen. (Note that the rule as shown allows different "usual" kitchens, but only one usual cleansing agent.) To properly default the statement

"Genie washed dishes"

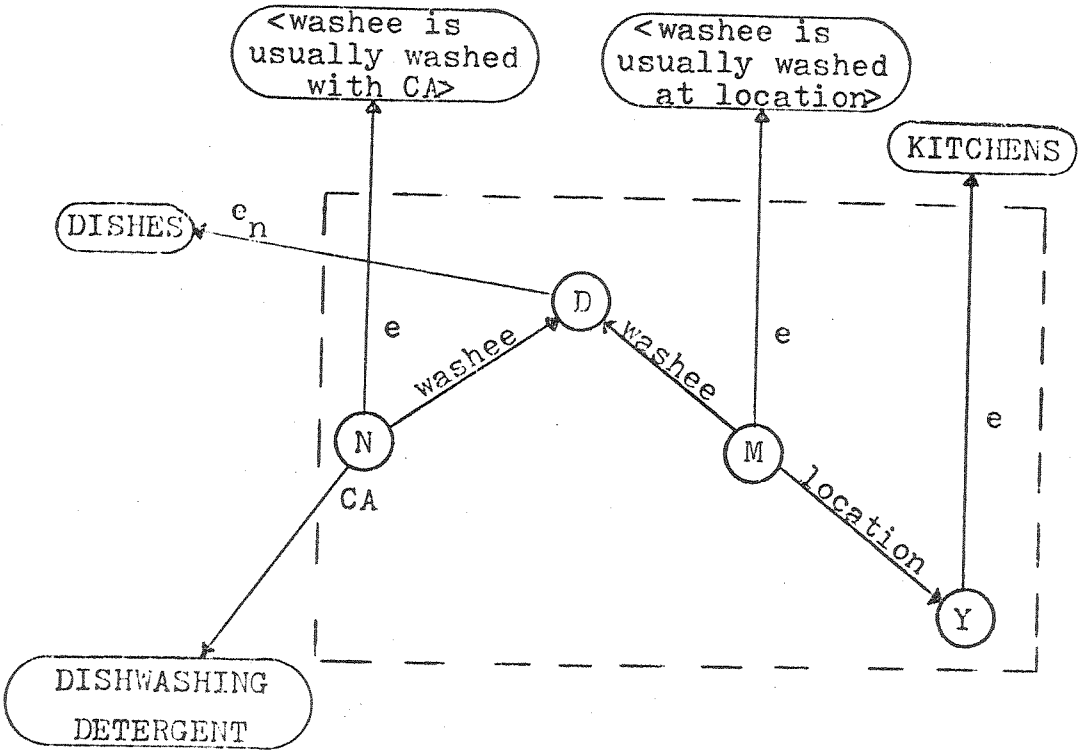
the nets of both Figures 18.14 and 18.16 must be used.

In closing this section, the author would like to



A "Usual" Cleansing Agent

Figure 18.15



A Rule of the "Usual"

Figure 18.16

note that default rules are at present only in the development stage. While they show promise, they have not yet been tested in any implementations.

18.7 Black box rules

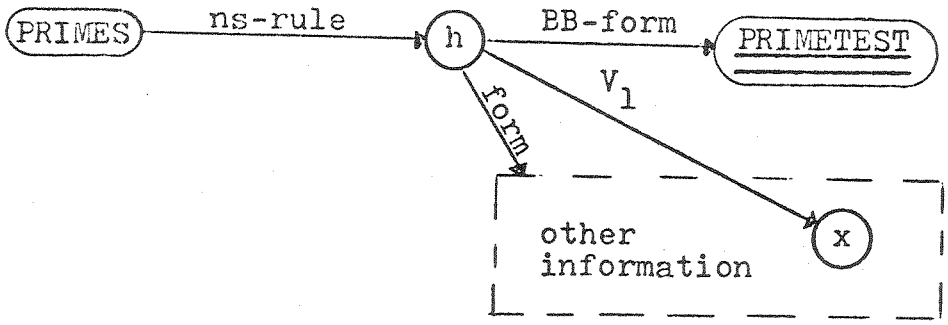
Rules associated with categories have two basic functions. First, they encode general information concerning category members and, second, they encode tests for the determination of category membership. While general category information needs to be in an explicit, manipulatable form, testing information is sometimes more conveniently encoded by computer subprograms.

For example, the category of primes may be defined by the ns-rule h

"positive integer x is not divisible by
any integer d between 2 and x."

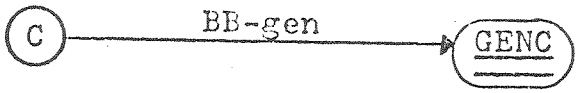
To determine whether or not some positive integer x is prime, it may be computationally convenient to express h as a standard computer subprogram (while maintaining an explicit encoding of h for use in other tasks). To see how this may be done, consider Figure 18.17.

In Figure 18.17, node 'PRIMES' represents the category of prime numbers. An ns-rule-arc links 'PRIMES' to 'h', which represents rule h. As would be expected, h has a #@v_1 and a #@form (whose internal structure has been



A Black Box Rule

Figure 18.17



A Category and its Black Box Generator

Figure 18.18

omitted). But rule h also has a #@BB-form encoded by node 'PRIMETEST'. PRIMETEST is the entry point of a Black Box (computer program) which tests whether or not a given object #x is a prime. Since #x is (hopefully) a number, the internal label of node '#x' might conveniently be (a pointer to) the standard computer representation of the number #x. Predicate-function PRIMETEST may be written in such a way that the computer encoding of the number #x may be found from '#x'.

18.8 Black box generators

It is generally uneconomical to explicitly enumerate all members of a category. Fortunately, for some categories there exist algorithms for successively enumerating (or generating) the category elements. Suppose GENC is a standard computer algorithm which generates the next element of category C (given certain reentry information). GENC is called a black box generator of C and may be associated with C through the attribute @BB-gen. The relationship between C and GENC is shown in Figure 18.18.

If for any reason it should be desirable for the system to understand a generation or testing operation, these computer precedures may be described (at some level of detail) by process automata.

Chapter 19

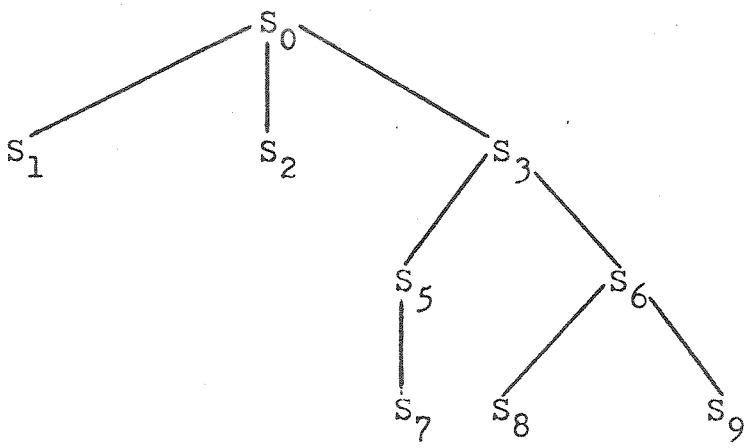
Net Space Partitioning

Theoretically, the elementary building blocks of networks presented in Chapter 15 are sufficient in themselves to record the information needed for language processing and understanding. However, knowledge processing by nets may be made more computationally convenient by introducing net space partitioning to set apart certain groups of nodes and arcs from other entries in the net. In fact, the #@forms of rules and general statements have already been introduced as entities which group (i.e., partition) selected nodes and arcs into special bundles.

19.1 The notion of net space

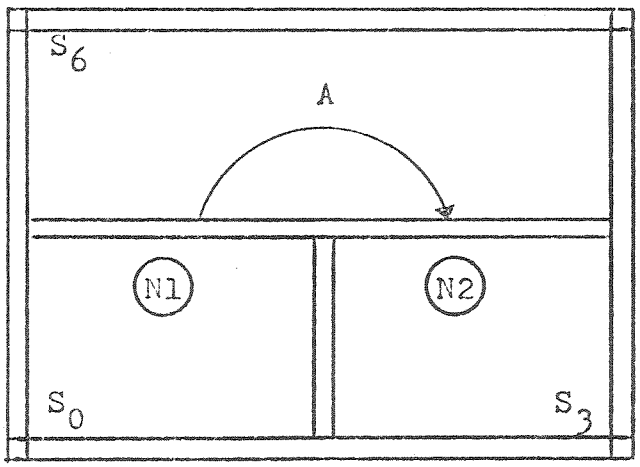
The essence of the partition concept is that every arc and every node in the network lies in exactly one of several net spaces. Nodes and arcs lying in the same space have been assembled into a unit and separated from other parts of the network. (Various reasons for this separation will be presented shortly.) While nodes and arcs lying in separate spaces may still be linked, the linkage must cross boundaries which separate one net space from another.

The various net spaces are arranged in a tree (or partial ordering) such as that shown in Figure 19.01. The



A Net Space Partition Tree

Figure 19.01



Two Nodes and an Arc, Each in Separate Spaces

Figure 19.02

total net is almost always viewed from some point in the net space tree. When viewing the total net from some net space S, only those nodes and arcs are visible which lie in S or one of its ancestors in the space tree. (There are some special exceptions to this rule.) Thus, for example, from space S_6 of Figure 19.01 only those nodes and arcs lying in net spaces S_6, S_3 and S_0 are visible.

As an example of net space partitions, consider the situation, illustrated in Figure 19.02, in which a node 'N1' lies in space S_0, a node 'N2' lies in S_3 and an A-arc from 'N1' to 'N2' lies in S_6. Using the tree of Figure 19.01, only 'N1' may be seen from S_0. But from S_6, the arc and both nodes may be seen.

In constructing figures to represent various network configurations, a few simple conventions have been followed throughout this work.

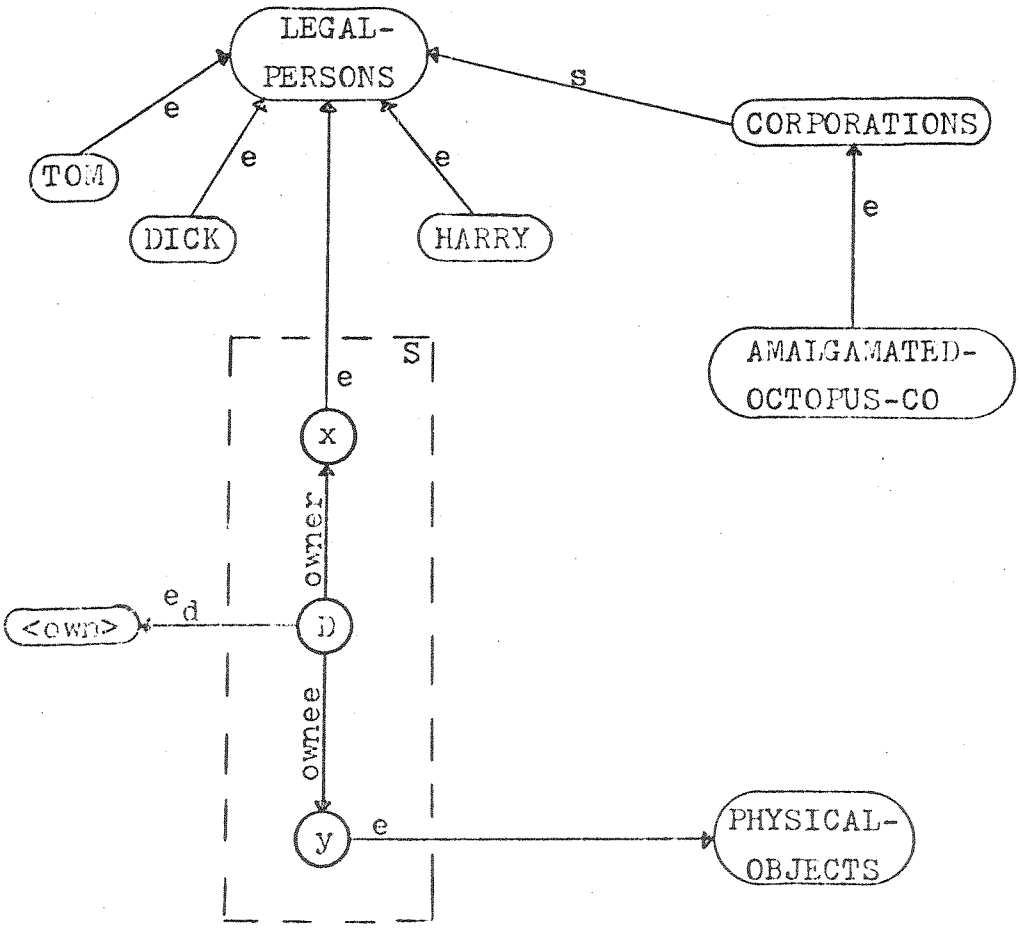
1) Nodes and arcs lying in the same net space are enclosed within a region bounded by a broken line.
2) An implicit broken line surrounds each figure. That is, the broken line delimiting the top space in each figure has been omitted.
3) An arc which crosses a broken line belongs to the region in which its label appears.
4) When one broken-line area is drawn within another, the inner area represents a space which is an immediate

descendant (in the partition tree) of the space represented by the outer region. Further, the inner area may be treated as a super-node which lies in the outer area.

19.2 The #@form as a net space

For applications in natural language understanding and for general knowledge representation, partitioning facilitates the encoding of quantified rules and general statements. Network renditions of these constructs have, of course, already been discussed. The thing to note here is that each #@form is actually a separate net space. Network information is stored in such a way that by knowing the name of the #@form net space, all nodes and arcs lying within the #@form space may be easily retrieved. Further, the #@form space is one level lower in the space partition tree than the level on which the associated form-arc appears. By virtue of being on this lower level, all nodes and arcs lying in a #@form are invisible when a rule or general statement is not in use.

The importance of this invisibility feature may be seen by considering the network of Figure 19.03. Suppose that for the performance of some task (e.g., to answer the question "Do you know the name of any person or corporation with net assets exceeding $1,000,000?") it is necessary to



Visible and Invisible LEGAL-PERSONS

Figure 19.03

find all explicitly represented legal persons. An appealing method for finding all such explicit entries is to begin processing at 'LEGAL-PERSONS' looking for all incoming e- and s-arcs. This method will lead to the discovery of 'TOM', 'DICK', 'HARRY' and eventually to 'AMALGAMATED-OCTOPUS-CO'. Further, the algorithm will (supposedly) discover variable 'x' of the d-rule of <own> since there is an e-arc from 'x' to 'LEGAL-PERSONS'. But 'x', being a variable which might stand for any legal person, is not an explicit representation of any person in particular and hence should not be considered.

Now the variable 'x' and its e-arc lie in a net space S which is the #@form of the d-rule of '<own>'. Space S is lower in the space tree than the net space in which 'LEGAL-PERSONS', 'TOM', etc., lie. Hence, neither 'x' nor its e-arc is visible in any branch of the space tree which does not have S as an ancestor. Search operations conducted with respect to net spaces which are not descendants of S need never see node 'x'.

19.3 Shapiro's network encoding of quantification

Partitioning off a portion of a semantic network for use as the #@form of a general statement is a new technique which has grown out of the research reported herein. Although #@forms are convenient for the encoding of

quantification, they are not essential. Three logically adequate techniques for the encoding of quantification without partitioning were cited in Section 17.4. Of these, the scheme of Stuart Shapiro (1971) is the best known and most likely rival to the scheme presented herein.

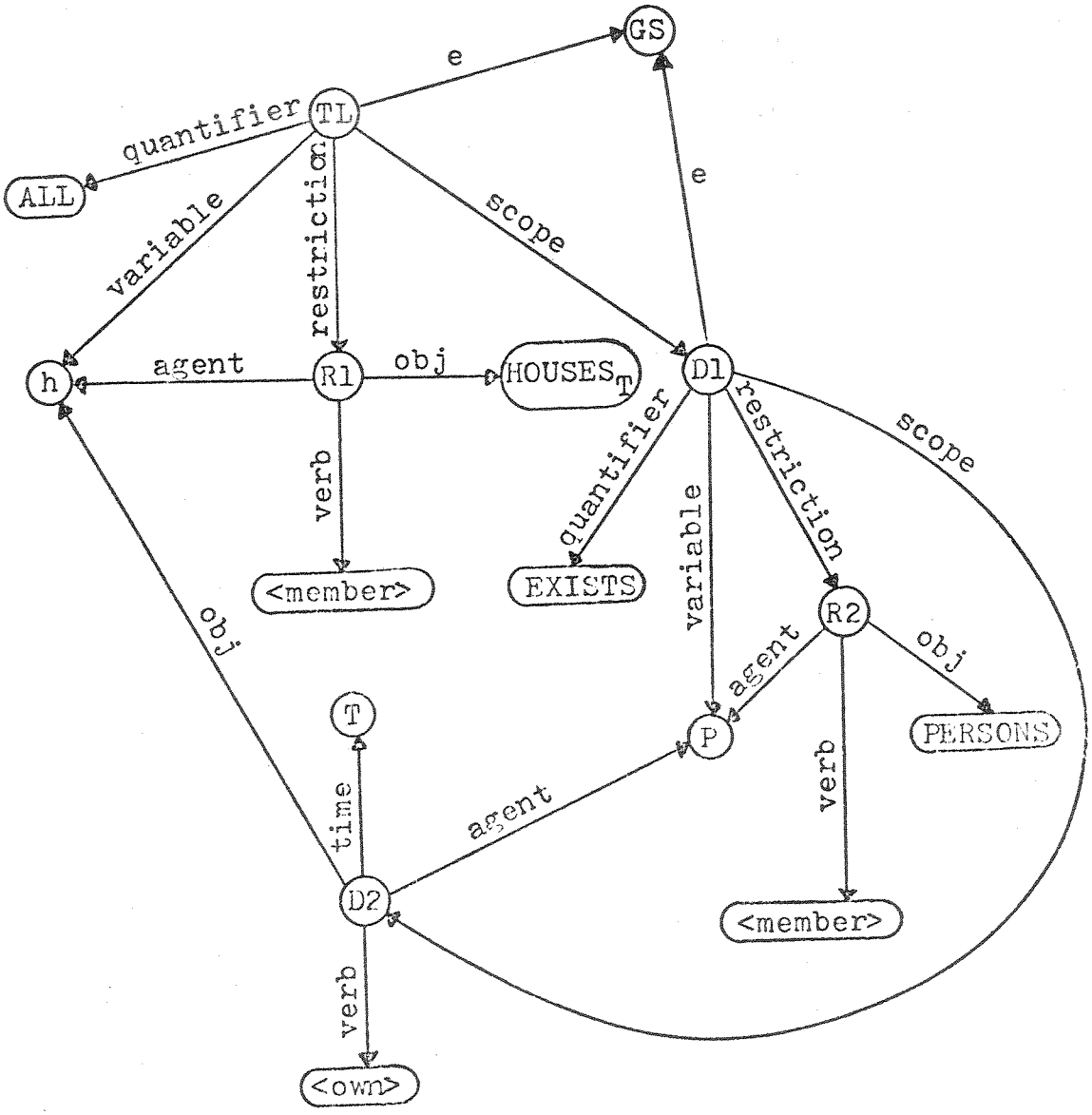
To compare Shapiro's scheme with partitioning, a Shapiro-like network is shown in Figure 19.04 which encodes the statement

$\forall h \in HOUSES_T$, $\exists p \in PERSONS$, p owns h at time T.

The partitioned network encoding of this statement is shown in Figure 17.03.

The interpretation of Figure 19.04 is as follows. TL, an element of GS, represents the complete general statement (just as g does in Figure 17.03). TL has four attributes, including an @quantifier, @variable, @restriction and @scope. Since the #@quantifier of TL is ALL, the #@scope of TL will be true for all assignments of values to the #@variable of TL which meet the #@restriction. The restriction on variable h is that its value must be a member of set $HOUSES_T$. Thus, for all assignments of variable h to values in $HOUSES_T$, the general statement D1 must hold. The interpretation of D1 is that there exist assignments of values to variable p from set PERSONS making statement D2 true. D2 is the statement that #p owns #h at time T.

To begin the comparison of Figures 17.03 and 19.04,



The Encoding of "Every House Has an Owner"

following Shapiro

Figure 19.04

notice first that Figure 17.03 is visually much simpler than Figure 19.04. Figure 17.03 contains 9 nodes, 9 arcs and 2 spaces (the top space and the #@form). Figure 19.04 contains 16 nodes, 20 arcs and 1 space. To be fair, node GS of Figure 19.04 might be eliminated along with its incoming e-arcs since Shapiro did not mention a set such as GS in his paper. (If these are eliminated, one must ask how all general statements may be found and also ask how it is possible to know what kind of entities 'TL' and 'D1' represent.) It also seems fair to subtract off 4 nodes and 4 arcs since the encoding of set membership in the #@restrictions of Figure 19.04 are unnecessarily expensive. With these adjustments, Figure 19.04 has 11 nodes and 14 arcs corresponding to requirements of 124% and 155% of the associated requirements of Figure 17.03.

It may be argued that nodes such as 'ALL' need be created only once and should not count in the encoding of any one general statement. On this basis, Figure 17.03 has four new nodes ('g', 'h', 'an:owning' and 'p') while Figure 19.04 has 7 ('TL', 'h', 'R1', 'D1', 'R2', 'p' and 'D2') or 75% more. All arcs in both figures are needed to encode the current general statement.

As more and more quantifiers are added, the system with partitioning will use significantly fewer nodes and arcs than the system without. Without partitioning, the

introduction of each quantifier adds one new node (the element of GS) and four new arcs (quantifier, variable, restriction, scope) as an overhead before variables and restrictions begin to be counted. Using partitions, the quantification

$$\forall \forall \ldots \forall \exists \exists \ldots \exists -$$

may be added for the cost of one node, one form and n + 1 arcs where n is the number of universal quantifiers. (There are n ∀v-arcs and 1 form-arc.) The existential quantifiers are free. Thus the quantification machinery for

$$\forall x \forall y \forall z \exists a \exists b \exists c$$

will cost 6 nodes and 24 arcs without partitioning, 1 node and 4 arcs with partitioning.

This comparison of Figures 17.03 and 19.04 indicates that partitioned networks may use significantly fewer nodes and arcs than networks without partitioning. This elimination of cumbersome structure makes partitioned networks easier to work with.

In addition to a simplification of structure, partitioned networks enjoy certain other advantages. The invisibility feature discussed in the last section is one of these. In looking for concrete instances of the verb <own>, there is nothing in the network of Figure 19.04 to prevent D2 from being found. Certainly, D2 may be

eliminated as a concrete instance of <own>, but only through extra processing. One might hope to eliminate D2 quickly by seeing the incoming scope-arc from 'D1'. But this test will not always work. If the general statement had been "Every house has an owner and a mortgage holder," then the scope-arc would have come into a node representing the conjunction of "p owns h" and "m holds mortgage on h." If the scope test doesn't work, the values of case arguments may be inspected for incoming "variable" arcs. For a concrete instance of <own>, every outgoing arc would have to be tested! Alternatively, one could run up the chain of conjunctions and disjunctions, but this could be quite long. In general, discovering the variable(s) within whose scope a given entity lies is nontrivial.

Turning to a similar issue, it is clear from Figure 17.03 that node 'an:owning' is inside the scope of h while 'T' is outside the scope. That is, it is clear that there is only one time T under consideration, but that there are many instances of owning events. This distinction is not clear in the network of Figure 19.04.

Perhaps the essential difference between the Shapiro technique and the partitioning technique may be explained in terms of parentheses. In linear languages such as predicate calculus notation, parentheses are used to single out portions of a statement for subjugation under

higher order operators. The central problem of implementing quantification and other higher order predicates in networks is how to encode parentheses or their equivalent.

Shapiro's approach is to encode parentheses by appealing to the conventional network formalisms of nodes and arcs. In his approach, information contained in parentheses should be pointed to either explicitly, by a variable- or scope-arc, or implicitly, by a pointer from a secondary entity which is pointed to by an explicit parentheses-making arc. (These pointers might run in either direction. Kay's back pointers to the variables of Skolem functions essentially use the same scheme in reverse.)

If there are explicit pointers to all parenthesized entities, there is a proliferation of pointers. If some pointers are carried implicitly, as in Shapiro's system, the network must be supplemented by a fairly large body of conventions for deriving the implicit linkages. To the extent that these conventions are not well defined, confusion will result. Even if the conventions are well defined, the deduction procedures needed to derive implicit scope linkages may be computationally expensive.

The approach of network partitioning is to go outside the conventional network formalisms to find a new and more direct method for encoding parentheses in networks. A net space performs exactly the same task in delimiting an

area in a net that a pair of parentheses does in delimiting a linear string. Every node and arc of a partitioned network is directly and explicitly linked to its corresponding set of parentheses (and indirectly, through the partition tree, to more general levels of parentheses). Since all the links are explicit, there is no need for a body of conventions for deducing implicit links. Since the linkage is provided by a mechanism especially designed for the purpose (and not through ordinary network arcs), the linkage is relatively economical.

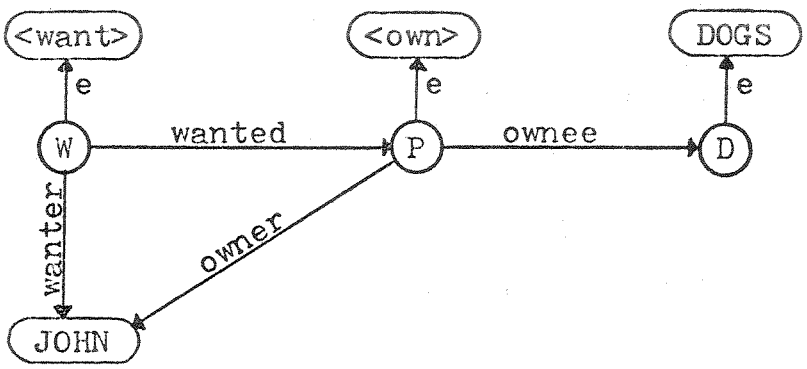
As might be expected, these special parentheses have applications outside quantification as will be seen in the sections which follow.

19.4 Using net spaces to encode hypothetical worlds

The isolation property of net spaces which distinguishes variables from their possible values may also be used to distinguish imaginary or hypothetical situations from their real world counterparts. To see this, consider the statement

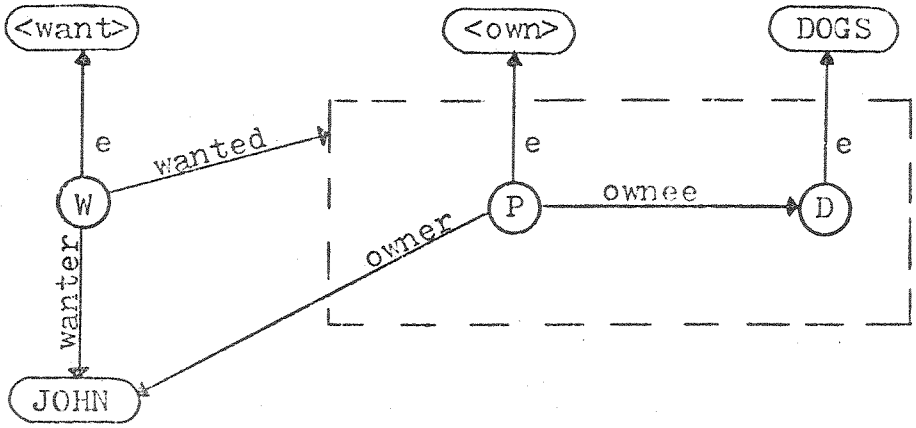
"John wanted to own a dog"

which may be interpreted as meaning that John wanted a situation to exist in which John owned some dog D. A first attempt at encoding John's wanting a dog is depicted in the network of Figure 19.05. In this network, node 'W'



"John Wanted to Own a Dog": First Attempt

Figure 19.05



"John Wanted to Own a Dog": Acceptable Version

Figure 19.06

represents a wanting situation with JOHN the #@wanter and P the #@wanted. P is, of course, the situation of JOHN's owning a (particular) dog D. (Note: Time arcs have been omitted for simplicity.)

There are at least two problems with this encoding. First, dog D is not a variable, but a particular element of DOGS. Thus, the network comes closer to saying that John wants Fido than to saying that John wants any dog he can get. The second (and most troublesome) problem with this network is that P is encoded as an ordinary element of <own>. Thus, by the rules of existential quantification presented in Section 17.3.3, the network asserts that owning situation P actually exists. But in reality, P need not exist since John may want to own a dog and yet never have his wish fulfilled.

The problems of Figure 19.05 are corrected by the network of Figure 19.06 which places the wanted ownership and the wanted (but unspecific) dog in a separate node space. The @wanted attribute of wantings now associates the wanting situation W with a hypothetical world. Paralleling the #@forms of rules, #@wanteds are associated with existence. In particular, the #@wanter wants the objects and relations encoded within the #@wanted space to have real existence. For the example at hand, JOHN wants the existence of objects #P, #D and relations

#P ∈ <own>

#D ∈ DOGS

#D is the #@ownee of #P

JOHN is the #@owner of #P.

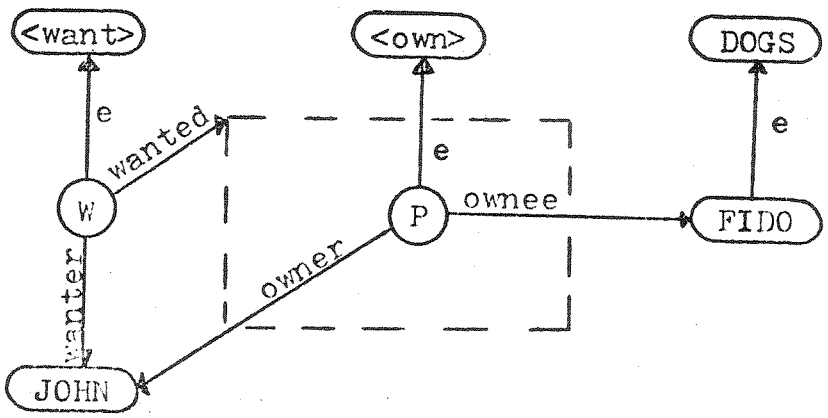
Had John wanted the particular dog Fido, the net of Figure 19.07 would have been appropriate.

The ability to place "unreal" information in quarantined hypothetical spaces has many applications. If John told Mary blah-blah where blah-blah may or may not be true, it may be encoded in a separate space. Likewise, stories, dreams and legends may be placed in hypothetical spaces. Thus, the data base may know a good deal about unicorns and sirens, including the fact that they are imaginary.

Hypothetical spaces also facilitate the encoding of multiple, alternative futures for use in planning. This special usage of net space partitioning is the subject of the next section.

19.5 The utility of partitioning in planning

One of the primary incentives for network partitioning is to allow systems which do planning to encode their data bases in semantic nets. (This allows the same base to be used for both planning and language processing.) The utility of net space partitioning for planning follows from the direct parallel between net spaces and the



John Wanted to Own Fido

Figure 19.07

contexts of such planning oriented languages as PLANNER, CONNIVER and QLISP.

In planning, a common procedure is to set up alternative world models depicting world states which may be achieved by performing various sequences of operations from an initial state. Each alternative world model is typically encoded with respect to a separate context. In nets, each alternative model may be encoded in a separate net space. As with the context mechanisms, only changes need be recorded in more local spaces. Most information for most worlds is encoded in partitions near the top of the space partition tree.

19.5.1 Spaces as instants in time

An interesting interpretation of the spaces that encode the world states produced by a planner is that they represent instants in time (or points within a branching tree of alternative time lines).

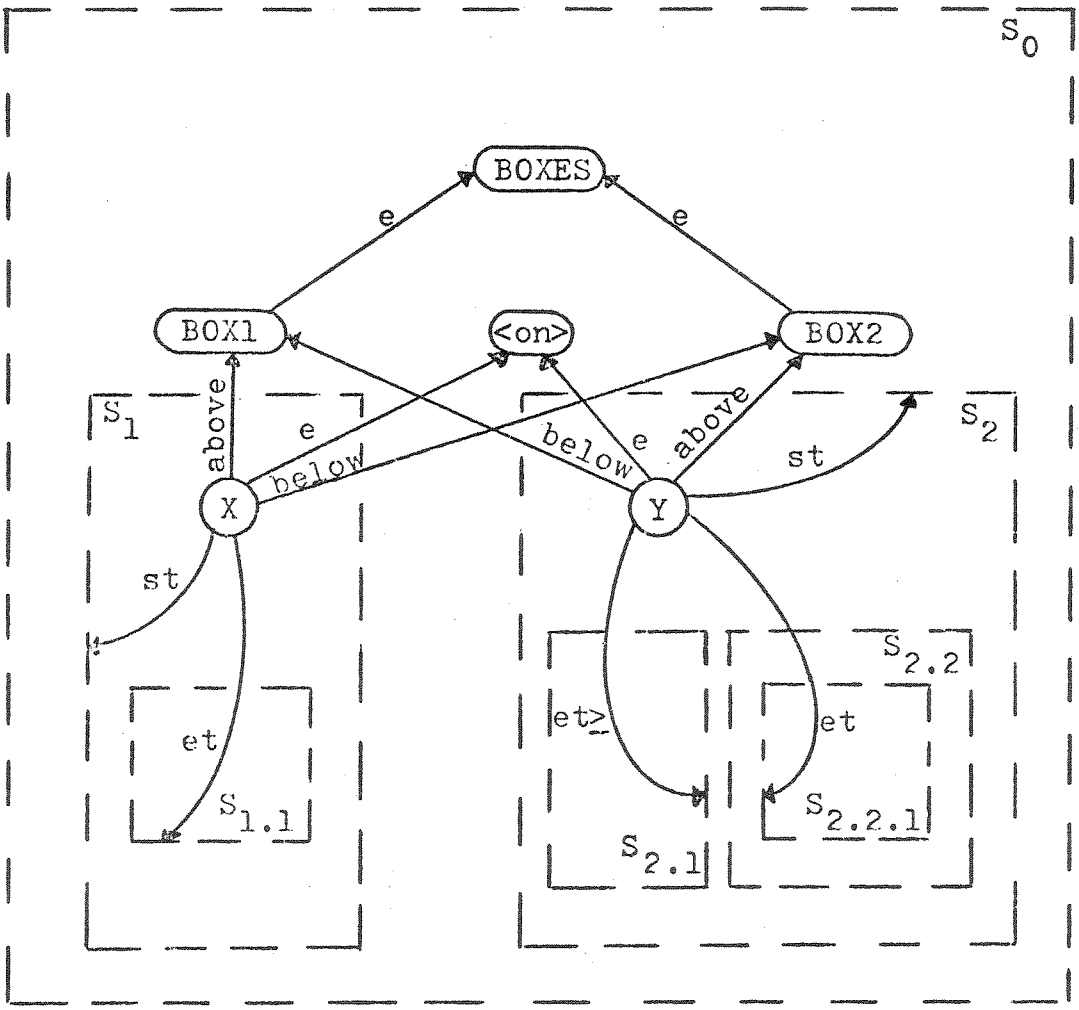
That is, the super-node which constitutes each state-space may itself be thought of as an element t of the set T of all instants in time. The information visible from the space is interpreted as the totality of (modeled) existence up to time t while the nodes and arcs lying in the space recorded changes in existence (objects being created or destroyed) occurring at t.

The use of this interpretation is shown in the planning spaces of Figure 19.08. In this figure, S_0 encodes the information available at the starting time. This includes the existence of boxes BOX1 and BOX2 and the situation set <on>. Space S_1 represents a descendant of S_0 in both the space tree and in time. S_1 encodes the situation X of BOX1 being on BOX2. This situation (produced from S_0 by some unspecified operation) has a #@st (a start time) of S_1. That is, super-node 'S_1' is pointed to as the representative of the instant at which the <on> situation began.

$S_{1.1}$ is a descendant of S_1 and encodes the effect of some (unspecified) operation. The effect has been to cause situation X to end. Thus, the #@et (the end time) of X is $S_{1.1}$.

Alternative plans to the S_0 - S_1 - $S_{1.1}$ sequence involve passing through state S_2. As a rival to S_1, S_2 encodes the effects of some operation which changes S_0 by placing BOX2 atop BOX1 to produce situation Y.

Suppose that from state S_2 the planner considers some operation whose effects might or might not cause situation Y to cease. The known results of the operation (not shown) may be recorded in a new space $S_{2.1}$. Since Y may have ended at time $S_{2.1}$, there is an et$\geq$-arc from 'Y' to '$S_{2.1}$' lying in $S_{2.1}$. The et$\geq$-arc encodes the fact that the end time of Y is greater than or equal to $S_{2.1}$.



Net Spaces as Planning Contexts

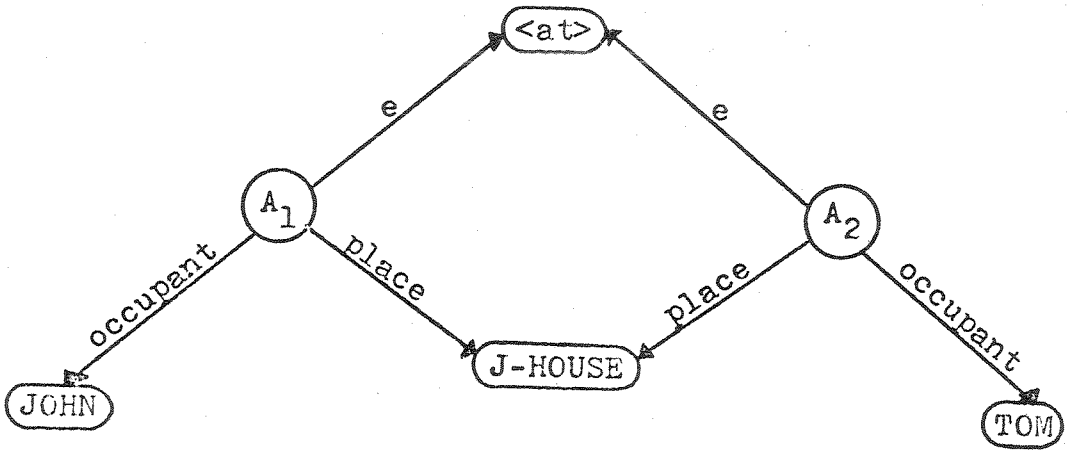
Figure 19.08

As a rival to $S_{2.1}$, the state of space S_2 may evolve into the state represented by space $S_{2.2}$ in which Y is known to be unaffected. $S_{2.2}$ may be transformed by subsequent operations to eventually produce the state of space $S_{2.2.1}$ in which situation Y ceases as indicated by the et-arc from 'Y' to '$S_{2.2.1}$' lying in $S_{2.2.1}$.

19.6 Relating net spaces to conjunction, disjunction, implication and negation

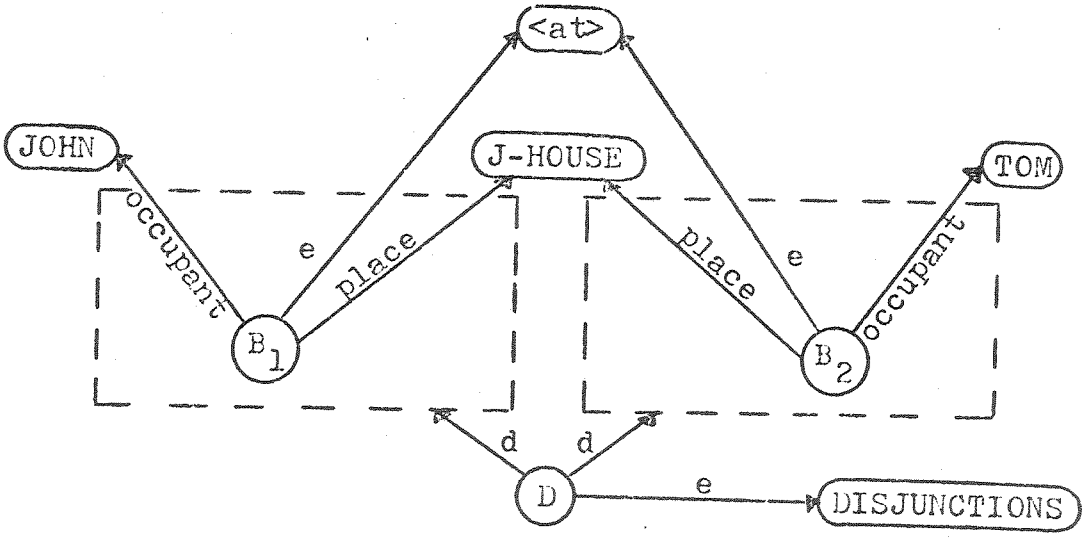
As stated in an earlier section (17.3.3), the very presence of a node or arc in a network is an assertion of existence (with respect to the world modeled by the space in which the node or arc lies). For example, the network of Figure 19.09 encodes the conjunction of two "at" events, A_1 and A_2. That is, the network shows both JOHN and TOM to be at J-HOUSE.

The encoding of disjunction is more complex as is seen by considering the disjunct "either JOHN or TOM is at J-HOUSE." Nodes encoding "JOHN at J-HOUSE" and "TOM at J-HOUSE" may not be placed in the top space of the net since this would imply the coexistence of both situations. Hence, each of these "at" situations is partitioned off into a separate net space as shown in Figure 19.10. That one or the other (or both) of the hypothetical worlds encoded by these spaces may be mapped onto real existence is



A Conjunction of <at> Events

Figure 19.09



A Disjunction of <at> Events

Figure 19.10

encoded by node 'D'. D, an element of DISJUNCTIONS, takes an arbitrary number of @d attributes. The interpretation of D is that one or more of the #@d has real existence.

The encoding of implication is similar to that of disjunction. Consider the implication

"JOHN at J-HOUSE implies TOM at J-HOUSE"

which is encoded by the network of Figure 19.11. Since neither "JOHN at J-HOUSE" nor "TOM at J-HOUSE" is known to be true, both must be partitioned off from reality. But the implication, I, that "JOHN at J-HOUSE implies TOM at J-HOUSE" <u>is</u> real and appears in the top space of the network. The interpretation of I is that its #@implication may be mapped into real existence if its #@precursor can be mapped onto reality. That is, if the #@precursor space encodes situations which have existence in reality, then the #@implication space must also encode situations which have real existence.

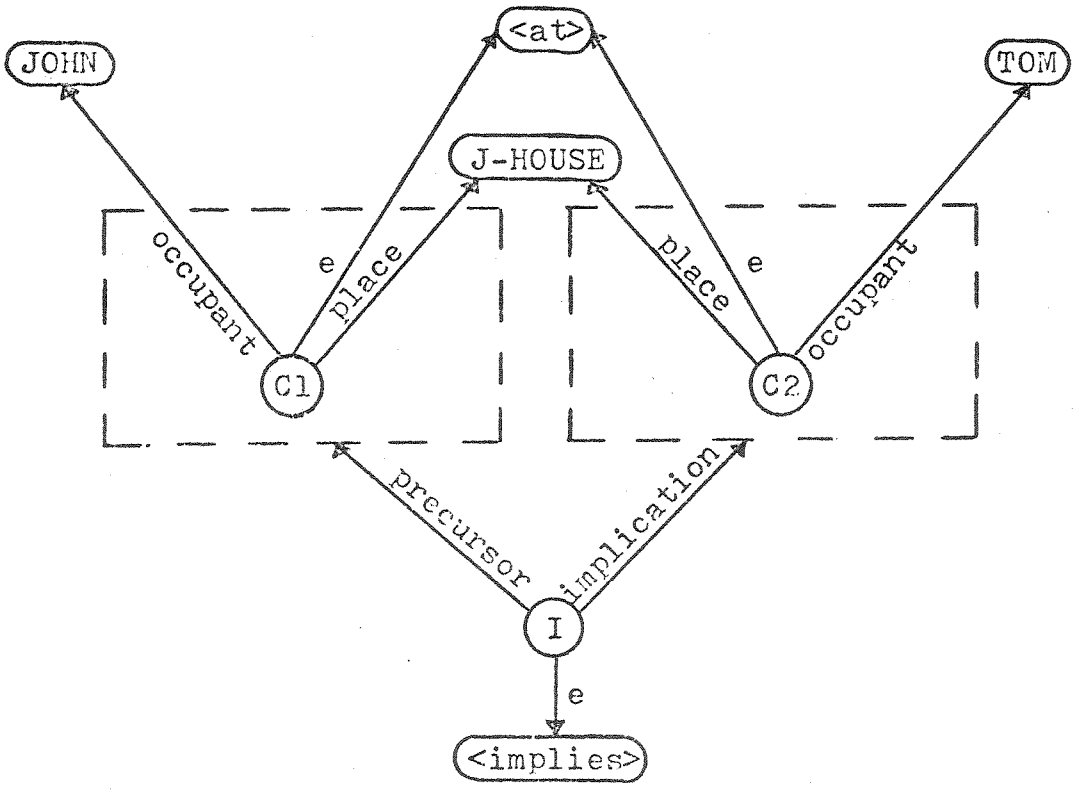
Implication statements as stated in English often carry implicit quantifications. For example,

"If John is at some place p, then Tom is there, too"

may be rendered by the predicate calculus expression

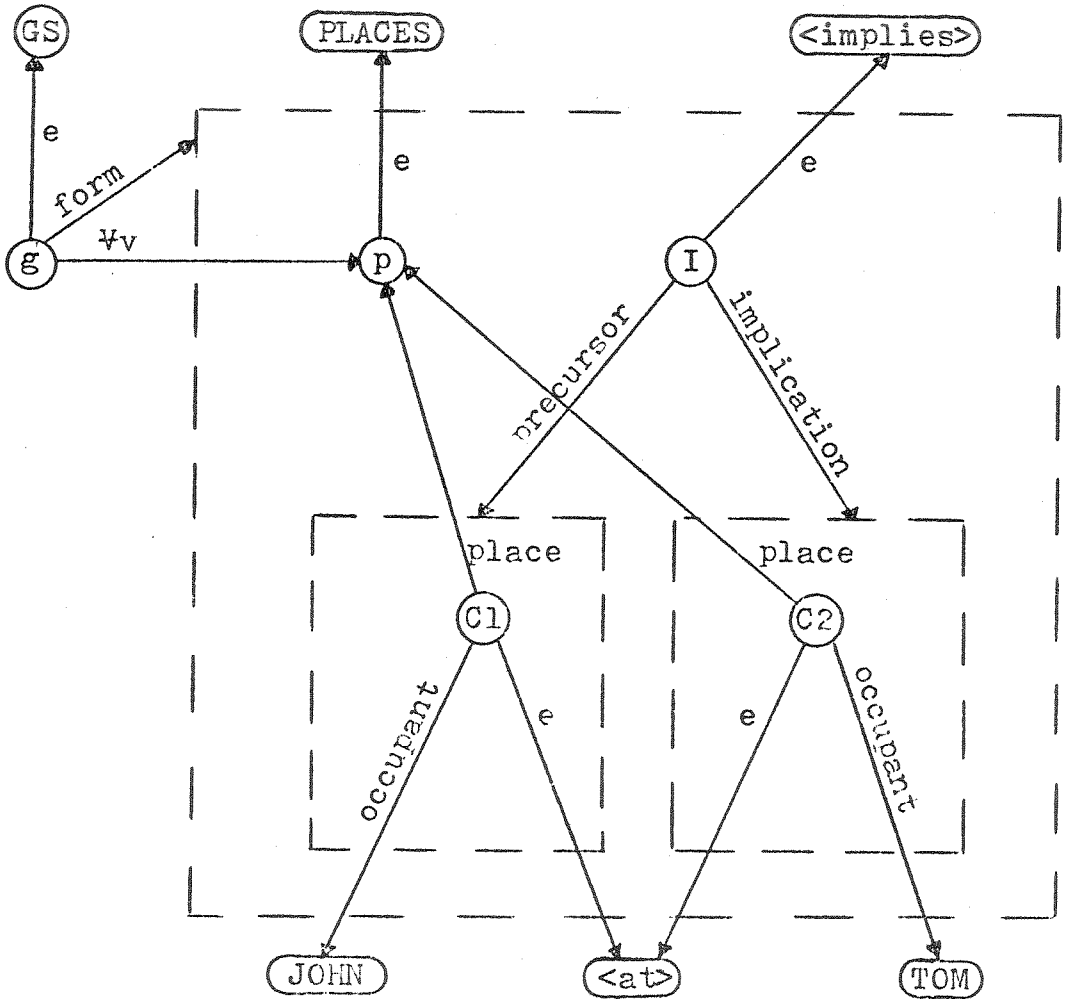
$$(\forall p \in \text{PLACES})\ [(\text{AT, JOHN, p}) \Rightarrow (\text{AT, TOM, p})]$$

which makes the quantification over places explicit. This statement is encoded by the network of Figure 19.12.



If JOHN is at J-HOUSE then TOM is at J-HOUSE

Figure 19.11



$(\forall p \in \text{PLACES})\ [(\text{AT, JOHN, p}) \Rightarrow (\text{AT, TOM, p})]$

If John is at Some Place, Then Tom is There Too

Figure 19.12

The encoding of negation may be accomplished by following the same pattern used for disjunction and implication. For example,

"JOHN is at J-HOUSE, but TOM is not at J-HOUSE"

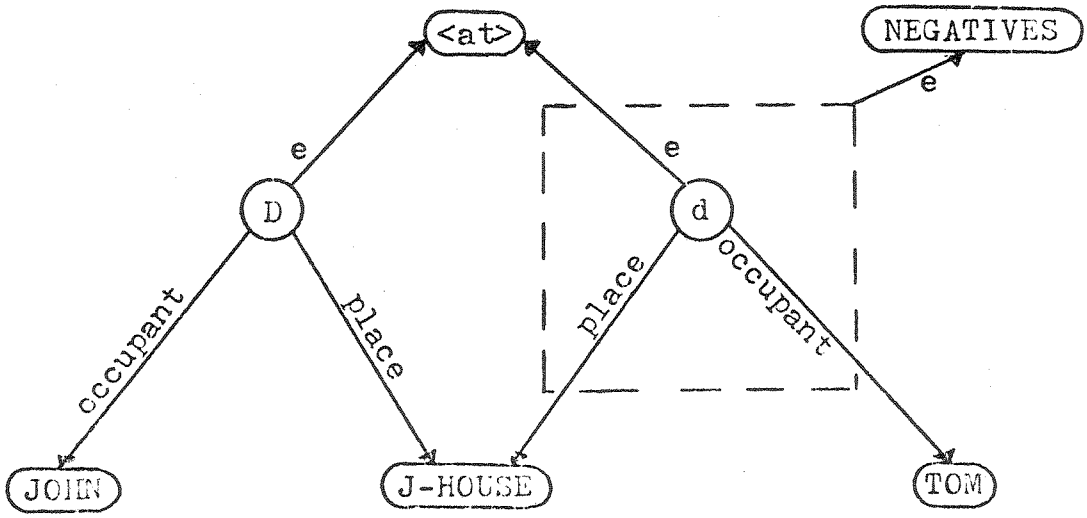
is encoded by the network of Figure 19.13 while

"Nobody is at J-HOUSE"

is encoded by the network of Figure 19.14.

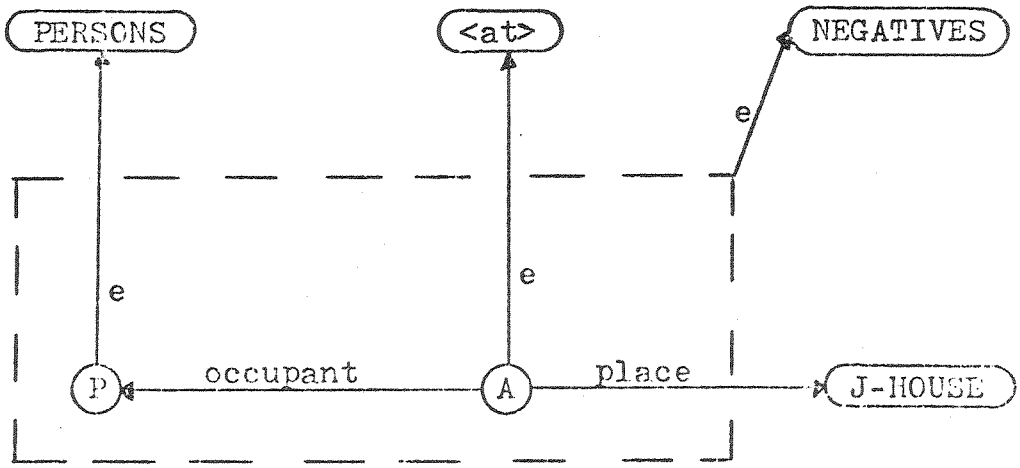
19.7 Using net spaces to encode levels of abstraction

Abstracting by rule partitioning has been discussed in Section 11.4.2. The thing to note here is that rules may be partitioned by placing them in different net spaces. For example, consider the network of Figure 19.15 which encodes information concerning electrical resistors. The category RESISTORS is encoded in net space S_0. At separate lower levels S_1 and S_2 there are encoded e_n-arcs linking RESISTORS to rules concerning the electrical properties of RESISTORS (in S_1) and the mechanical properties of RESISTORS (such as shape, color, weight, etc., in S_2). From S_1, the electrical properties of RESISTORS are visible while the mechanical properties are hidden. From S_2, this situation is reversed. If the partition tree is generalized to a lattice, see Figure 19.16, then a space S_3 may be defined below both S_1 and S_2. From S_3 rules regarding both the mechanical and electrical properties of resistors



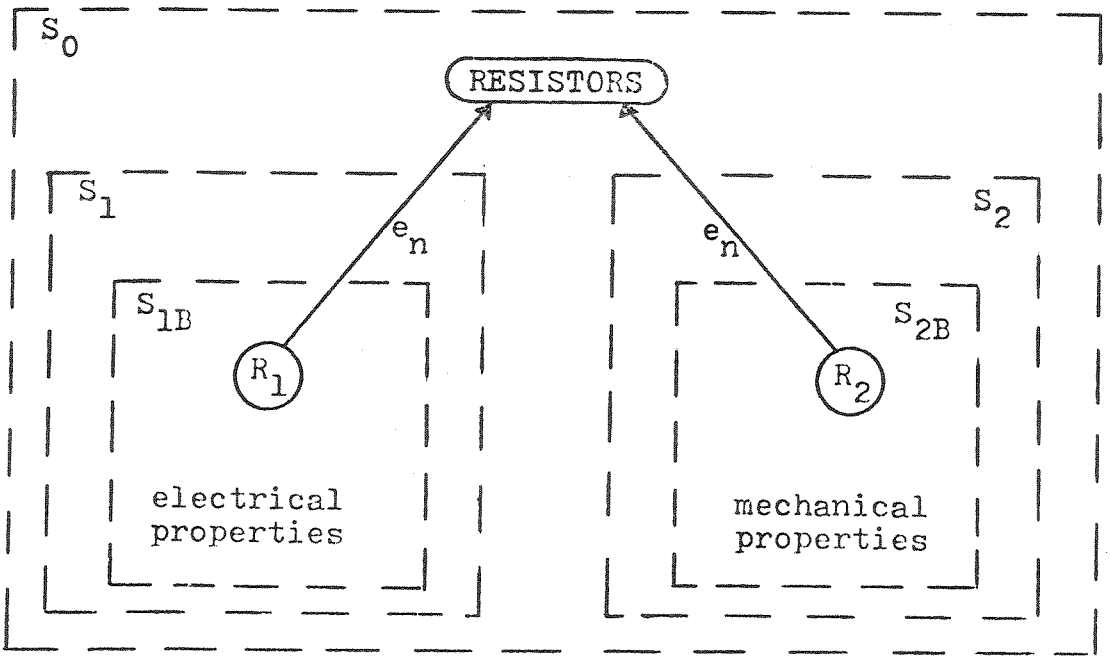
JOHN is at J-HOUSE but TOM is not at J-HOUSE

Figure 19.13



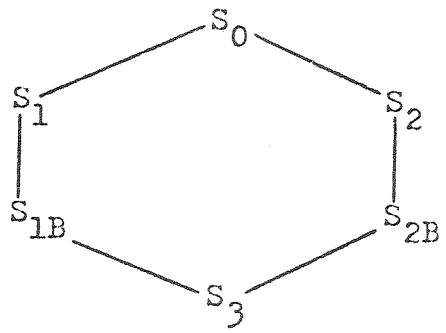
Nobody is at J-HOUSE

Figure 19.14



Two Rules Concerning Resistors

Figure 19.15



A Space Partition Lattice

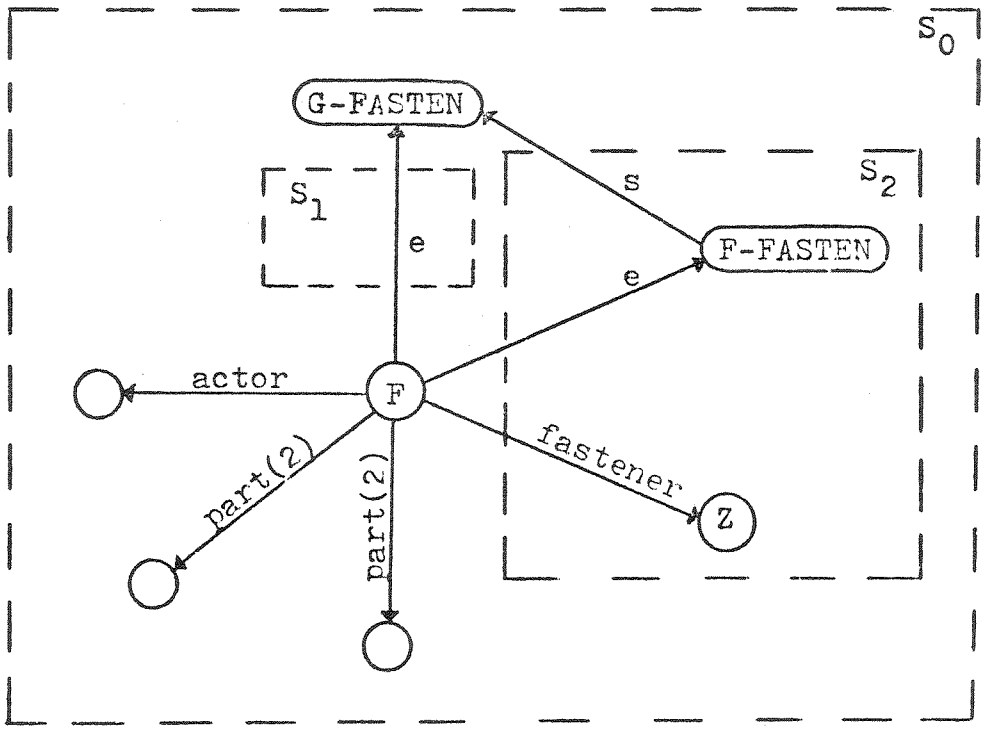
Figure 19.16

may be viewed.

The abstraction example just presented is an instance of abstraction with respect to semantic domain. It is also possible to encode various levels of detail (Sacerdoti-like abstraction levels) within the same semantic domain. The network of Figure 19.17, using the partition tree of Figure 19.18, encodes a fastening event F at two different levels of detail. From S_1 it appears that F is a direct element of G-FASTEN (see Section 18.5) and involves only a #@actor and two #@parts. From S_2, F is seen to be an element of F-FASTEN and hence is known to involve a #@fastener in the fastening operation. The #@fastener Z in net space S_2 is a detail of the fastening which is suppressed in S_1.

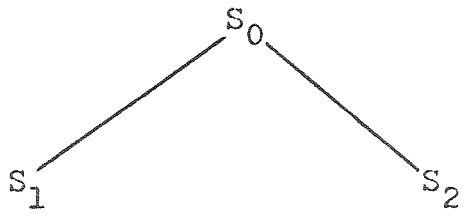
19.8 Using net spaces to aid in parsing

To see a very fundamental application of net spaces in natural language processing, consider the parser of a net-based system. The job of the parser is to translate natural language inputs into an internal representation which is understandable by the system. In producing the parse, the parser attempts to weld sequences of words into ever larger phrases. Each time a new phrase is proposed (with the aid of a syntax and existing sub-phrases), knowledge residing in the semantic net which constitutes the



Detail Abstraction Using Space Partitions

Figure 19.17



A Simple Partition Tree

Figure 19.18

system's data base is used to test the meaningfulness of the sequence (see Section 18.4).

In the process of determining the semantic well-formedness of a phrase, the system builds up a net structure which represents the phrase's meaning. These parser-built network structures, representing information (thought to be) encoded by the natural language input, must not be allowed to become confused with internal knowledge. Basically, there are two reasons for insisting upon this separation. First, it is important for the system to be able to distinguish between what it knows (or believes) to be true and what it has been told. Second, the process of parsing produces many erroneous interpretations of portions of the input sequence.

Amplifying the first reason, even after an input has been parsed completely, its intended meaning may not be obvious. That is, the input must be interpreted within the context of the current dialog. In this interpretation, anaphoric references and ellipsis must be resolved and conversational postulates (Gordon and Lakoff, 1975) must be brought to bear. Ultimately, dialog analysis may have a profound impact upon what the input is taken to mean. Once the intended meaning has been determined, the input may be copied into the knowledge base if the input is a believable statement. If the input is a question or command, other

appropriate actions must be taken. In any event, the input must be recorded in the discourse archives since it forms a part of the context in which future inputs will be interpreted.

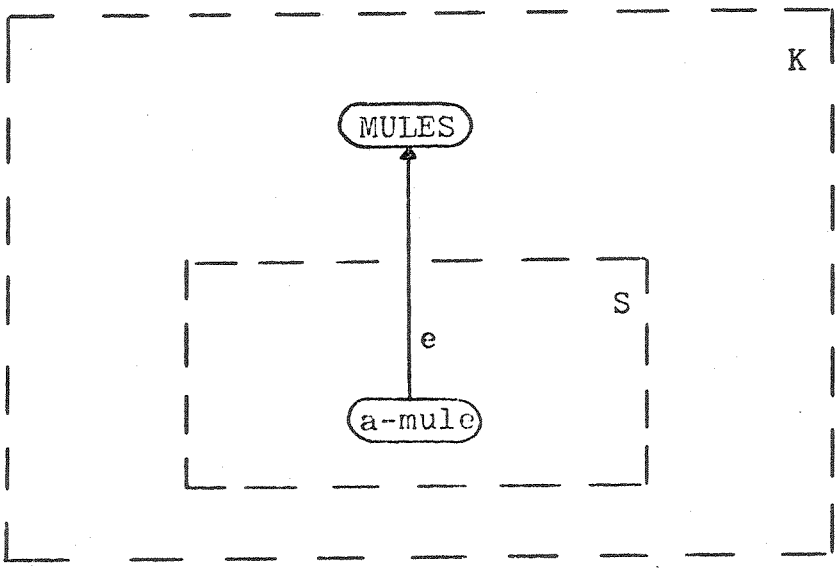
Amplifying the second reason for keeping inputs separated from internal knowledge, it is well known that even the best of parsers may follow paths through a grammar which, although they look promising within the local context, result in the construction of inappropriate interpretations of parts of the input. For example, a parser might erroneously identify the sentence "the man bought the mule" in the input sentence "the man bought the mule a harness." Or it might identify the noun phrase "the scholarly work" (as in "the scholarly work was published by the Tailor Press") in the input "the scholarly work long hours." Until the correct parse of an input has been identified, guesses concerning what phrases the input contains must be separated from other data in the system.

Just as would be expected, the structures representing inputs may be constructed in net spaces which are isolated from the other data in the net by the net-partition. With the system's general knowledge visible from some space K, representations of inputs may be constructed in "scratch" spaces S_i which lie below K in the partition ordering. Thus, information recorded in the scratch spaces

is invisible from K, but the general knowledge encoded in K (and its ancestors) is available for reference from the scratch spaces. For example, the meaning of the phrase "a mule" is encoded in space S of Figure 19.19 which lies below space K. The information recorded in S (the interpretation of the phrase) consists of a node which, by virtue of an e-arc to the node 'MULES' in K, represents a mule. In general, input information recorded in scratch spaces is always linked (by arcs) to concrete information visible from space K. This is a consequence of the fundamental principle that inputs are understood through an appeal to existing knowledge.

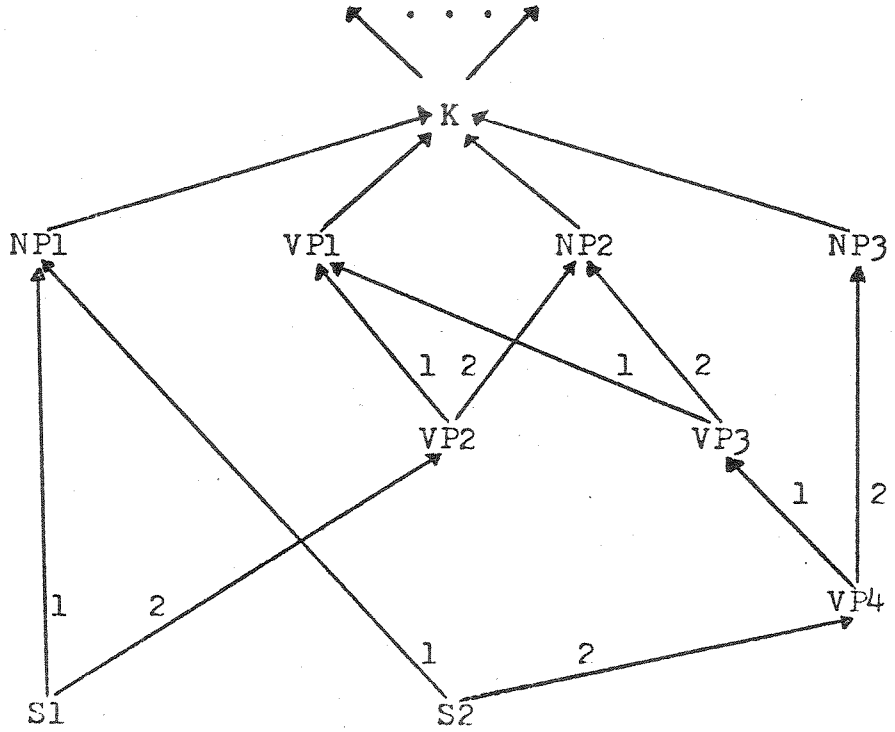
By generalizing the space partition tree into a directed graph such as that shown in Figure 19.20, net partitioning may do far more for a parsing system than just separate inputs from the data base. When viewing the network from a space S in a partial ordering like that of Figure 19.20, all nodes and arcs lying in spaces which may be reached from S by following upward paths in the directed graph are visible. For example, from space VP2 it is possible to see structures encoded in spaces VP2, VP1, NP2 and K. (Numbers appearing as labels on some arrows are artifacts of bookkeeping which will be explained shortly.)

By relaxing the tree structure into a partial ordering, the following scheme may be used to build up



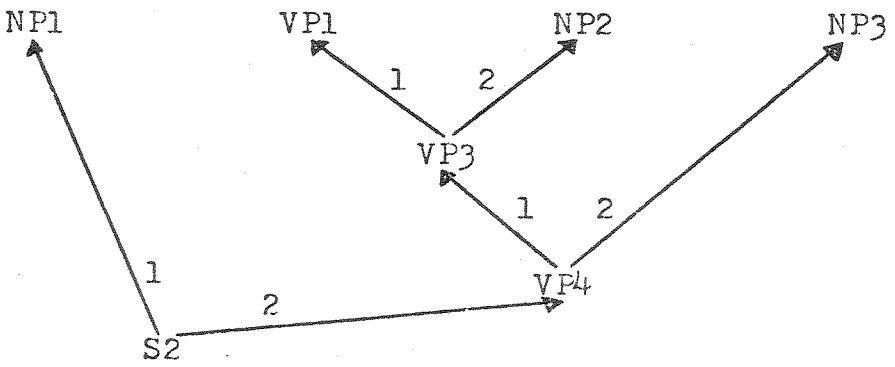
The Noun Phrase "a mule" as an Input

Figure 19.19



A Partial Ordering of Net Spaces

Figure 19.20



Paths from S2 to K

Figure 19.21

parses. First, the most elementary sentence fragments are represented by spaces which are direct descendants of K. Then, each complex phrase which subsumes multiple subphrases is encoded in a space which has as ancestors each of the spaces encoding its subparts. This new space inherits the structures representing the meaning of each of its subparts and need only record the interrelations of (or modifications to) its constituents. Further, competing hypotheses concerning the use of a common subphrase may share the subphrase's representation structure without confusion.

As an example of the application of this scheme, consider Figure 19.22 which indicates how the sentence

"The-man bought the-mule a-harness"

is parsed using the grammar

S → NP VP

VP → VP NP

where

VP → bought

NP → The-man

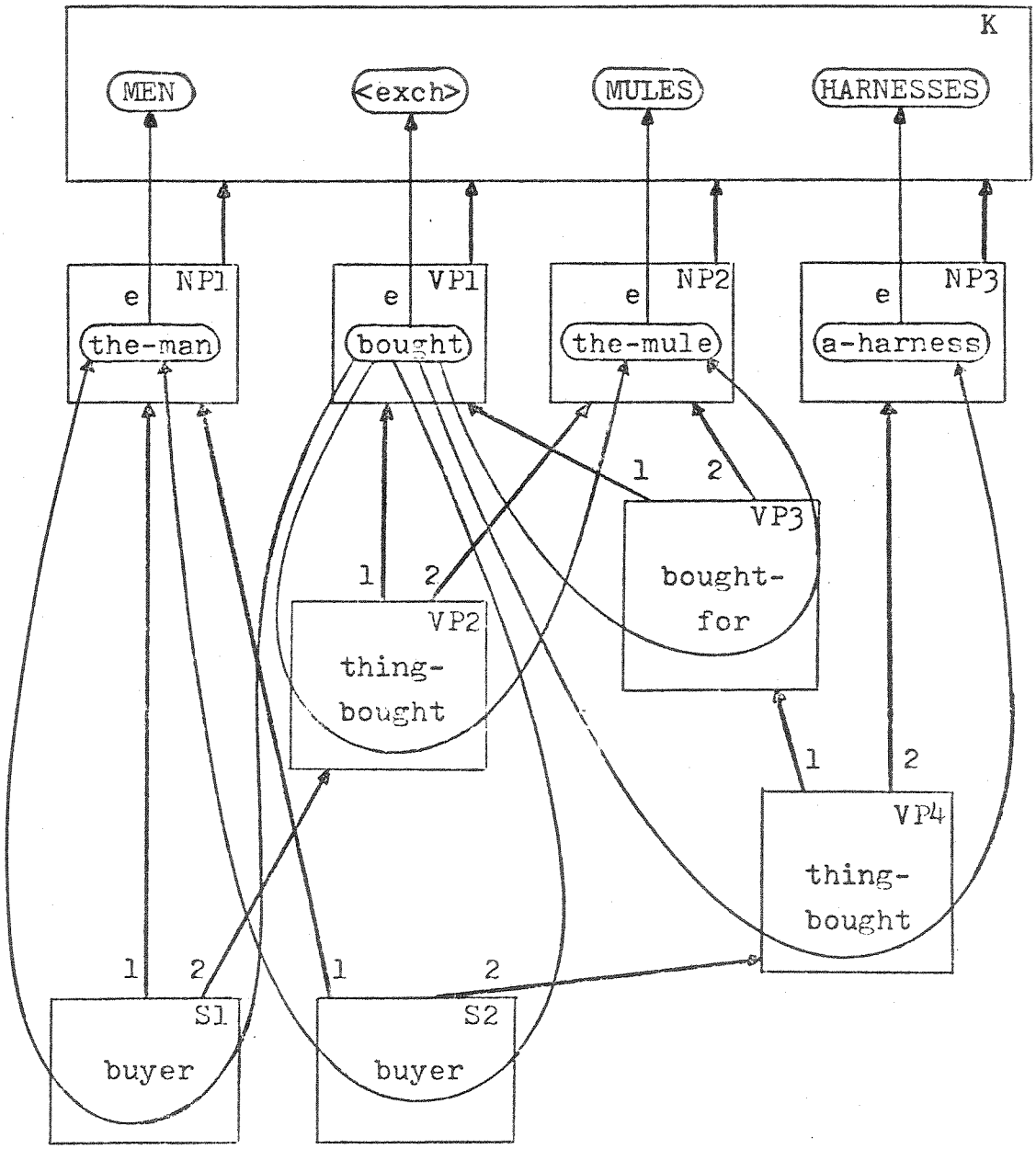
→ the-mule

→ a-harness

(Note: Hyphenated constructions are used to suppress distracting details.)

In this figure, each box represents a net space and arrows between spaces indicate the partial ordering.

At the start of processing, the system contains the knowledge recorded in space K. That is, it knows about



The-Man Bought the-Mule a-Harness

Figure 19.22

men, mules, harnesses and exchange events. Upon spotting the token "the-man", the system sets up a space, NP1, below K in the partial ordering. Within this space, a structure is constructed to represent the meaning of "the-man". Likewise, new spaces are set up to encode the other tokens in the sentence. (Note: "Bought" maps onto its canonical form "exchange.")

Using the grammar (and semantic knowledge such as that encoded in delineation rules), the parser may group the primitive subphrases into larger units. For example, the rule

VP → VP NP

may be used to group "bought" and "the-mule" into a longer phrase. Indeed, due to ambiguities in semantics, these subphrases may be combined in either of two ways. The first of these is encoded in space VP2, a descendant of both VP1 and NP2. According to this interpretation, the-mule is the #@thing-bought in the exchange event. An alternative interpretation is encoded in space VP3, also a descendant of VP1 and NP2 and a rival sister of VP2. According to this interpretation, something has been bought-for the-mule.

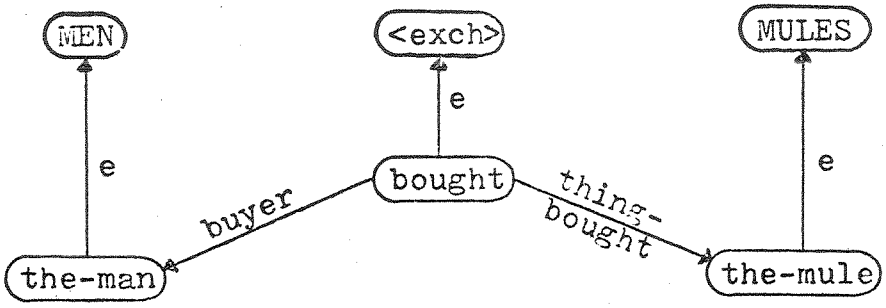
As processing continues, larger sentence fragments are produced until two complete sentences are discovered and encoded in spaces S1 and S2. At the end of processing,

the input as viewed from S1 looks like the network of Figure 19.23 while the input as viewed from S2 looks like the network of Figure 19.24. Since the parse corresponding to space S1 has not successfully explained the fragment "a-harness", it is rejected and S2 is accepted as the meaning of the input.

The partial ordering of spaces indicated in Figure 19.22 is identical to that represented more clearly in Figure 19.20. Viewing this ordering from the vantage of space S2 (and ignoring all links to space K) yields the structure of Figure 19.21 which, because of the choice of space labels, may be recognized as the parse tree of the input sentence!

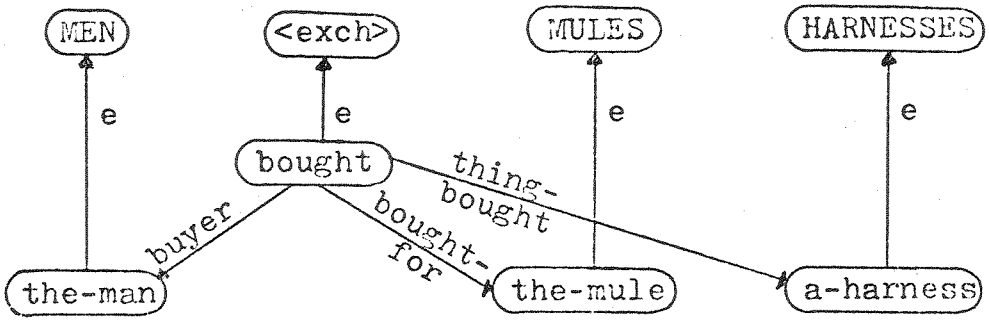
The structure thus built by the parser turns out to be well suited for later use by a dialog analyst. The semantic representation of the total sentence and each of its syntactic subparts is encoded in a separate net space. Further, the syntactic structure of the input is reflected in the partial ordering of the net spaces as a by-product. (To indicate the order in which surface structures appeared in the input string, arrows indicating the partial ordering of spaces have been numbered. Thus, for example, S2 was constructed from NP1 and VP4 with NP1 appearing first in the input.)

A comprehensive description of a language system



View of the Parse from S1

Figure 19.23



View of the Parse from S2

Figure 19.24

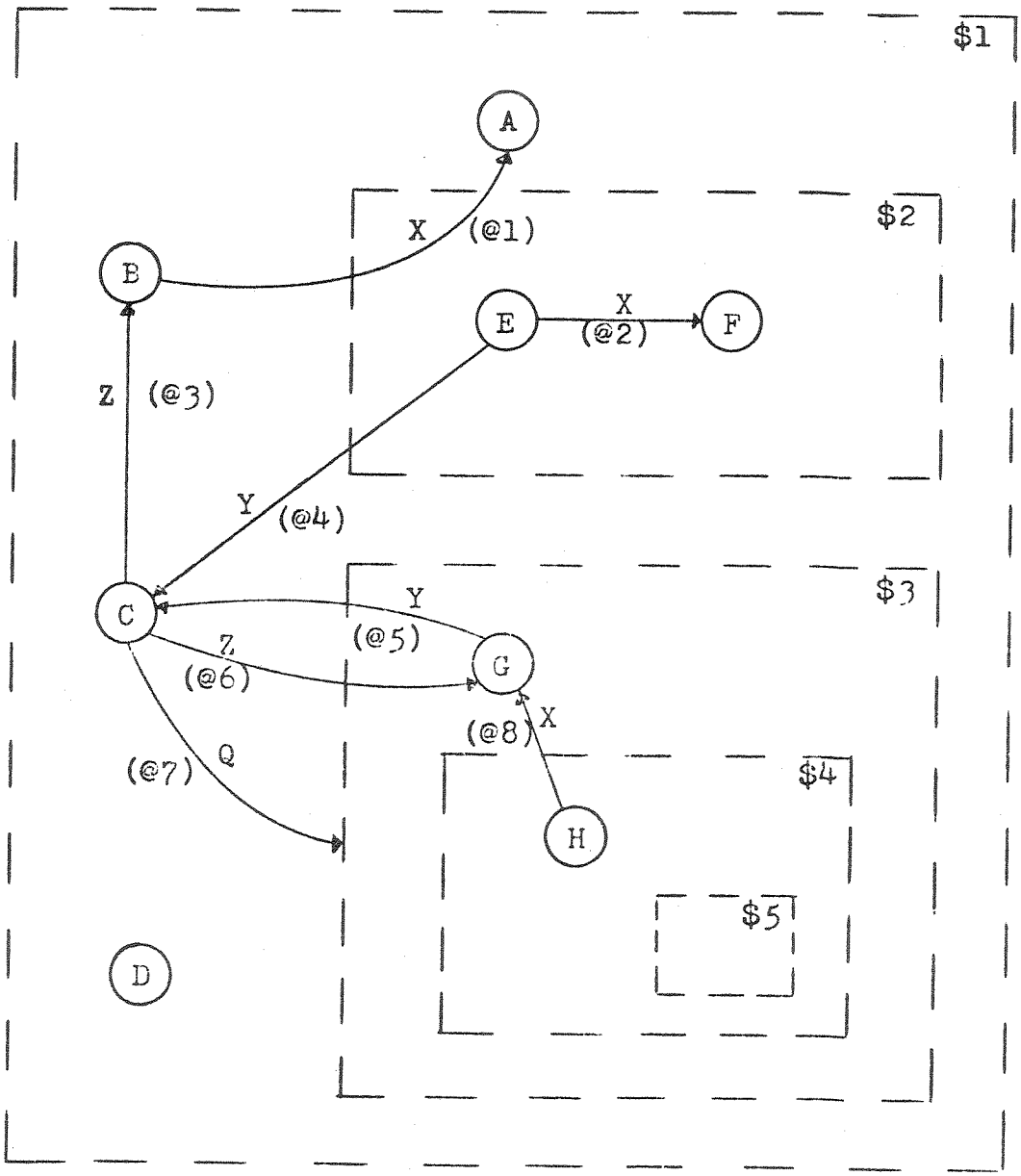
which parses into the structure just described is contained in Walker et al., 1975.

19.9 Implementing net partitioning in LISP

Borrowing from the languages GRASP (Pratt and Friedman, 1971), GROPE (Friedman, 1973; Slocum, 1974) and QLISP (Reboh and Sacerdoti, 1973), this section describes a technique for fabricating and manipulating partitioned networks in terms of the programming language LISP (McCarthy, 1965; Teitelman, 1974). To focus attention on the central mechanisms of this technique, the refinements needed to make it efficient have been suppressed.

In discussing how partitioned nets may be represented by computer data structures, it will be helpful to have some sample net in mind that has a number of illustrative features. Such a net is presented in Figure 19.25. (Symbols of the form "@n" appearing on arcs are for reference purposes only and are not intended as arc labels.)

The principal vehicle used to store information in the technique under discussion is the LISP property list. Spaces, nodes and arcs are all represented by atoms with specially defined properties. As a preview to the discussion, the reader may wish to glance at Figure 19.26 which shows the atoms and associated property lists encoding the sample network of Figure 19.25.



A Sample Network

Figure 19.25

The Spaces	
atom	property list
$1	VISIBLE ($1)
	NODES (A B C D $3)
	ARCS (@7 @6 @4 @3)
$2	VISIBLE ($2 $1)
	NODES (E F)
	ARCS (@2 @1)
$3	VISIBLE ($3 $1)
	NODES (G)
	ARCS (@8 @5)
note	INSPACE $1
	INS (@7)
	OUTS NIL
$4	VISIBLE ($4 $3 $1)
	NODES (H)
	ARCS NIL
$5	VISIBLE ($5 $4 $3 $1)
	NODES NIL
	ARCS NIL

The Nodes	
atom	property list
A	INSPACE $1
	INS (@1)
	OUTS NIL
B	INSPACE $1
	INS (@3)
	OUTS (@1)
C	INSPACE $1
	INS (@5 @4)
	OUTS (@7 @6 @3)
D	INSPACE $1
	INS NIL
	OUTS NIL
E	INSPACE $2
	INS NIL
	OUTS (@4 @2)
F	INSPACE $2
	INS (@2)
	OUTS NIL
G	INSPACE $3
	INS (@8 @6)
	OUTS (@5)
H	INSPACE $4
	INS NIL
	OUTS (@8)

Internal Representation of Sample Network

Figure 19.26

The Arcs					
atom	property list			atom	property list
@1	INSPACE	$1		@5	INSPACE $3
	REL	X			REL Y
	FROM	B			FROM G
	TO	A			TO C
@2	INSPACE	$2		@6	INSPACE $1
	REL	X			REL Z
	FROM	E			FROM C
	TO	F			TO G
@3	INSPACE	$1		@7	INSPACE $1
	REL	Z			REL Q
	FROM	C			FROM C
	TO	B			TO $3
@4	INSPACE	$1		@8	INSPACE $3
	REL	Y			REL X
	FROM	E			FROM H
	TO	C			TO G

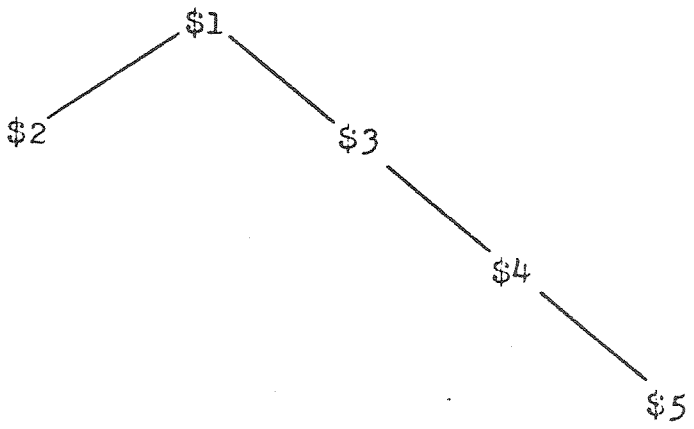
Figure 19.26 (Concluded)

19.9.1 Creating and representing spaces

Each node, each arc and each space is represented as an atom with a property list. The atoms representing net spaces are all formed by concatenating the symbol "$" with a character string representing an integer. The integer n in the space-atom "$n" indicates that the atom corresponds to the n^{th} space generated. As shown in the left column of Figure 19.26, each space-atom has the properties 'VISIBLE, 'NODES and 'ARCS. The value of a space-atom's 'VISIBLE property is a list of spaces which are visible when the network is viewed from the vantage of that space. For example, from space $4 of Figures 19.25 and 19.27, it is possible to see structures in spaces $4, $3 and $1. Hence, the value of the 'VISIBLE property for $4 is the list ($4 $3 $1) as shown in Figure 19.26.

The nodes and arcs appearing in a space $n are listed under the properties 'NODES and 'ARCS respectively. Thus, for example, space $2 has the list (E F) as the value of its 'NODES property and the list (@2 @1) as the value of its 'ARCS property.

The space $1 is automatically created by the initial loading of the network system. (This top space is created by the call (PUT '$1 'VISIBLE '($1)).) The 'VISIBLE property of $1 has value ($1) and the properties 'NODES and 'ARCS both initially have values NIL. As nodes



The Partition Tree of the Sample Network

Figure 19.27

and arcs are created, the values of properties 'NODES and 'ARCS are modified.

Starting from space $1, new spaces are created by the function call

(CRESPACE ANCESTOR)

where ANCESTOR is an existing net space. (The prefix CRE means CREATE.) When CRESPACE is called with an argument $n, it first creates a new space-atom $m where m is an integer corresponding to the number of spaces which have been created. The new atom $m is given the property 'VISIBLE with value ($m $n ... $1), where the value of the 'VISIBLE property of $n is ($n ... $1). The values of $m's properties 'NODES and 'ARCS are (tacitly) "set" to NIL.

(Note: If the partition tree is generalized to a partial ordering, then CRESPACE is called with multiple ancestors and is written as a non-spread function. For the call

(CRESPACE $x $y ... $z)

the value of $m's 'VISIBLE property is

(CONS '$m i)

where i is the intersection of the 'VISIBLEs of $x, $y,..., $z.)

19.9.2 Creating and representing nodes

Once a space has been created, nodes and arcs may

be created within the space. The structure representing a node consists of a unique atomic name (supplied by the user) and a property list consisting of the properties 'INSPACE, 'INS and 'OUTS. The value of 'INSPACE is the space in which the node lies. The values of properties 'INS and 'OUTS are lists of incoming and outgoing arcs to and from the node respectively. As an example of a node structure, consider node C of Figures 19.25 and 19.26. The 'INSPACE of C is space-atom $1. The 'INS is (@5 @4) and the 'OUTS is (@7 @6 @3).

Nodes are created by the function call

(CREND NAME SPACE)

where NAME is the atomic name of the new node and SPACE is the space in which the new node is to lie. The operation of CREND takes place in two parts. First the property 'INSPACE is PUT on the property list of NAME with value SPACE. Properties 'INS and 'OUTS are (tacitly) "set" to NIL at this time. Second, the atom NAME is CONSed onto the value of SPACE's property 'NODES. For example, (CREND 'D '$1) produces the results shown in Figure 19.28 if called in the context of the initial configuration indicated on the left.

(Note: If the NAME argument to function CREND is a space-atom, then node-like properties are added to the property list of the space! For example, the super-node $3

BEFORE		AFTER	
atom	property list	atom	property list
$1	VISIBLE ($1) NODES (C B A) ARCS (@4 @3)	$1	VISIBLE ($1) NODES (D C B A) ARCS (@4 @3)
D	none	D	INSPACE $1 INS NIL OUTS NIL

The Result of Executing (CREND 'D '$1)

Figure 19.28

of Figures 19.25 and 19.26 was produced by the call

(CREND '$3 '$1).

Since the names of space and node properties are mutually exclusive, it is perfectly acceptable for one atom (such as $3) to have both.)

19.9.3 Creating and representing arcs

The atoms representing arcs are all formed by concatenating the symbol "@" with the integer corresponding to the number of arcs generated. Arcs have the properties 'INSPACE, 'REL, 'FROM and 'TO whose values are the space in which the arc lies, the relation type encoded by the arc (the arc label), and the from-node and to-node of the arc respectively.

An arc labeled REL may be created in an existing space SPACE between existing nodes FN and TN by the function call

(CREARC REL FN TN SPACE).

Function CREARC proceeds as follows: First a new arc-atom @m is created as the value of CREARC. The atom is then given a property list which includes the properties 'INSPACE (with value SPACE), 'REL (with value REL), 'FROM (with value FN) and 'TO (with value TN). Further, the atom @m is CONSed onto the value of FN's property 'OUTS, onto the value of TN's property 'INS and onto the value of

SPACE's property 'ARCS. An example of the operation of CREARC is presented in Figure 19.29.

19.9.4 Functions for network interrogation

A number of functions must be defined for retrieving data from the network structures presented above.

To determine the space in which a network ITEM (either a node or an arc) lies, the call

(SPACE ITEM)

may be made. Function SPACE returns the value of ITEM's 'INSPACE property. The S-expression for SPACE is

(SPACE (LAMBDA (ITEM) (GET ITEM 'INSPACE))).

Three functions (FN, TN and REL) may be defined to extract the from-node, to-node and relation-type of an arc. Their definitions are as follows:

(FN ARC) returns the value of ARC's 'FROM property.

(TN ARC) returns the value of ARC's 'TO property.

(REL ARC) returns the value of ARC's 'REL property.

It is sometimes important to know all nodes or arcs lying in a given space SPACE. Functions NODESET1 and ARCSET1 may be used for this purpose. The "1" appearing as the last symbol in these names indicates that only one space is considered. The definitions of these functions are as follows:

(NODESET1 SPACE) returns the value of SPACE's 'NODES property.

BEFORE		AFTER	
atom	property list	atom	property list
$1	VISIBLE ($1)	$1	VISIBLE ($1)
	NODES (D C B A $3)		NODES (D C B A $3)
	ARCS (@3)		ARCS (@4 @3)
$2	VISIBLE ($2 $1)	$2	VISIBLE ($2 $1)
	NODES (F E)		NODES (F E)
	ARCS (@2 @1)		ARCS (@2 @1)
C	INSPACE $1	C	INSPACE $1
	INS NIL		INS (@4)
	OUTS (@3)		OUTS (@3)
E	INSPACE $2	E	INSPACE $2
	INS NIL		INS NIL
	OUTS (@2)		OUTS (@4 @2)
		@4	INSPACE $1
			REL Y
			FROM E
			TO C

The Result of Executing (CREARC 'Y 'E 'C '$1)

Figure 19.29

(ARCSET1 SPACE) returns the value of SPACE's 'ARCS property.

Formally, the S-expression for NODESET1 is

```
(NODESET1 (LAMBDA (SPACE) (GET  SPACE  'NODES))).
```

When it is necessary to know all nodes or arcs which are Visible from a given space SPACE, the functions NODESETV and ARCSETV may be used. NODESETV is defined by the S-expression

```
(NODESETV (LAMBDA (SPACE)
   (MAPAPC (GET  SPACE  'VISIBLE) 'NODESET1)))
```

where MAPAPC is the APPEND mapper defined by

```
(MAPAPC (LAMBDA (MAPX  FUN)
   (COND ((NULL  MAPX) NIL)
         (T (APPEND (APPLY*  FUN (CAR  MAPX))
                    (MAPAPC (CDR  MAPX) FUN))))))).
```

ARCSETV is defined analogously.

Most tasks performed on semantic networks make heavy use of functions which return the set of incoming or outgoing arcs from a node. There are ten such functions.

(SOA1X NODE SPACE) returns the Set of Outgoing Arcs from NODE which lie in the 1 space SPACE. (The "X" in "SOA1X" stands for eXtra argument.)

(SOA1 NODE) returns the set of outgoing arcs from NODE which lie in the same space as NODE itself.

(SOAVX NODE SPACE) returns the set of outgoing arcs from NODE which are Visible from SPACE.

(SOAV NODE) returns (SOAVX NODE (SPACE NODE)).

(SOAG NODE) returns the set of all outgoing arcs from NODE regardless of what space they lie in. The "G" stands for "Global."

The functions SIA1X, SIA1, SIAVX, SIAV and SIAG are the analogous functions for incoming arcs.

S-expressions defining the outgoing-arc functions are as follows:

```
(SOAG (LAMBDA (NODE) (GET NODE 'OUTS)))

(SOA1X (LAMBDA (NODE SPACE)
   (SUBSET* (SOAG NODE) 'EQ 'SPACE SPACE)))

(SOA1 (LAMBDA (NODE) (SOA1X NODE (SPACE NODE))))

(SOAVX (LAMBDA (NODE SPACE)
   (SUBSET* (SOAG NODE) 'MEMB 'SPACE
      (GET SPACE 'VISIBLE))))

(SOAV (LAMBDA (NODE) (SOAVX NODE (SPACE NODE))))
```

where

```
(SUBSET* (LAMBDA (LIST TEST FUN TARG2)
   (COND ((NULL LIST) NIL)
         ((APPLY* TEST (APPLY* FUN (CAR LIST))
                       TARG2)
          (CONS (CAR LIST) (SUBSET* (CDR LIST)
                                    FUN TEST TARG2)))
         (T (SUBSET* (CDR LIST) FUN TEST TARG2))))).
```

The primitive functions presented in this section, along with the creation functions presented earlier, provide a net builder with all the basic tools needed to construct and manipulate partitioned networks.

19.9.5 Efficiency

The technique presented above for encoding and manipulating networks has the virtue of easy readability and was presented primarily to show the feasibility of partitioned-network systems. While the data structures are straightforward and the functions are all very short, the technique is somewhat inefficient. Efficient methods for network systems have been explored by Friedman (1973) and by Slocum (1974). Their GROPE language stores nodes and arcs in fixed-size blocks of memory cells, eliminating many pointers. A more sophisticated method for implementing partitioning may be adapted from the QA4 context mechanism (Rulifson et al., 1972). Following QA4, the 'INS and 'OUTS of nodes are indexed by space. For example, the value of some node's 'OUT property might be the list

((7 @31 @6) (3 @17 @9 @5) (1 @2)),

meaning that the node has outgoing arcs @31 and @6 in space 7, @17, @9 and @5 in space 3 and @2 in space 1. If the 'VISIBLE of a space drops the "$" and uses just numbers (e.g., ($8 $4 $3 $1) becomes (8 4 3 1)) then SOAVX may be redefined as follows.

```
(SOAVX (LAMBDA (NODE  SPACE)
 (PROG (ANSWER  VSBL  SPC  OUTS)
  (SETQ  OUTS (GET  NODE  'OUTS))
  (SETQ  VSBL (GET  SPACE  'VISIBLE))
```

```
  (SETQ  ANSWER  NIL)
LOOP1
  (COND ((NULL  VSBL) (RETURN  ANSWER)))
  (SETQ  SPC (CAR  VSBL))
  (SETQ  VSBL (CDR  VSBL))
LOOP2
  (COND ((NULL  OUTS) (RETURN  ANSWER))
        ((EQ  SPC (CAAR  OUTS))
         (SETQ  ANSWER (APPEND (CDAR  OUTS) ANSWER))
         (SETQ  OUTS (CDR  OUTS))
         (GO LOOP1))
        ((GREATERP  SPC (CAAR  OUTS)) (GO  LOOP1)))
  (SETQ  OUTS (CDR  OUTS))
  (GO  LOOP2))))
```

This version of SOAVX eliminates the need to test the space-acceptability of each arc separately and hence is much faster than the old version for nodes with several outgoing arcs.

Chapter 20

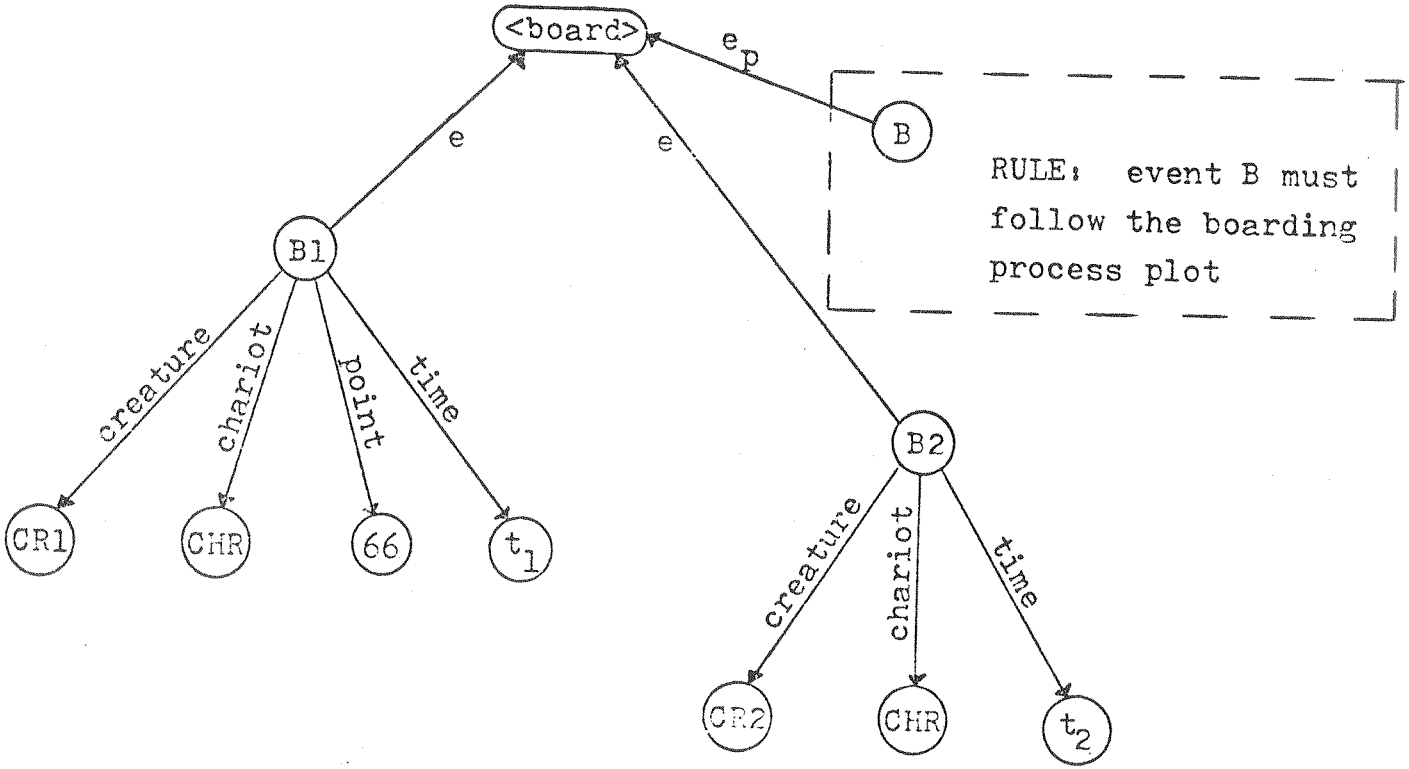
The Network Encoding of Process

This chapter ponders the problem of encoding process knowldege in networks. As discussed in Chapter 10, to encode process knowledge it is necessary to have representations for both process plots and their instantiations. Representing instantiations turns out to be a relatively easy matter. But process plots are complicated collections of knowledge describing the intertwining of changing situations, causes, effects and times. Representations of these plots must be expected to reflect their complex internal structure.

20.1 Relating plots to their instantiations

The relationship between process plots and their instantiations fits neatly into the category system. The set of all instantiations of a particular process constitutes a category whose defining rule, stated in terms of a process plot, indicates that an event belongs to the category iff it follows the plot.

To see how an event category, its plot and instantiations may be encoded in a net, consider the network of Figure 20.01. This net encodes information concerning boarding events in the creatures' world. (In Section



Relationships Between an Event Category, Its Plot and Instantiations

Figure 20.01

10.2.2.1, the plot for this category of events was described by the instantaneous scenario BOARD.) Node '<board>' represents the category of boarding events. Particular instantiations of <board> include B1 and B2. Each boarding event has a #@creature, a #@chariot, a #@point and a #@time. For example, B1 is the boarding of #@chariot CHR by #@creature CR1 at #@point 66 at #@time t_1. B2 represents the boarding of CHR by CR2 at t_2. (The #@point of B2 is unknown.)

Category <board> is defined by a process rule (p-rule). In this figure, this rule is connected to '<board>' through a shorthand e_p-arc (following e_d-arcs, etc.). The structure of this rule (shown in Figure 20.08) will be discussed subsequently.

20.2 Notes on time arcs

Before taking up the encoding of process plots, a few conventions regarding time arcs must be established.

In previous sections, objects have been associated with time through the attributes @st, @et and @time. The #@st of an object is the time at which the object comes into existence. For events and situations, this is the time at which the event or situation is started. The #@et of an object is the time at which the object ceases to exist (or ends). Technically, the #@time of an object

should be the set (i.e., the interval) over which the object has existence. In practice, time-arcs are used to point to particular members of the existence time interval. Thus, in the networks, any #@time of an object is just some time with respect to which the object has contemporary existence.

In general, an object has existence over a half open interval [#@st, #@et). (See Section 6.2.) Any time t taken from this interval is a #@time of the object. By convention, #@et may also be considered to be a #@time. If an object exists for a single instant (a notion of theoretical convenience), then #@st = #@et and the only #@time associated with the object is (by convention) #@st = #@et. Rather than equate [#@st, #@et) with ∅ in such cases, let the "interval" be equated with {#@time}. For instantaneous events such as boardings, it is convenient to record only the event #@time, which is necessarily unique. This #@time is equivalent to the associated #@st and #@et.

(Objects which are not subject to creation or destruction may be assigned @st and @et values of minus and plus infinity respectively. Rather than record this information explicitly for each such object, the objects may be collected into changeless categories. Timelessness may then be expressed in category rules.)

20.3 Encoding a simple process plot

While the discussion of Section 20.1 would most logically be extended by filling in the details of the p-rule for boarding events, it will be more instructive to consider another category of process whose plot may be represented by a simpler net. This other process is the activation of an alarm clock's alarm. The plot for this type of process (whose simulation is discussed at some length in Hendrix, 1973b) may be posed in terms of the following instantaneous scenario.

Scenario name: "ALARM-ACTIVATION"
Parameters: {klock, stime}
Initiation conditions:
{[≡ELEM klock ≡ALARM-CLOCKS],
[≡ELEM stime ≡TIMES],
[≡SET-TIME klock stime],
[≡EQUAL stime $t_{@}$]}
Effects - instantaneous:
delete: {[≡SET-TIME klock stime]}
add: {[≡ALARM-SOUNDING klock]}

The general idea conveyed by this scenario is that there must exist an alarm clock #klock and a time #stime such that #klock has been set to go off at #stime. On a typical alarm clock, [≡SET-TIME klock stime] means that the #klock's alarm hand points to #stime and the alarm ON-OFF button is in the ON position. If the "current time" (one interpretation of $t_{@}$) ever reaches #stime, then the event of alarm activation takes place. The effect of the event

is to terminate the situation of #klock's being set to sound at #stime and to produce the situation in which #klock's alarm is sounding.

The situation of #klock's being (i.e., state of being) set to sound at #stime exists over some interval $[t_1, t_2)$. The situation comes into existence at time t_1 because someone sets the alarm. The situation is terminated at time t_2 when

A) someone sets alarm to a new time (by moving alarm hand)

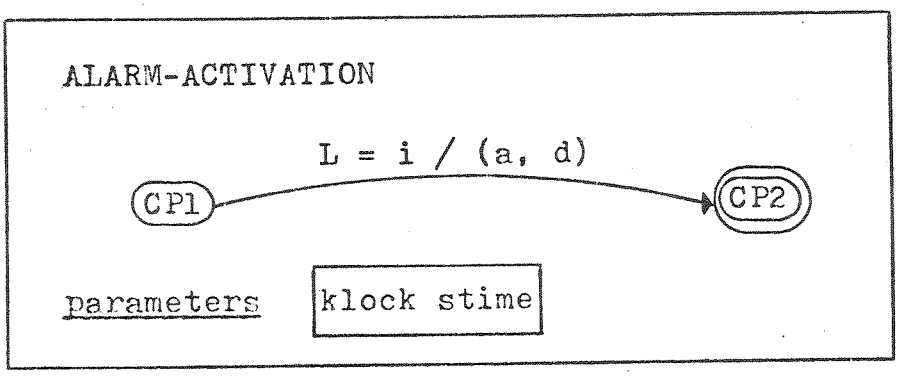
B) alarm is shut off altogether (ON-OFF button to OFF)

C) current, real world time reaches #stime causing the activation of the alarm.

Note the relationship between $[t_1, t_2)$ and #stime. If $t_2 <$ #stime, then termination must have been caused by possibility A or B above. If t_2 = #stime, possibility C applies and an alarm activation event occurs. In no case can #stime be less than t_2.

The process plot for alarm activations is also represented by process automaton PA-AA which is depicted in Figure 20.02. This pa is a direct translation from scenario ALARM-ACTIVATION.

The control structure (transition net) of PA-AA may be encoded in a network as shown in Figure 20.03. The nodes '+CP1+' and '+CP2+' of Figure 20.03 encode the control points +CP1+ and +CP2+ of Figure 20.02. As strange as



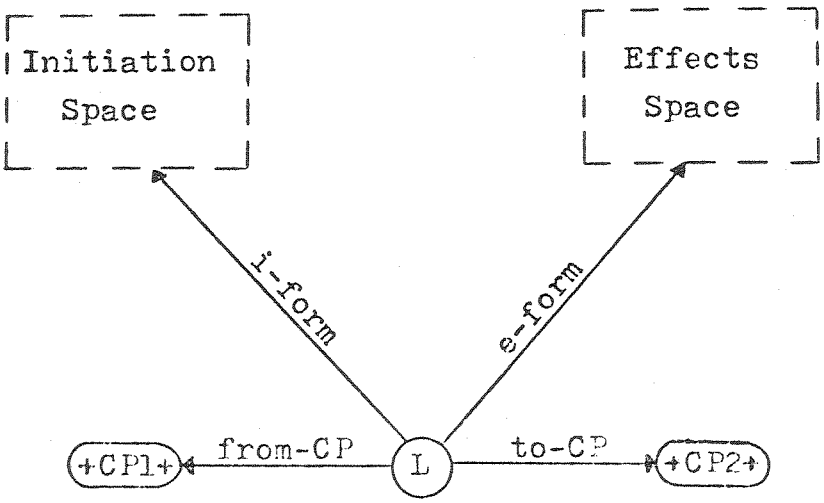
i = {[≡ELEM klock ≡ALARM-CLOCKS],
[≡ELEM stime ≡TIMES],
[≡SET-TIME klock stime],
[≡EQUAL stime $t_{@}$]}

d = {[≡SET-TIME klock stime]}

a = {[≡ALARM-SOUNDING klock]}

PA: Process Automaton for ALARM-ACTIVATION

Figure 20.02



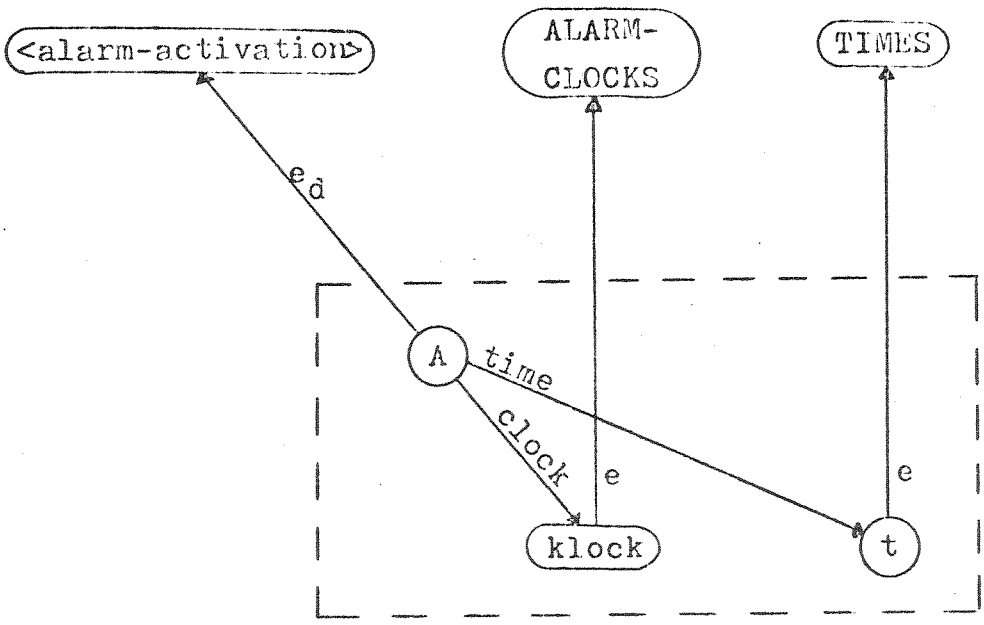
Network Encoding of a PA Transition Net

Figure 20.03

it may seem, the transition arc L of PA-AA's control structure is encoded in Figure 20.03 by the node 'L'. L has a #@from-CP and a #@to-CP indicating which pa control points it connects. In PA-AA, L is associated with certain initiation conditions i and certain effects encoded by the add-delete pair (a, d). In the network of Figure 20.03, L has a #@i-form and a #@e-form. The #@i-form is a net space encoding the initiation conditions which must be met before L may be crossed. The effects of crossing L are encoded in the #@e-form. This #@e-form net space includes both add and delete information.

In terms of network encoding, the parameters of an event (and of its corresponding pa) may be specified by a d-rule. For example, the parameters of an alarm-activation are specified by the d-rule of Figure 20.04. According to this rule, an instance #A of alarm-activation has a #@clock #klock, an element of ALARM-CLOCKS, and a #@time #t, an element of TIMES. Now an alarm-activation takes place in the single instant when the "current time" is the time at which the alarm is set to go off. Hence, #@time specifies both the $\#t_{@}$ and #stime of PA-AA and (implicitly) indicates that they are identical.

The initiation conditions of PA-AA are partially specified by the delineation rule, the rule indicating that #klock is an element of ALARM-CLOCKS and that #t is an



Delineation of <alarm-activation>

Figure 20.04

element of TIMES. Really, these conditions are global to the pa's control structure and were introduced into the i portion of L in Figure 20.02 only for convenience.

The balance of L's initiation conditions are encoded by the network of Figure 20.05. According to this network, there must exist an instance of <alarm-set> in which the #@clock is #klock and the #@set-time is #t. This is equivalent to the template restriction

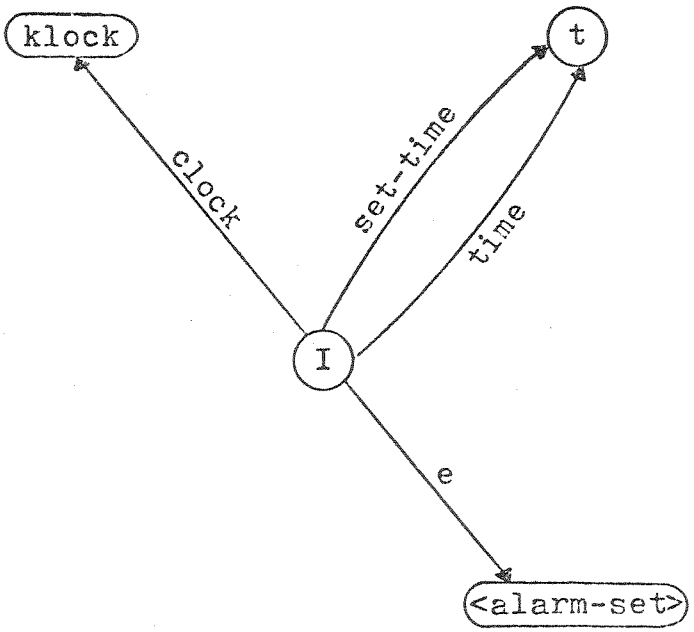
[≡SET-TIME klock stime].

Further, the network indicates (through the time-arc) that the <alarm-set> situation must be in existence at time #t. This carries information equivalent to the template statement

[≡EQUAL stime $t_{@}$]

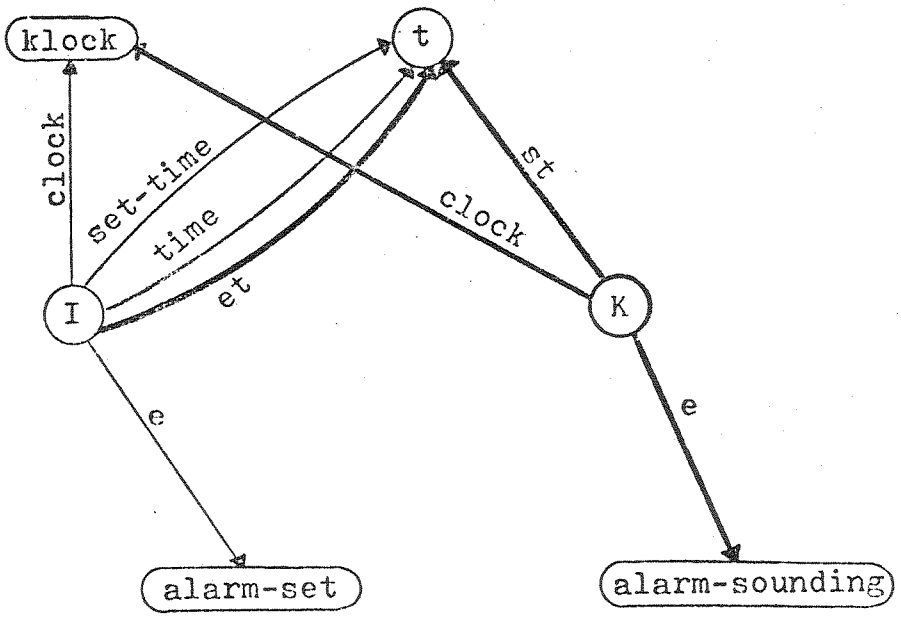
since the time- and set-time-arcs from 'I' point to the same node 't'.

The effects of an alarm-activation are encoded by those portions of the network of Figure 20.06 which are in bold print. In particular, an alarm-activation implies that the <alarm-set> situation #I ends (the et-arc) at #t. Further, an <alarm-sounding> situation #K with #@clock #klock is produced at #t. That is, the #@st of #K is #t. In short, the additions in Figure 20.06 over Figure 20.05 encode the termination of an initiation condition and the birth of a new <alarm-sounding> situation (whose



Initiation Conditions of ALARM-ACTIVATION

Figure 20.05



Effects of ALARM-ACTIVATION

Figure 20.06

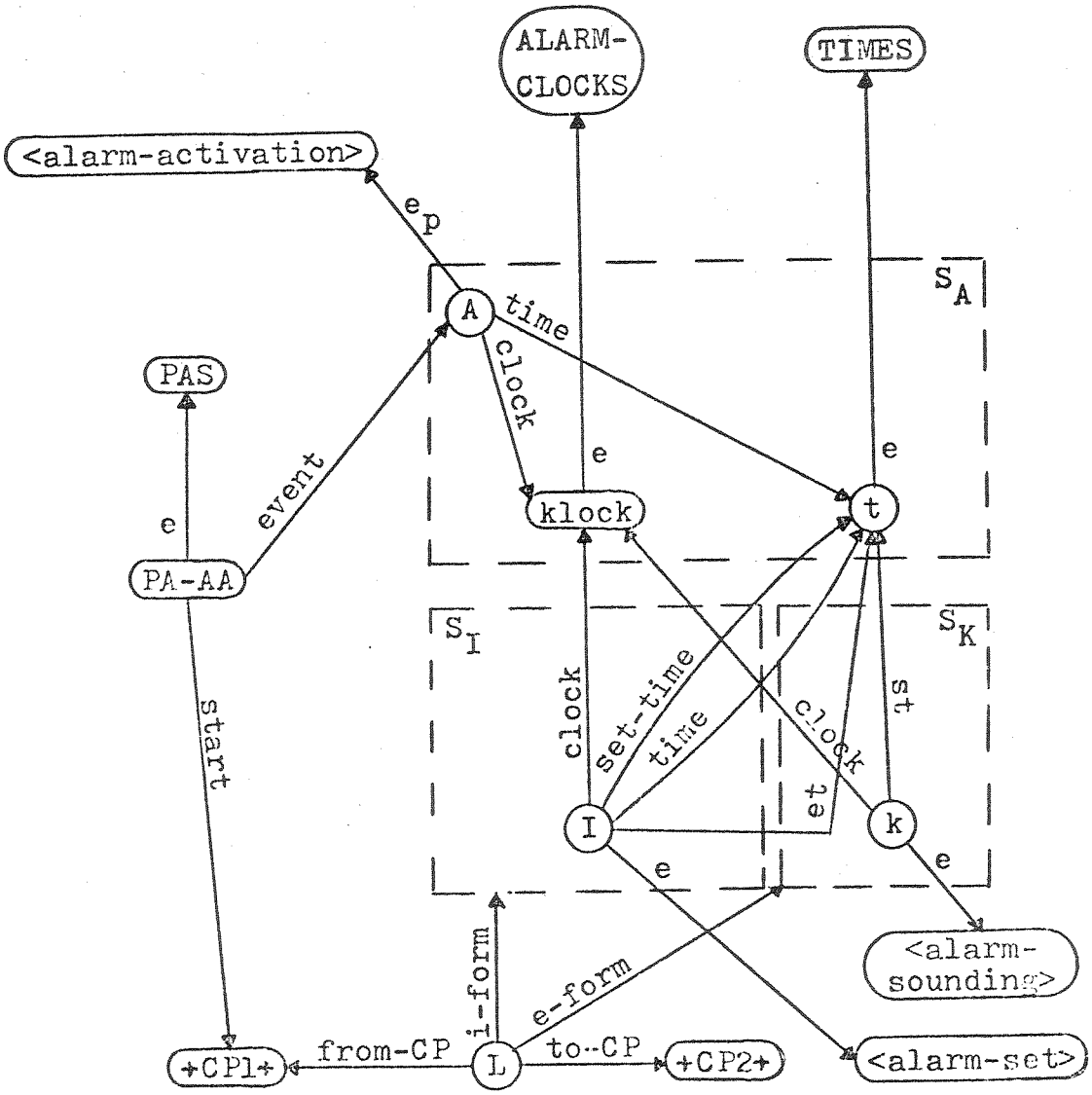
termination time is left unspecified).

The networks of Figures 20.03 through 20.06 may be incorporated into the single net of Figure 20.07 which presents the network encoding of the process plot of <alarm-activation> in its entirety.

To the upper left of Figure 20.07 there appears the familiar node '<alarm-activation>', representing the category of alarm-activation events. This category is defined by a process rule, $RULE_A$, through a shorthand e_p-arc from 'A' to '<alarm-activation>'. The form space in which 'A' lies (space S_A) contains all the information of the delineation rule of Figure 20.04. Indeed, space S_A may be used to determine attribute-range information for instances of <alarm-activation>.

Coming into node 'A' is an event-arc from node 'PA-AA', denoting that A is not only the "archetypal element" of <alarm-activation>, but also the archetype of events following the process plot of process automaton PA-AA. By this dual role, A indicates that members of <alarm-activation> follow PA-AA and that events which follow PA-AA are members of category <alarm-activation>.

Focusing attention now on node 'PA-AA', the e-arc to 'PAS' indicates that PA-AA is a process automaton. As described in Section 10.3, pas have two basic parts: a parameter set and a control structure. The parameter set



Network Encoding of the <alarm-activation> Process Plot

Figure 20.07

of PA-AA is accessible through the event-arc to 'A'. Each attribute of A is an event parameter. Further, A itself may be viewed as a composite, structured parameter which absorbs all other event parameters into its internal composition.

As mentioned earlier, certain global pa constraints may be encoded in the form space of A. For PA-AA, the facts that #klock must be an ALARM-CLOCK and #t a TIME are so recorded.

The control structure of 'PA-AA' is accessible through the start-arc which points to '+CP1+', the representation of PA-AA's starting point +CP1+. The control structure itself is encoded just as in Figure 20.03. But the #@i-form of "arc" L (space S_I) may now be seen to encode the initiation conditions represented by the net of Figure 20.05. Similarly, the #@e-form of L (space S_K) encodes the effect information of Figure 20.06. Note particularly that the et-arc from 'I' to 't' lies in space S_K!

(The reader may very well think that the control structure for PA-AA is both bulky and unnecessary. This is indeed the case for very simple pas with only one transition arc. The control structure is maintained here only for compatibility with more complex pas. If only instantaneous pas are to be used, the control structure may be eliminated and the i-form-arc and e-form-arc may be drawn

directly from 'PA-AA'.)

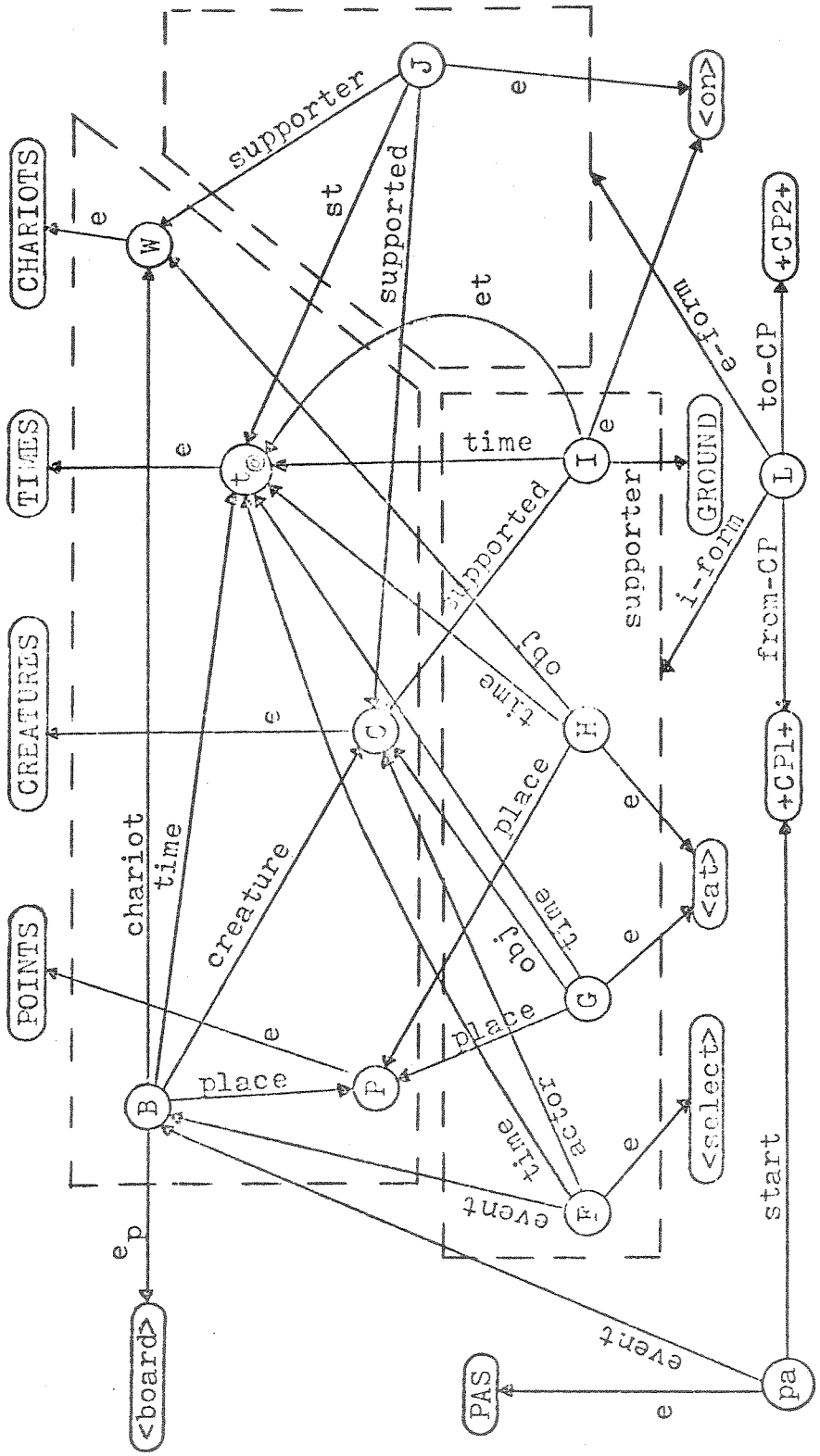
A second example of the network encoding of an instantaneous process plot is provided by the network of Figure 20.08 which encodes the process rule for boarding events. Hopefully, the reader will be able to interpret this network for himself since it is very similar to Figure 20.07. The initiation conditions of <board> are, however, more complex than the initiation conditions for <alarm-activation> in that they include four situations (F, G, H and I) rather than just one. Notice that only one of these initiation conditions (situation I) is directly affected (i.e., terminated) by the process.

20.4 Using the process representation

In the course of its operations, an artificial intelligence may require an understanding of processes for several different purposes. The network encoding of process knowledge exemplified by the nets of Figures 20.07 and 20.08 are manifestations of an attempt to encode process knowledge in an explicit and impartial form suitable for a variety of applications.

20.4.1 Deduction applications

In natural language processing, one of the main applications for process plots is in making deductions from the knowledge that some particular event has occurred.



Network Encoding of the <board> Process Plot

Figure 20.08

Typically, process deductions are used to answer the general question

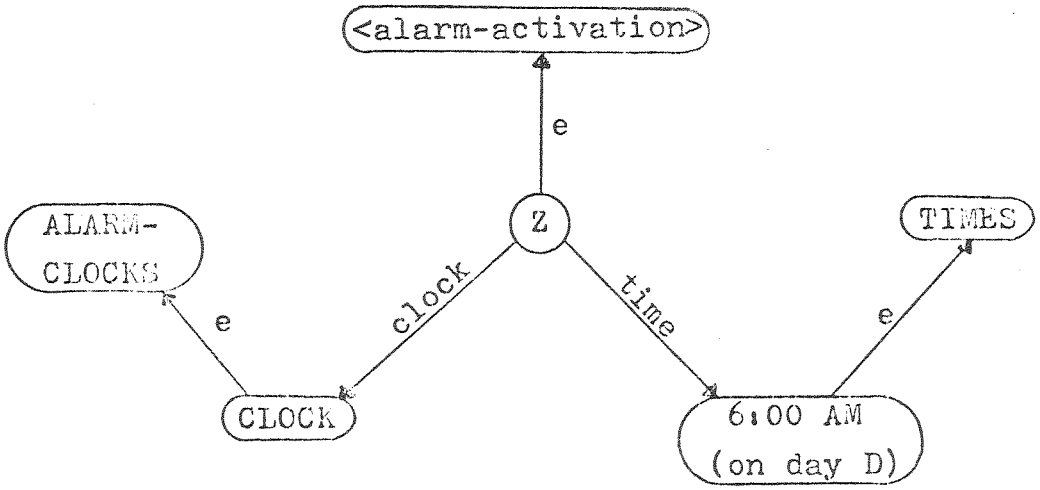
> "In what ways did (will) the occurrence of event Z affect the state of the world?"

For example, the input sentence

> "The clock's alarm went off at 6:00 A.M. (on day d)"

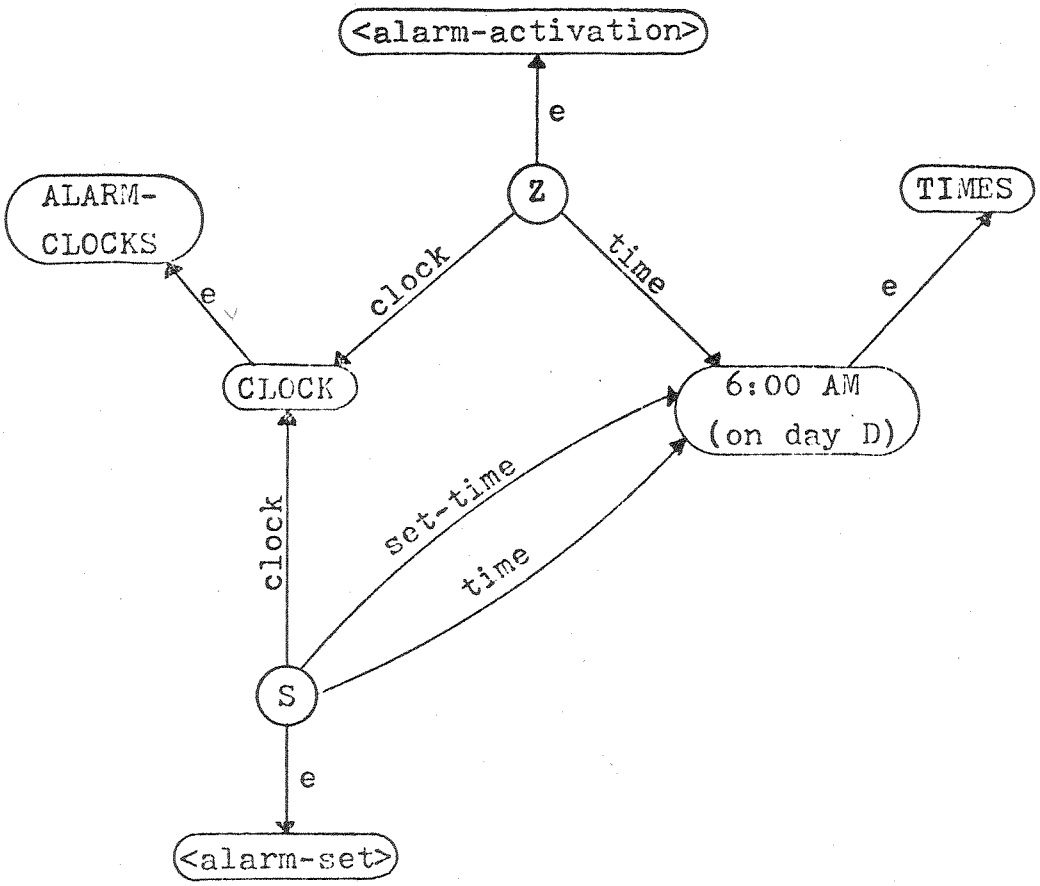
asserts the existence of a particular event Z in which the alarm of a particular clock (say CLOCK) went off at 6:00 A.M. Using the knowledge encoded in Figure 20.07, the event Z may be reconstructed to determine what its existence implies about changes in the state of the world.

The chronological reconstruction of event Z may be used to deduce the sequence of information expressed by Figures 20.09, 20.10 and 20.11. Starting with the ordinary delineation data encoded by space S_A of Figure 20.07, the basic information shown in Figure 20.09 follows immediately from Z's membership in <alarm-activation>. Building upon this base, the control structure of PA-AA may be investigated to determine the dynamics of Z. From PA-AA's control structure it is obvious that the occurrence of Z implies that transition "arc" L must have been crossed. Hence, instantiations of the structure in initiation space S_I must have existed. As shown in Figure 20.10, this implies the existence of an <alarm-set> situation in which CLOCK was



Delineation Deductions for
<alarm-activation> Event Z

Figure 20.09



Adding Initiation Deductions to
<alarm-activation> Event Z

Figure 20.10

set to go off at 6:00 A.M.

Since L must have been crossed, the structure of effect space S_K must also have been instantiated. Thus Z implies that the <alarm-set> situation terminated at 6:00 A.M. Further, Z implies that an <alarm-sounding> situation with #@clock CLOCK came into existence at that same time. These facts are recorded in the composite deduction net of Figure 20.11. (The time-arc from 'S' to '6:00' is omitted in Figure 20.11, being subsumed by the et-arc.)

The information deduced from the existence of event Z finds its function in answering such questions as

"Has the alarm of CLOCK ever been set?"

"When was the alarm set to go off?"

(both readings)

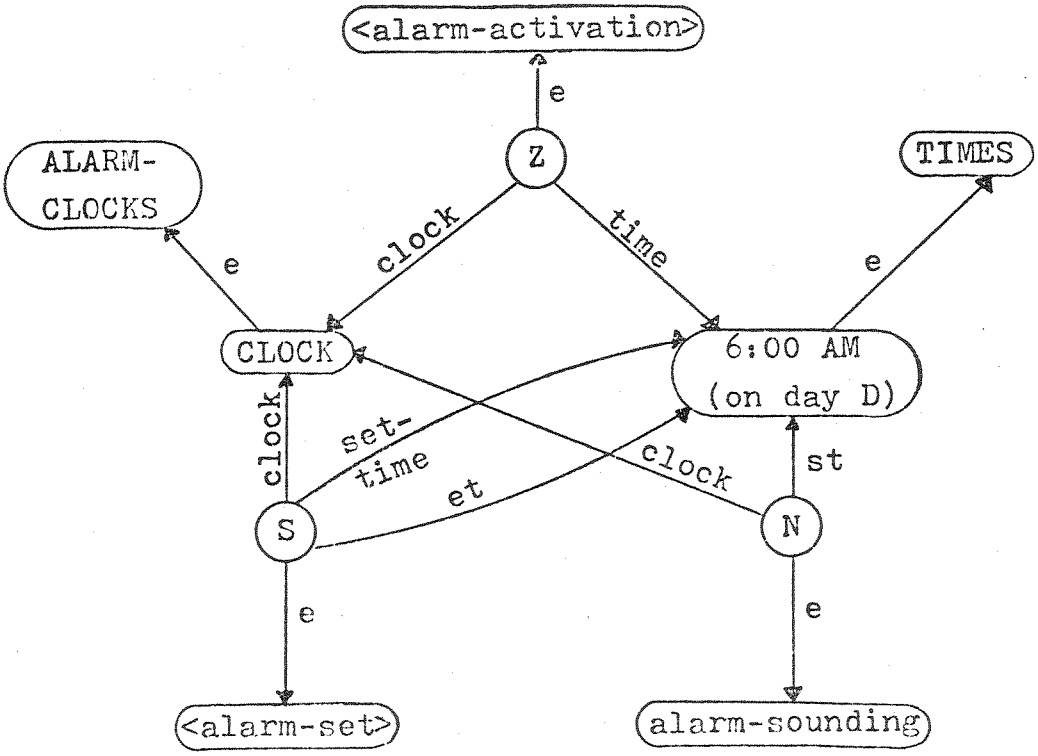
"When did the alarm start sounding?"

Perhaps the utility of process deductions in question answering is better illustrated by appealing to a more complex process category such as the category <exch> of exchange events (including buyings, sellings, tradings, etc.). By methods analogous to those just presented for <alarm-activation> events, from the input sentence

"John bought the hound from Mr. Baskerville"

it is possible to imply that:

There existed a situation in which Mr. Baskerville owned a hound.



Adding Effect Deductions to
<alarm-activation> Event Z

Figure 20.11

Mr. Baskerville's ownership of the hound was terminated by the <exch> event.

As a result of the <exch>, a situation came into existence in which John owned the hound.

For categories of events which are characterized by a collection of subprocesses (events with pas using PUSH arcs), process deductions will indicate which subevents are implied by the macro-events. For example, from

"Cedric called Helen by telephone"

it is possible to deduce that

"Cedric dialed Helen's number on a telephone"

and to answer such questions as

"Did Cedric dial Helen's number before he spoke with Helen?"

20.4.2 Prediction / Simulation applications

Since both humans and intelligent machines typically have a collection of goals (or at least constraints) to be met in the future, an ability to predict forthcoming circumstances is needed so that plans may be constructed to ward off disaster and capitalize upon opportunities. Out of this prophetic need there arises a major genus of questions exemplified by such queries as

"What will become of us?"

and

"What will happen if blah-blah-blah?"

Questions of this latter type are of particular importance in planning since their framing and solution forms the basis for selecting a course of action from among manifold alternatives (and since such questions are needed for planning over multiple contingencies).

An accepted method for making predictions about the future is to perform simulations using a model. (This method finds wide use in both business analysis and engineering.) In Hendrix (1973b) the author has presented a method for performing simulations using SWMs and process scenarios such as those described in Chapter 10. The thing to note here is that the same techniques may be applied to models represented by semantic nets.

As an aid to simulation, the network encoding of process plots carefully divides initiation conditions and effects into separate net spaces. For example, the initiation conditions of transition arc L for <alarm-activation> (Figure 20.07) are in an isolated space S_I while the related effects are in a separate space S_K. A simulation monitor attempts (dynamically) to match the various initiation spaces of (arcs within) process plots. When matches are made, the corresponding effects may be simulated. The effects of simulated processes and the progress of model time lead to new matches as a simulation evolves.

20.4.3 Applications in cause and effect analysis

The network encoding of process plots also provides a facility for cause/effect analysis. The mechanics of how a process produces changes in the world are encoded in the process plot's control structure. All qualitative (discrete) changes of state are modeled by the crossing of control structure transition arcs. The crossing of such arcs requires that certain initiation conditions C be met and results in the production of certain effects E. One interpretation of a transition arc is that it constitutes a cause-effect pair. Under this interpretation, the arc initiation conditions C are the direct cause (within the context of the plot control structure) of the arc's effects E.

An intelligent system may use the cause-effect interpretation of transition arcs in answering such general questions as

1) "Why did V happen?"
2) "Why did W do X?"
3) "How may Y be achieved?"
4) "Why did Z fail to happen?"

As an example of the first of these general questions, consider how

"Why was the alarm sounding?"

might be answered. From Figure 20.07, the sounding of an alarm may be explained through backward reasoning as the

result of a clock's being set to go off at some time t and time t being reached. This is the "state explanation." The "process explanation" is that the alarm is sounding as a result of an <alarm-activation> event.

Questions of the second type involve cause-effect analysis in the forward direction, the general scenario being that W does X because X will cause (directly or indirectly) some Q which W wants.

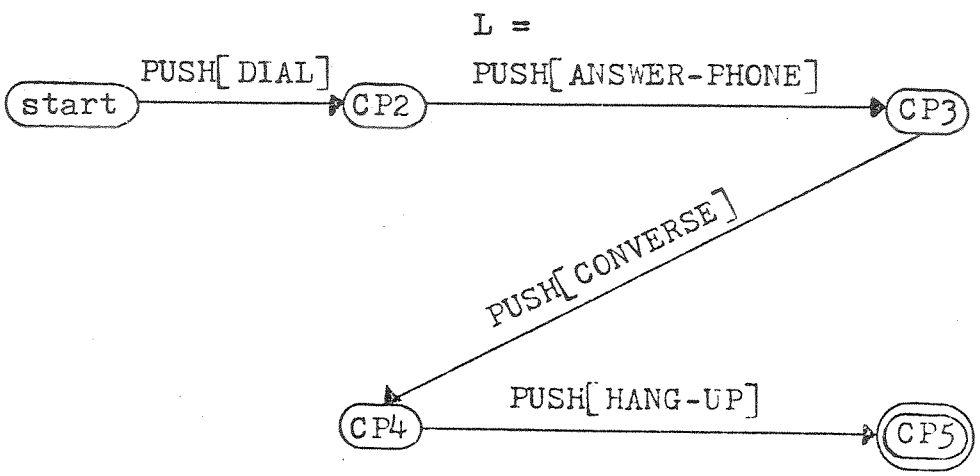
The third question, "How may Y be achieved?" is the planning question. Clearly, all planners are very dependent upon cause-effect analysis. (For determining the longer range effects of various courses of action, a planner may also need to use the simulations discussed earlier.)

The fourth question, determining why some Z failed to happen, is important in understanding "tried to" statements. Consider the statement

"Elmo tried to call Cleo"

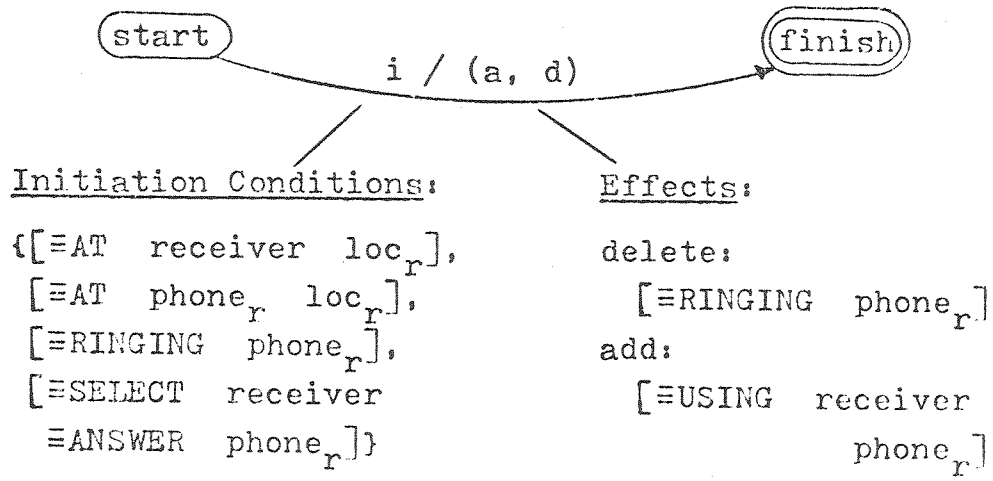
in which Elmo engages in the process of calling Cleo on the telephone, but somehow is frustrated in carrying the process through to completion. To see what may have gone wrong, the <call> process automaton may be examined.

Simplifying the structure and suppressing many details, one formulation of a pa for <call> is presented in Figure 20.12. This pa uses PUSH arcs to indicate that a calling consists of a dialing, an answering, a conversing



Sketch of High Level <call> Process Automaton

Figure 20.12



Sketch of <answer-phone> Process Automaton

Figure 20.13

and a hanging up. Since Elmo tried to call, he must have started through this sequence and met with difficulty along the way.

To see how one of the <call> subprocesses may have failed, consider the PUSH to <answer-phone>, the <answer-phone> pa being given by Figure 20.13. According to its pa, <answer-phone> succeeds only if the pa's sole arc can be crossed. But this requires a receiver (a person to receive the call) to be at the location of the target phone. If there is "no one home," the <answer-phone> process fails producing failure in the <call> process. Nevertheless, Elmo started the <call> process and gets credit for "trying to."

20.5 More complex control structures

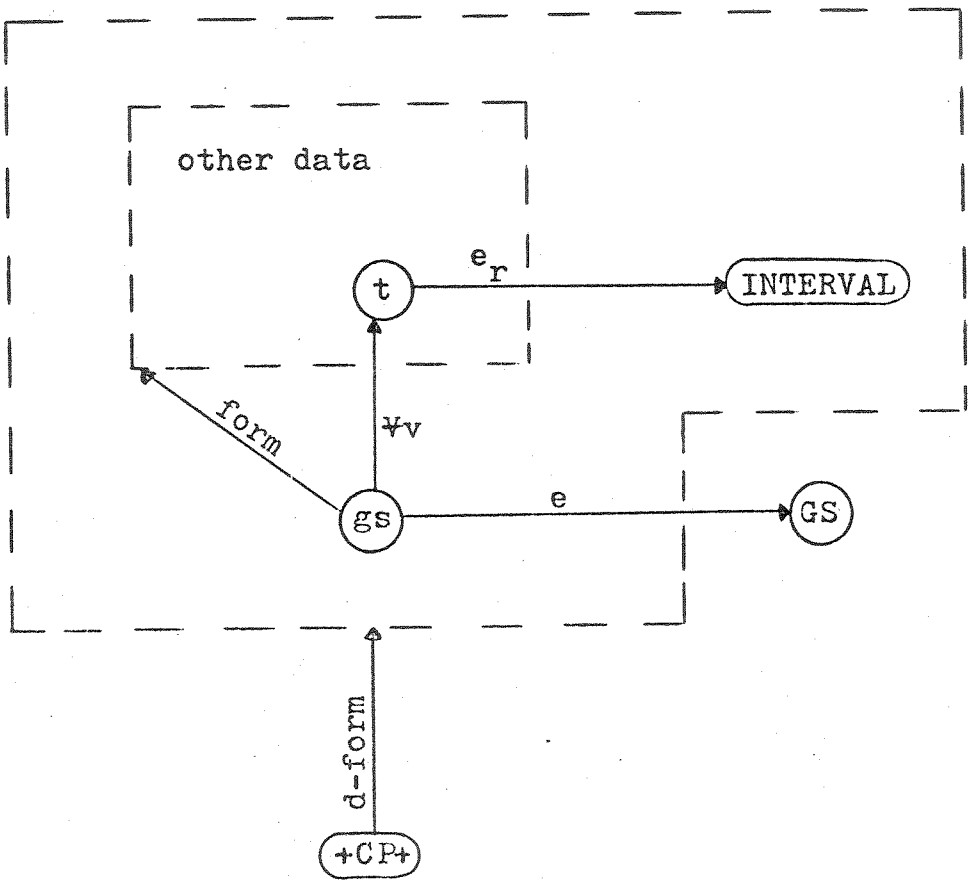
The process automata encoded by the networks of Figures 20.07 and 20.08 are used to model processes of the very simplest type (i.e., instantaneous processes). Nevertheless, they serve to illustrate a general approach to process modeling using networks.

More complex processes will, of course, require the introduction of additional formalisms into the encoding structures. But most of these additions can be imagined here without going into the complexities of their details. To start with, processes with internal options or sequences

of subprocesses are recorded by pas with multiple arcs and control points. Special control points (e.g., types →&, ←& and OR) may be introduced by creating control point categories and using e-arcs to show which category each cp belongs to.

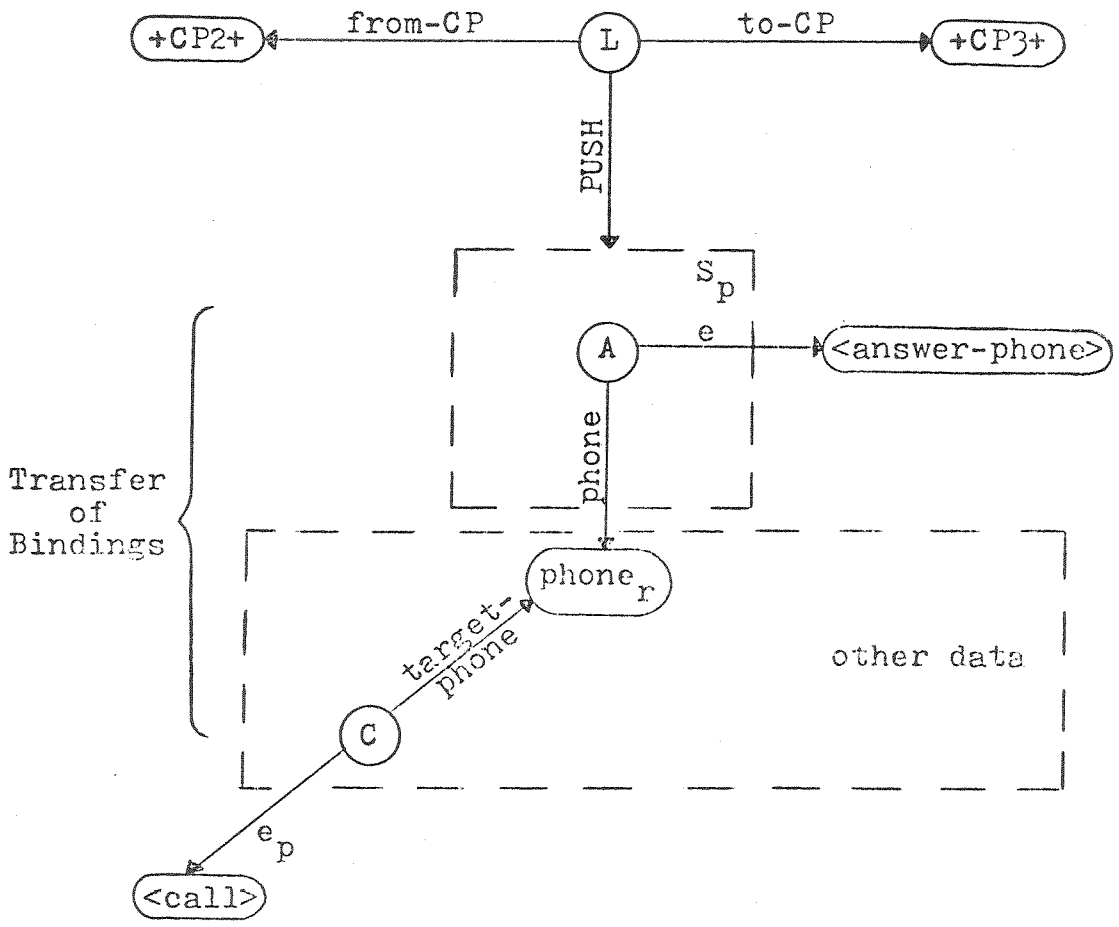
As mentioned in Chapter 10, duration processes such as <walk> are characterized by states of continuous flux. The continuous flux may be encoded by associating general statements with pa control points. The time variables of such a general statement are allowed to range over all instants in the duration over which process control is resident at the associated control point. Abstractly, this situation is encoded by the network of Figure 20.14. Node '+CP+' represents a control point at which control may be resident for some interval. The d-form of +CP+ is a space containing the time interval 'INTERVAL' and general statements ranging over INTERVAL.

PUSH arcs also require special formalisms. The PUSH arc L of Figure 20.12 which connects +CP2+ to +CP3+ and PUSHes to an <answer-phone> event is encoded in Figure 20.15. 'L' is linked by a PUSH-arc to a special net space, S_p. S_p is very much like a subroutine call (or, in some cases, multiple-parallel calls). To cross L, the structure of S_p must be instantiated. Thus, there must exist a situation #A where #A is an <answer-phone> event whose #@phone



The State of Continuous Flux
Associated with a Control Point

Figure 20.14



Formalism for PUSH Arcs

Figure 20.15

is #phone$_r$. This #phone$_r$ is the same phone as the #@target-phone of the parent <call> event.

Chapter 21

Concluding Remarks

21.1 The research goal

The research goal of the work reported here has been to develop a modeling scheme capable of encoding the multiple aspects of world knowledge through a single uniform paradigm that is precise, easily manipulatable and amenable to use by digital computers.

21.2 The principal components of the devised modeling scheme

In seeking to achieve the research goal, the problems of how to characterize various aspects of nature, how to efficiently organize bodies of related pieces of information and how to conveniently represent information in a digital computer have been studied in concert, the approach to each problem being constrained by aspects of the others.

In keeping with the goal of developing a uniform paradigm, the notion of the all encompassing object has been adopted as the fundamental building block of the modeling scheme. Virtually anything may be classified as an object, including such (seemingly) diverse things as physical objects, situations, relations, events, process definitions, sets, categories, time intervals, rules,

quantified expressions, colors, ideas and languages. In terms of computer data structures, any object may be represented by a node in a semantic network (although it is convenient to represent some fundamental objects as arcs).

Certain types of objects (notably situations, relations and events) are gestalts of the interconnections among sets of objects. To better understand the nature of these composite objects, the differences between the informal notion of a relationship which changes with time and the mathematical notion of a relationship as an invariant set of n-tuples has been studied. Rather than regard the interconnections between objects as being either informal or mathematical relations, it has been convenient to think always in terms of situation objects. Unlike informal (lay) relationships which vacillate through time or mathematical relations which are timeless, each situation has existence over a single time interval. From outside this interval the situation is not regarded as being false, but rather as being a true part of the past or future.

An effort has been made to integrate the concepts of time and change into the modeling system in a consistent and realistic way. Time receives its proper treatment as a continuous phenomenon being modeled by a dense, ordered set of objects. Change is thought of as a transfer of focus from one set of objects which have existence at some time

t_1 to another set of objects which have existence at another time t_2. (Modern physics theories were not considered.)

To characterize frequently reoccurring patterns of change, the notion of process has been developed. An individual event is regarded as a concrete instantiation of some generalized sequence of changes defining a process type. Each process type is described by a process automaton which resembles both a Woods' AFSTN-type network and a STRIPS operator. A process automaton is composed of a set of control points interconnected by transition arcs. Each such arc encodes a set of initiation conditions and a set of effects. When the initiation conditions of an arc leaving an active control point are met, the arc is CAUSED to be taken and its discrete EFFECTs produced. While process control is resident in a control point, continuous effects may be in progress. When fully utilized, process automata may describe simultaneous and continuous processes, invoke subprocesses and encode sequences of change at varying levels of detail. Such automata may describe both ordinary events and such computer processes as natural language parsing.

To increase modeling efficiency, similar objects are grouped into categories so that they may be thought about and talked about collectively. The categories are organized into a hierarchical taxonomy allowing individuals

and subcategories to inherit the properties of supercategories. Each category is tantamount to a set-object and each link in the hierarchy to a subset or membership situation. Of significance to natural language processing is the fact that nouns and verbs generally serve as pointers to categories and that rules describing the nature of category members may be used to help parse input sentences.

One of the most interesting types of objects with which the modeling system is capable of dealing is the hypothetical world. Hypothetical worlds are used to model such things as dreaming and wishing events, to encode quantified expressions, and to facilitate planning and abstraction.

The uniform recording medium in which all modeling information is encoded for use by computer is the partitioned semantic network. With its backtrackable linkage, the network provides a fully inverted memory store making it easy to locate all (explicitly encoded) objects which are directly related to any given object. The partitioning mechanism allows attention to be focused on selected subsets of the total network and allows conflicting, alternative worlds to be constructed without confusion. Further, the partitioning facilitates the encoding of generalized, quantified expressions.

21.3 New tools for modeling

The search for a uniform scheme of representation has lead to the development of two new tools for modeling, the partitioned semantic network and the process automaton, which have potential applications outside the realm of natural language semantics systems.

In a nutshell, partitioning provides in semantic networks what parentheses provide in predicate calculus notation. The partitioning is not encoded in terms of constructs built up from conventional nodes and arcs, but rather is implemented through net spaces, a new type of primitive entity on a par with nodes and arcs. By imposing a partial ordering on net spaces, a generalization of the context mechanisms of languages such as QLISP is produced.

In a semantics system, net spaces may be used to delimit the scopes of quantified variables, encode logical connectives such as disjunction, implication and negation, distinguish hypothetical and imaginary situations from reality, encode alternative worlds for consideration during planning, focus attention at a particular level of detail or particular region of the semantic net, and encode intentional definitions of categories.

An important discovery concerning net spaces is that they may be used to build structures which simultaneously capture both the syntax and semantic network

translation of a natural language utterance. Since this structure (see Section 19.8) also encodes the semantic network translation of each syntactic subcomponent, it appears well suited for discourse analysis, especially for the resolution of ellipsis.

Process automata are structures for describing both continuous and discrete processes with (possibly) multiple internal states or stages. These structures combine features from simple automata, STRIPS operators and Woods' AFSTN system. Unlike the passive control points of other automata, the control points of process automata may encode continuous states of flux such as the position of a raft floating downstream.

The structure of a process automaton resembles a network grammar for parsing sentences. But a process automaton describes possible sequences of events and subprocesses rather than possible sequences of words and subphrases. Thus, a process automaton may be used to describe a class of events as opposed to a class of strings (a language). In the "parsing" mode, a process automaton determines whether or not a particular sequence of changes falls within a particular class of events. In the "generative" mode, a process automaton produces an event of a given class.

21.4 Chronology of computer program implementations

It is a fundamental tenet of computer science that any theory concerning computation or data organization is incomplete until refined and tempered by experiments involving the construction of running computer programs. Following this tenet, in 1972, the author in collaboration with Jonathan Slocum and Craig Thompson built a natural language question answering system based on what are now seen to be primitive notions of semantic networks, canonical verbs and event procedures (which evolved into process automata). The construction of this system (reported in Hendrix, Thompson and Slocum, 1973) proved to be very useful to its builders both for its positive and its negative results. On the positive side, the program demonstrated the ability to parse into and generate from network structures based on canonical verbs. Further, it demonstrated the ability to perform question answering tasks using algorithms based on net structure matching, and it proved the feasibility of introducing the dynamics of robot-like world modeling into the previously static medium of networks. On the negative side, it showed the original concept of semantic nets to be wanting in a number of ways. In particular, it became clear that there was needed an ability to encode general statements as well as specific data, to organize the data hierarchically, and to represent process

definitions by networks so that questions about such processes could be answered in the same way as other queries. The original conceptions of time and process were also shown to be lacking. Time needed a continuous representation and provisions were needed to model multiple processes proceeding simultaneously. A final piece of negative information was that even this simple system was pushing the space resources of the University of Texas computer and it would not be possible to build more ambitious systems until either virtual LISP or virtual GROPE was available.

Following the experience with the 1972 system, for a time the question of how to represent process knowledge was the principal focus of research. Working from Woods' AFSTN system and from STRIPS operators, the notion of process automata was developed. Since the AFSTN system itself serves as a demonstration of the feasibility of describing discrete process by networks, it was decided to concentrate programming effort on producing a system capable of maintaining models of simultaneous and continuous processes. This effort produced the successful simulation system described in Hendrix (1973b) which builds descriptions of processes from discrete scenarios (based on two control point pas) and simple duration scenarios (based on three control point pas).

Only after the author joined the SRI speech project

in mid 1974 did the support facilities necessary to construct a large system embodying many of the revised notions of semantic networks become available. Using these more powerful facilities, a semantic system employing partitioned semantic networks was constructed. The system makes full use of the hierarchical taxonomy, the delineation rules (see Sections 11.3 and 18.4) and the net space partitioning of parses (Section 19.8). The parse partitioning scheme was, in fact, turned up as a direct result of the speech programming effort.

The programming of a question answerer which is capable of understanding the network encoding of quantified statements and processes has recently begun. Early results of this effort should be available in the winter of 1975.

21.5 Topics for future research

The top priority project for future research efforts is the development of a question answering system which, when given a network description of a user's question, will automatically devise and execute a strategy for retrieving the requested information from a semantic network encoding the system's knowledge base. Work on this project is currently in progress.

One of the most exciting features of partitioned networks is the ability to encode information about

processes and other objects at various levels of detail. Given an input query and some knowledge of the system user, an important research goal is to devise a method for determining the appropriate level at which to interrogate the data base.

By virtue of suppressing unnecessary detail, it should prove to be true that the higher the abstraction level at which a given question can be answered, the less work required. Abstraction might also be used to aid question answering by following the strategy of Sacerdoti's ABSTRIPS planner (Sacerdoti, 1974). Following this strategy, complex questions would be answered by first producing an answer at a high level of abstraction and then refining the answer at successively lower levels, using the abstracted answer as a guide.

Yet another area for research in abstraction concerns the generation of text. When writing about a particular subject area, an author typically first considers his information at a high level of abstraction, dividing it into a few chapters. Then each chapter is considered at finer and finer levels of detail, with information being broken up into sections, then into paragraphs and finally into individual sentences. The introductory chapter of a book is often an abstraction (overview) of the book's contents. At the opposite extreme, the first sentence of a

paragraph is often an abstract summary of the paragraph's contents. Thus a study of the abstraction techniques employed by the users of language would undoubtedly aid in the generation of extended text from nets.

A thorn in the flesh of virtually all natural language semantic systems is the mass noun. While some progress has been made in this area (for example, the model of a bucket filling process discussed in Hendrix, 1973b) the proper treatment of mass remains an open question. While waiting for a general solution to present itself, masses such as "a gallon of gasoline" are currently encoded by nodes with a mass-subset-arc to the substance in question (e.g., to 'GASOLINE') and a measure-arc to the associated quantity (e.g., to '1-GALLON').

Since process automata have not yet been incorporated into a running semantic system, much work remains in this area. To answer "how to" questions, a planner capable of understanding pas must be constructed. A planner capable of dealing with the more complex pas would, of course, find it necessary to plan for simultaneous actions and continuous processes. Such a planner with its more realistic treatment of time would be a welcome addition to AI technology.

While process automata were developed to describe the meaning of verbs such as "walk" or "buy" or "read," it

seems clear that automata may also describe nouns such as "football game," "lecture" and "picnic" (as well as the noun-like gerunds "walking" and "buying") since these nouns are the names of event types. But even ordinary physical objects have an event-like nature. The life of a baseball, for example, consists of being made, sold, thrown, caught and struck with bats. The use of process automata to describe the event-like characteristics of physical objects has not received much study and deserves to be explored further.

In addition to encoding definitions of verbs, process automata may encode specific and detailed information concerning procedures for performing complex tasks. Such an automaton may be used to give advice about how the task should be performed and to monitor the progress of the performance of the task by a human being. This, in a nutshell, is the conception of the "computer based consultant." Such a consultant is now being developed at SRI (Nilsson, 1975) with the current system using procedural nets (Sacerdoti, 1975) in the place of process automata. (Since procedural nets were not designed as a part of a language system, linguistic links will need to be added to interface these structures with a natural language component.)

The development of an interpreter for executing process automata should be a fairly trivial task, given

Woods' ATSTN system to build on. Having such an interpreter would allow the natural language translation processes and other tasks performed by the system to be encoded as pas. Encoded in this form, the system would have a certain degree of self-awareness, being able to describe its own internal workings (at some level of abstraction - see Section 12.4). Further, when a heretofore unknown process is described to the system, the system should be able to build up a pa to encode the concept of the new process. If the process concerns something which a computer is capable of doing, then the newly learned process could be executed.

Fuzzy concepts such as what it means to be "very tall" or "moderately rich" have been studied by Zadeh (1974a, 1974b), but have not been incorporated into network representations in any but the most trivial ways. An interesting project would be to build a process automaton describing a pa-test rule (Section 12.3) for a category such as the set of "successful" people. This pa would encode a procedure for performing a fuzzy computation to determine if a given person is "successful" given certain information such as how rich, how well-liked and how well-known the person is.

Deutsch (see the language section of Nilsson, 1975) has suggested that a semantic network may be partitioned in multiple ways. In addition to the partitioning described

herein which is used for the encoding of quantification and other logical structures, a second partitioning may be imposed upon a network to divide it into regions dealing with particular semantic domains. If information concerning the assembly of jeep engines is grouped in one space, then discourse routines may restrict their activities to this area when processing inputs dealing solely with this topic. This use of a secondary partitioning is currently undergoing active research.

"What my net can't catch isn't fish."

- Sir Arthur Eddington

GLOSSARY

AT: Usually used as a formal relation set of pairs (x, p) where object x is at place p (in the appropriate context of time). (Section 6.2)

AT*: A set of triples (t, x, p) indicating that object x is at location p at time t. (Section 6.3)

AT:** A set of quadtuples (t_1, t_2, x, p) indicating that object x was at location p over the interval $[t_1, t_2)$. (Section 6.3)

<at>: The set of at situations in which a #@occupant is at a #@place from time #@st until time #@et. (Section 16.4)

Binding set: A set of symbol-value pairs (s, v) used to establish contexts in which the bindings of the symbols s are the corresponding values v. (Section 9.1)

C: A function which maps a binding set G to a context q. In context q, S-expressions will evaluate to their usual LISP values where the LISP a-list reflects the bindings of set G. (Section 9.1)

Chronofluxion: Any change in the validity of lay relationships caused by changing the focus of attention from one time to another. (Section 6.5)

Chronorelation: A lay relationship which assumes only a given time. For example, "John is at home" assumes "at a given time." If John has more than one home, then more than a given time is assumed and the relationship ceases to be a chronorelation. (Section 6.2)

Cluster: A construct resembling the n-tuple, but allowing multiple components in each of its n positions. For example, the cluster (a; b, c; c) has two second components, b and c. (Section 5.5)

DPS: Set of duration process scenarios. (Section 10.2.2.2.1)

e_d: Like e_{ns}, but for delineation rules. (Section 18.3.2)

e_{ns}: Label of pointer to archetypal element defining a necessary and sufficient rule. (Section 18.3.2)

form: An arc label used to associate certain entities (notably general statements and wanting situations) with constructs denoting the structure of a hypothetical world. The #@form of a general statement may be thought of as the scope of the universal quantifiers. (Section 17.2)

G: Normally used to denote a binding set. (Section 9.1)

GS: The set of general statements. (Section 17.2)

H_c: $((x_1, x_2, \ldots, x_n), \underline{r}) \in H_c$ iff relation $\underline{r}$ (either lay or formal) holds over $(x_1, x_2, \ldots, x_n)$ in context c. (Section 5.3)

H^h: $(Z_1, Z_2, \ldots, Z_n) \in H^h$ iff $Z_1, Z_2, \ldots, Z_n$ obey rule h. (Section 11.1)

I: Normally used to denote a time interval $[t_1, t_2)$. (Section 9.3)

$\underline{I}$: A map from instants in time to isochronons. $\underline{I}(t)$ is a context whose only presupposition fixes time at t. (Section 6.5)

IIPS: Set of instantiated instantaneous process scenarios. (Section 10.2.2.1.1)

IPS: Set of instantaneous process scenarios. (Section 10.2.2.1.1)

Isochronon: A context whose only presupposition is a fixed time. (Section 6.5)

J: $(c, r, R_c) \in J$ iff R_c is a formal relation encoding lay relation r in context c. (Section 5.3)

Macro-state: The macro-state M of a context c is the set of all relational statements which hold in c. (Section 6.5)

OWNS, OWNS*, OWNS** and <own>: See corresponding entries for AT.

pa: A process automaton.

Process Automaton: A formalism for describing process plots which resembles the AFSTN system of Woods. (Section 10.3)

RP: The repeating pattern relation set. Similar to $\overline{\forall}$, but uses clusters to encode both the universal and existential quantification. (Section 9.2.3.3)

$RULE_x$: The one-place rule using variable x. (Section 18.3.2)

Scenario: A formalism for describing process plots which resembles the "operators" of STRIPS. (Section 10.2.2)

T: Normally used to denote the set of all instants of time. (Section 3.2)

Template: An S-expression used as a complex symbol to denote an n-tuple. The structure of the denoted n-tuple resembles the structure of the symbolizing S-expression. (Section 7.2)

U: The universal set of objects. (Section 4.1)

U*: The set of pairs (t, x) taken from T X U such that object x has contemporary existence at time t. (Section 4.2)

V_q: $(s, v) \in V_q$ iff symbol s denotes value v in context q. V_q is the value relation. (Section 7.1)

$\underline{value}_q$: A map from a symbol s to the value of symbol s in context q. $\underline{value}_q(s) = v$ iff $(s, v) \in V_q$. (Section 7.1)

: (1) Used to denote the time invariant relation R derived from a time variant relationship R. (Section 6.3)
(2) A symbol such that $\underline{value}_{\underline{C}(G)}(*) = G$. (Section 9.1)

: Used to denote a formal relation over a time interval. See AT.

@: Used to denote attributes. Hence, @x is the x attribute. Symbols of the form @<integer> are used as LISP atoms to refer to arcs.

$\overline{\forall}$: A relation set used to encode universal quantification.

$(x, X, G, d) \in \overline{\forall}$ iff
$\forall y \in X, \underline{value}_{\underline{C}(G \cup \{(x, y)\})}(d)$
where G is a binding set and d is a template. (Section 9.2.3.2)

$\overline{\forall}$v:	An arc label used to associate a general statement with its universally quantified variables. (Section 17.2)
$\overline{\exists}$:	A relation set paralleling $\overline{\forall}$ which is used to encode existential quantification. (Section 9.2.3.2)
$\overline{\epsilon}_q$:	The "element in context" relation. s $\overline{\epsilon}_q$ R iff $\underline{\text{value}}_q(s) \in R$. (Section 9.2.2)
#:	Used to denote "value of". Hence, #x is value of x and #@x is value of attribute x or value of the x attribute.
'<string>':	The node with label <string>. (Section 15.1)
<string>-arc:	An arc with label <string>. (Section 15.1)
+<string>+:	The automaton control point with label <string>. (Section 10.3)

BIBLIOGRAPHY

Abelson, R. P. (1973) "The Structure of Belief Systems," in Schank and Colby (eds.), Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Abelson, R. P. and J. D. Carroll (1965) "Computer Simulation of Individual Belief Systems," Amer. Behav. Sci. 8, 24-30.

Adams, J. A. (1967) Human Memory. Mc Graw-Hill, New York.

Amarel, S. (1968) "On Representations of Problems of Reasoning About Actions," in Michie (ed.), Machine Intelligence 3. Edinburgh University Press, Edinburgh.

Anderson, J. R. and G. H. Bower (1973) Human Associative Memory. V. H. Winston and Sons, Washington, D. C.

Baron, R. J., D. P. Friedman, L. G. Shapiro and J. Slocum (1973) "Graph Processing Using GROPE/360," Tech. Report 73-13, Dept. of Computer Science, University of Iowa, Iowa City.

Bledsoe, W. W. and P. Bruel (1973) "A Man-Machine Theorem Proving System," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 56-65.

Bernays, P. (1937) "A System of Axiomatic Set-Theory, Part I," Journal of Symbolic Logic 2, 65-77.

Bobrow, D. (1968) "Natural Language Input for a Computer Program Solving System," in Minsky (ed.), Semantic Information Processing. MIT Press, Cambridge, Ma.

Boole, G. (1847) The Mathematical Analysis of Logic. London and Cambridge, England.

Brown, J. S., A. G. Bell and F. Zdybel (1973) "Steps Toward an Artificially Intelligent CAI System: An Interim Report on Research in Progress," BBN Report No. 2970, Bolt, Beranek and Newman, Cambridge, Ma.

Brown, J. S., R. R. Burton and A. G. Bell (1974) "SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting," BBN Report No. 2790, Bolt, Beranek and Newman, Cambridge, Ma.

Brown, J. S., R. R. Burton and F. Zdybel (1972) "A Model Driven Question Answering System for a CAI Environment," Tech. Report No. 13, University of California, Irvine.

Bruce, B. C. (1972) "A Model for Temporal References and its Application in a Question Answering Program," Artificial Intelligence 3, 1-26.

Bruce, B. C. (1973) "Case Structure Systems," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 364-371.

Carbonell, J. R. and A. M. Collins (1973) "Natural Semantics in Artificial Intelligence," Proc. of the Third Int. Joint Conf. on A. I., Stanford, Ca., 344-351.

Charniak, E. C. (1972) "Toward a Model of Children's Story Comprehension," AI TR-266, MIT, Cambridge, Ma.

Chomsky, N. (1965) Aspects of the Theory of Syntax. MIT Press, Cambridge, Ma.

Codd, E. F. (1970) "A Relational Model of Data for Large Shared Data Banks," CACM 13 (6), 377-387.

Colby, K. M. (1973) "Simulation of Belief Systems," in Schank and Colby (eds.), Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Feigenbaum, E. A. and J. Feldman, eds. (1963) Computers and Thought. McGraw-Hill, New York.

Fikes, R. E., P. E. Hart and N. J. Nilsson (1972a) "Learning and Executing Generalized Robot Plans," Artificial Intelligence 3, 251-288.

Fikes, R. E., P. E. Hart and N. J. Nilsson (1972b) "Some New Directions in Robot Problem Solving," in Meltzer and Michie (eds.), Machine Intelligence 7. Edinburgh Univ. Press, Edinburgh.

Fikes, R. E. and N. J. Nilsson (1971) "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence 2, 189-208.

Fillmore, C. (1968) "The Case for Case," in Bach and Harms (eds.), Universals in Linguistic Theory. Holt, Rinehart and Winston, New York.

Friedman, D. P. (1973) "GROPE: A Graph Processing Language and its Formal Definition," Tech. Report TR-20, Dept. of Computer Sciences, University of Texas, Austin.

Gordon, D. and G. Lakoff (1975) "Conversational Postulates," in Cole and Morgan (eds.), Syntax and Semantics 3. Academic Press, New York.

Gordon, G. (1969) System Simulation. Prentice-Hall, Englewood Cliffs, N. J.

Green, B. F., A. K. Wolf, C. Chomsky and K. Laughery (1963) "Baseball: An Automatic Question Answerer," in Feigenbaum and Feldman (eds.), Computers and Thought. McGraw-Hill, New York.

Green, C. (1969) "Theorem-Proving by Resolution as a Basis for Question-Answering Systems," in Meltzer and Michie (eds.), Machine Intelligence 4. American Elsevier, New York.

Green, C. and B. Raphael (1968) "The Use of Theorem-Proving Techniques in Question-Answering Systems," Proc. ACM 23rd Nat. Conf., 169-181.

Hayes, P. (1971) "A Logic of Action," in Meltzer and Michie (eds.), Machine Intelligence 6. American Elsevier, New York.

Hayes, P. (1973) "The Frame Problem and Related Problems in Artificial Intelligence," in Elithorn and Jones (eds.), Artificial and Human Thinking. Elsevier Scientific, New York.

Hendrix, G. G. (1973a) "Question Answering Via Canonical Verbs and Semantic Models: A Model of Textual Meaning," Tech. Report NL12, Dept. of Computer Sciences, University of Texas, Austin.

Hendrix, G. G. (1973b) "Modeling Simultaneous Actions and Continuous Processes," Artificial Intelligence 4, 145-180.

Hendrix, G. G., C. W. Thompson and J. Slocum (1973) "Language Processing Via Canonical Verbs and Semantic Models," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 262-269.

Hewitt, C. (1969) "PLANNER: A Language for Proving Theorems in Robots," First Int. Joint Conf. on A. I., Washington, D. C., 295-301.

Hewitt, C. (1971) "Procedural Embedding of Knowledge in PLANNER," Proc. of the Second Int. Joint Conf. on A. I., London, 167-182.

Hopcroft, J. E. and J. D. Ullman (1969) Formal Languages and their Relation to Automata. Addison-Wesley, Reading, Ma.

Kaplan, R. M. (1972) "Augmented Transition Networks as Psychological Models of Sentence Comprehension," Artificial Intelligence 3, 77-100.

Katz, J. J. (1966) The Philosophy of Language. Harper and Row, New York.

Kay, M. (1973) "The MIND System," in Rustin (ed.), Natural Language Processing. Algorithmics Press, New York.

Kelly, M. D. (1971) "Edge Detection in Pictures by Computer Using Planning," in Meltzer and Michie (eds.), Machine Intelligence 6. American Elsevier, New York.

Lindsay, R. K. (1963) "Inferential Memory as the Basis of Machines Which Understand Language," in Feigenbaum and Feldman (eds.), Computers and Thought. McGraw-Hill, New York.

Lindsay, R. K. (1973) "In Defense of Ad Hoc Systems," in Schank and Colby (eds.), Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Lindsay, P. H. and D. A. Norman (1972) Human Information Processing. Academic Press, New York.

McCarthy, J., et al. (1965) LISP 1.5 Programmer's Manual. MIT Press, Cambridge, Ma.

McCarthy, J. (1968) "Programs with Common Sense," in Minsky (ed.), Semantic Information Processing. MIT Press, Cambridge, Ma.

McCarthy, J. and P. Hayes (1969) "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in Meltzer and Michie (eds.), Machine Intelligence 4. American Elsevier, New York.

McDermott, D. V. and G. J. Sussman (1972) The Conniver Reference Manual. A. I. Memo, MIT Project Mac.

Mealy, G. H. (1955) "A Method for Synthesizing Sequential Circuits," Bell System Technical Journal 34 (5), 1045-1079.

Minsky, M. (1974) "A Framework for Representing Knowledge," Memo 306, A. I. Laboratory, MIT, Cambridge, Ma.

Minsky, M., ed. (1969) Semantic Information Processing. MIT Press, Cambridge, Ma.

Montague, R. (1974) Formal Philosophy: Selected Papers of Richard Montague, R. H. Thomason (ed.). Yale University Press, New Haven, Conn.

Mylopoulos, J., N. Badler, L. Melli and N. Roussopoulos (1973) "1.PAK: A SNOBOL-Based Programming Language for Artificial Intelligence Applications," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 691-696.

Newell, A., et al. (1973) Final Report of a Study Group on Speech Understanding Systems. North Holland, Amsterdam.

Newell, A., J. C. Shaw and H. A. Simon (1958) "Chess-Playing Programs and the Problem of Complexity," IBM Journal of Research and Development 2 (4), 320-335.

Newell, A., J. C. Shaw and H. A. Simon (1960) "Report on a General Problem Solving Program," Proc. Int. Conf. on Information Processing, UNESCO, Paris, 256-264. (Reprinted in Computers and Automation, July 1959.)

Newell, A. and H. A. Simon (1963) "GPS, A Program That Simulates Human Thought," in Feigenbaum and Feldman (eds.), Computers and Thought. McGraw-Hill, New York.

Neumann, J. V. (1925) "Eine Axiomatisierung der Mengenlehre," Jour. fur reine u. angew. Math., (154), 219-240.

Nilsson, N., (1971) Problem Solving Methods in Artificial Intelligence. McGraw-Hill, New York.

Nilsson, N., ed. (1975) "Artificial Intelligence - Research and Applications," Project 3805 Progress Report, Artificial Intelligence Center, SRI, Menlo Park, Ca.

Norman, D. (1969) Memory and Attention. Wiley and Sons, New York.

Norman, D., ed. (1970) Models of Human Memory. Academic Press, New York.

Palme, J. (1971) "Making Computers Understand Natural Language," in Findler and Meltzer (eds.), Artificial Intelligence and Heuristic Programming. American Elsevier, New York.

Polya, G. (1945) How to Solve It. Princeton Univ. Press, Princeton, N. J.

Pratt, T. W. and D. P. Friedman (1971) "A Language Extension for Graph Processing and its Formal Semantics," CACM 14 (7), 460-467.

Quillian, M. R. (1968) "Semantic Memory," in Minsky (ed.), Semantic Information Processing. MIT Press, Cambridge, Ma.

Quillian, M. R. (1969) "The Teachable Language Comprehender: A Simulation Program and Theory of Language," CACM 12 (8), 459-476.

Quine, W. V. D. (1951) Mathematical Logic. Harvard University Press, Cambridge, Ma.

Quine, W. V. D. (1961) From a Logical Point of View. Harper and Row, New York.

Raphael, B. (1968) "SIR: A Computer Program for Semantic Information Retrieval," in Minsky (ed.), Semantic Information Processing. MIT Press, Cambridge, Ma.

Raphael, B. (1970) "The Frame Problem in Problem-Solving Systems," Proc. Adv. Study Inst. on Artificial Intelligence and Heuristic Programming, Menaggio, Italy.

Reboh, R. and E. Sacerdoti (1973) "A Preliminary QLISP Manual," TN 81, Artificial Intelligence Center, SRI, Menlo Park, Ca.

Rieger, C. J. (1974) "Conceptual Memory: A Theory and Computer Program for Processing Meaning Content of Natural Language Utterances," AIM-233, Stanford Artificial Intelligence Laboratory, Stanford, Ca.

Robinson, J. A. (1965) "A Machine-Oriented Logic Based on the Resolution Principle," JACM 12 (1), 23-41..

Rulifson, J., J. A. Derkson and R. W. Waldinger (1972) "QA4: A Procedural Calculus for Intuitive Reasoning," TN 73, Artificial Intelligence Center, SRI, Menlo Park, Ca.

Rumelhart, D. E., P. H. Lindsay and D. A. Norman (1972) "A Process Model for Long Term Memory," in Tulving and Donaldson (eds.), Organization of Memory. Academic Press, New York.

Rumelhart, D. E. and D. A. Norman (1973) "Active Semantic Networks as a Model of Human Memory," Advance Paper Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 450-457.

Sacerdoti, E. D. (1974) "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence 5, 115-135.

Sacerdoti, E. D. (1975) "A Structure for Plans and Behavior," Ph.D. thesis, Stanford University, Stanford, Ca.

Salomma, A. (1969) Theory of Automata. Pergamon Press, Oxford.

Samuel, A. L. (1963) "Some Studies in Machine Learning Usin Using the Game of Checkers," in Feigenbaum and Feldman (eds.), Computers and Thought. McGraw-Hill, New York.

Sandewall, E. (1970) "A Set-Oriented Property Structure Representation for Binary Relations," in Meltzer and Michie (eds.), Machine Intelligence 5. American Elsevier, New York.

Sandewall, E. (1971a) "A Programming Tool for Management of a Predicate-Calculus-Oriented Data Base," Proc. Second Int. Joint Conf. on A. I., London, 159-166.

Sandewall, E. (1971b) "Representing Natural Language Information in Predicate Calculus," in Meltzer and Michie (eds.), Machine Intelligence 6. American Elsevier, New York.

Sandewall, E. (1972) "PCF-2, A First-Order Calculus for Expressing Conceptual Information," Computer Science Report, Computer Science Dept., Uppsala University, Uppsala, Sweden.

Schank, R. (1971) "Finding the Conceptual Content and Intention in an Utterance in Natural Language Conversation," Proc. of the Second Int. Joint Conf. on A. I., London, 444-454.

Schank, R. (1972) "Conceptual Dependency: A Theory of Natural Language Understanding," Cognitive Psychology 3 (4), 552-631.

Schank, R. (1873a) "The Fourteen Primitive Actions and Their Inferences," Stanford AIM-183, Computer Science Dept., Stanford University, Stanford, Ca.

Schank, R. (1973b) "Identification of Conceptualizations Underlying Natural Language," in Schank and Colby (eds.), Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Schank, R., et al. (1973) "MARGIE: Memory, Analysis, Response, Generation and Inference on English," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 255-261.

Schank, R. and K. Colby, eds. (1973) Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Schank, R., N. Goldman, C. J. Rieger and C. K. Riesbeck (1972) "Primitive Concepts Underlying Verbs of Thought," AIM-162, Stanford University, Stanford, Ca.

Schubert, L. K. (1974) "Extending the Expressive Power of Semantic Networks," TR 74-18, Dept. of Computer Science, University of Alberta, Edmonton, Alberta, Canada.

Schwarcz, R. M., J. F. Burger and R. F. Simmons (1970) "A Deductive Question Answerer for Natural Language Inference," CACM 13 (3), 167-183.

Scragg, G. W. (1974) "Answering Questions about Processes," Dissertation, University of California, San Diego.

Shapiro, S. (1971) "A Net Structure for Semantic Information Storage, Deduction and Retrieval," Proc. of the Second Int. Joint Conf. on A. I., London, 512-523.

Siklóssy, L. and J. Dreussi (1973) "An Efficient Robot Planner Which Generates Its Own Procedures," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, 423-430.

Siklóssy, L. and J. Roach (1973) "Proving the Impossible is Impossible is Possible: Disproofs Based on Hereditary Partitions," Advance Papers of the Third Int. Joint Conf. on A. I., Stanford, Ca., 383-387.

Simmons, R. F. (1966) "Storage and Retrieval of Aspects of Meaning in Directed Graph Structures," CACM 9 (3), 211-215.

Simmons, R. F. (1970) "Natural Language Question Answering Systems: 1969," CACM 13 (1), 15-30.

Simmons, R. F. (1972) "Some Semantic Structures for Representing English Meanings," in Carroll and Freedle (eds.), Language Comprehension and the Acquisition of Knowledge. V. H. Winston, Washington, D. C.

Simmons, R. F. and B. C. Bruce (1971) "Some Relations Between Predicate Calculus and Semantic Net Representations of Discourse," Advance Papers of the Second Int. Joint Conf. on A. I., London, 524-530.

Simmons, R. F., J. F. Burger and R. M. Schwarcz (1968) "A Conceptual Model of Verbal Understanding," Proc. AFIPS 1968 Fall Joint Computer Conference, Vol. 33, San Francisco, Ca., 441-456.

Simmons, R. F. (1973) "Semantic Networks: Their Computation and Use for Understanding English Sentences," in Schank and Colby (eds.), Computer Models of Thought and Language. W. H. Freeman and Co., San Francisco.

Simmons, R. F. and J. Slocum (1972) "Generating English Discourse from Semantic Networks," CACM 15 (10), 891-905.

Slocum, J. (1974) "The Graph Processing Language GROPE," Tech. Report NL-22, Dept. of Computer Sciences, University of Texas, Austin.

Slocum, J. (1973) "Question Answering Via Canonical Verbs and Semantic Models: Generating English from the Model," Tech. Report NL-13, Dept. of Computer Sciences, University of Texas, Austin.

Teitelman, W. (1974) INTERLISP Reference Manual. Bolt, Beranek and Newman, Cambridge, Ma., and Xerox, Palo Alto, Ca.

Tesler, L., H. Enea and K. Colby (1968) "A Directed Graph Representation for Computer Simulation of Belief Systems," Math. Biosciences 2 (1/2).

Thompson, C. W. (1973) "Question Answering Via Canonical Verbs and Semantic Models: Parsing to Canonical Verb Forms," Tech. Report NL-11, Dept. of Computer Sciences, University of Texas, Austin.

Walker, D. E. (1974) "Speech Understanding Research, Annual Report," Progect 1526, Artificial Intelligence Center, SRI, Menlo Park, Ca.

Walker, D. E. (1975) "Speech Understanding Research," Annual Report, Project 3804, Artificial Intelligence Center, SRI, Menlo Park, Ca.

Whitehead, A. N. (1929) Process and Reality. The Macmillan Co. (Reprinted by Free Press, New York, 1969.)

Whitehead, A. N. (1920) The Concept of Nature. Cambridge Univ. Press, London. (Reprinted in 1971.)

Winograd, T. (1972) Understanding Natural Language. Academic Press, New York.

Winograd, T. (1974) "Five Lectures on Artificial Intelligence," AIM-246, Stanford Artificial Intelligence Laboratory, Stanford, Ca.

Winston, P. H. (1970) "Learning Structural Descriptions from Examples," AI TR-231, Artificial Intelligence Laboratory, MIT, Cambridge, Ma.

Woods, W. A. (1967) "Semantics for a Question-Answering System," Ph.D. thesis, Division of Eng. and Applied Physics, Harvard University, Cambridge, Ma.

Woods, W. (1968) "Procedural Semantics for a Question Answering Machine," Proc. AFIPS 1968 Fall Joint Computer Conference, Vol. 31, Washington, D. C., 457-472.

Woods, W. A. (1970) "Transition Network Grammars for Natural Language Analysis," CACM 13 (10), 591-606.

Woods, W. A. (1973) "An Experimental Parsing System for Transition Network Grammars," in Rustin (ed.), Natural Language Processing. Algorithmics Press, New York.

Woods, W. A. (forthcoming 1975) "What's in a Link: Foundations of Semantic Networks," in Bobrow and Collins (eds.), Representation and Understanding: Studies in Cognitive Science. Academic Press, New York.

Zadeh, L. A. (1974a) "A Fuzzy-Algorithmic Approach to the Definition of Complex or Imprecise Concepts," Memo No. ERL-M474, Electronics Research Lab., University of California, Berkeley.

Zadeh, L. A. (1974b) "Fuzzy Logic and Approximate Reasoning," Memo No. ERL-M479, Electronics Research Lab., University of California, Berkeley.