SRI International

A NETWORK-BASED KNOWLEDGE REPRESENTATION AND ITS NATURAL DEDUCTION SYSTEM

TECHNICAL NOTE 147

JULY 1977

By: Richard Fikes
Xerox, Palo Alto Research Center
Palo Alto, California

Gary G. Hendrix
Artificial Intelligence Center

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Abstract

We describe a knowledge representation scheme called K-NET and a problem solving system called SNIFFER designed to answer queries using a K-NET knowledge base. K-NET uses a partitioned semantic net to combine the expressive capabilities of the first-order predicate calculus with linkage to procedural knowledge and with full indexing of objects to the relationships in which they participate. Facilities are also included for representing taxonomies of sets and for maintaining hierarchies of contexts. SNIFFER is a manager and coordinator of deductive and problem-solving processes. The basic system includes a logically complete set of natural deduction facilities that do not require statements to be converted into clause or prenex normal form. Using SNIFFER's coroutine-based control structure, alternative proofs may be constructed in pseudo-parallel and results shared among them. In addition, SNIFFER can also manage the application of specialist procedures that have specific knowledge about a particular domain or about the topology of the K-NET structures. For example, specialist procedures are used to manipulate taxonomic information and to link the system to information in external data bases.

A Network-Based Knowledge Representation and its Natural Deduction System

by

Richard Fikes and Gary Hendrix

Introduction

This paper describes a question answering system whose principal components are a network-based knowledge representation scheme called K-NET and a problem solving system called SNIFFER (an acronym for Semantic Net Interpretation Facility Fortified with External Routines), designed to answer queries using a K-NET knowledge base.

The goal of the effort has been to create a design that allows specialized representations and deductive schemes to be used where they are effective, while providing recourse to a logically complete natural deduction mechanism when necessary. SNIFFER has been designed with the intention that most of the question answering work will be performed by special domain-dependent procedures. These specialists can take advantage of the particular topology of the K-NET structures designed to represent domain-specific types of knowledge. Specialist procedures also allow SNIFFER to do certain types of problem solving usually considered outside the range of conventional deduction. For example, specialists may be added that know how to extract information from conventional data bases or do scheduling and planning. In this paper we seek to indicate the handles for adding specialized knowledge while concentrating on the fundamental issues of implementing natural deduction for network systems.

SNIFFER and K-NET are evolving systems, versions of which have been used as major components in larger systems developed in the SRI Artificial Intelligence Center, including the SRI Speech Understanding System (Walker 1976).

To help the reader relate our work to other knowledge representation facilities and problem

solving systems, we begin by presenting the distinguishing and characterizing features of our system before focusing on a more detailed overview that elaborates on these features and provides concrete examples.

Characterizing Features of K-NET

K-NET provides facilities for creating a partitioned semantic network of labeled nodes connected by labeled unidirectional arcs. A node represents an entity in the world being modeled and an arc represents a binary relationship between the nodes that it connects. For example, the nodes **John** and **Men** in Figure 1 represent a man John and the set of all men, respectively. The arc labeled "e" from **John** to **Men** indicates that John is an element of the set of men. Relationships can be considered to be entities and be represented by nodes with "case" arcs pointing to the participants in the relationship. For example, node Q represents the ownership relationship (situation) existing between John and the automobile "Ole-Black" over the time interval from t_1 to t_2.

K-NET can be characterized by the following list of features:

* Facilities are provided for representing multiple "worlds" and the relationships among them. In particular, the network can be partitioned into subnets (called spaces). Spaces can be hierarchically embedded by treating an entire space at one level in the hierarchy as a single node in a space at the next higher level. A "context" mechanism exists that allows only a given set of spaces to be "visible" to the retrieval procedures at any one time. Examples of alternative worlds include those contained in a disjunction, or the world composed of the set of beliefs that John has about Sally as opposed to the world composed of the set of beliefs that Sally has about herself.

* The expressive facilities of the representation scheme include those of the first order predicate calculus, including existential and universal quantification. (Higher order predicates are also representable in K-NET, but only trivial interpretation facilities exist for them in SNIFFER.) That is, the knowledge base can contain statements represented as negations ("John does not love

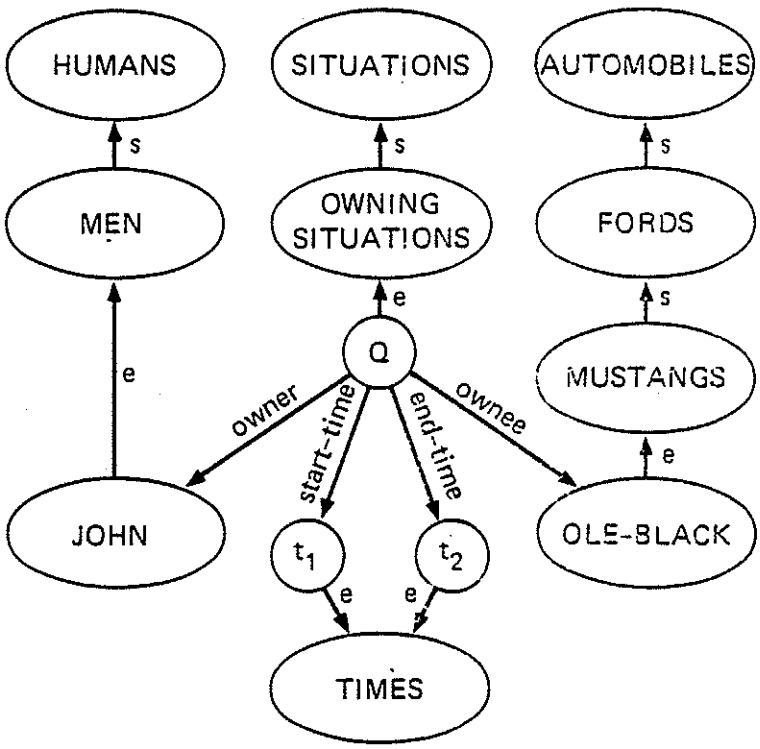


FIGURE 1 A SIMPLE SEMANTIC NETWORK

Mary."), disjunctions ("John loves either Sally or Sue."), or implications ("If Sue answers John's phone call, then John will ask Sue for a date."), and containing arbitrary nestings of existential and universal quantifiers ("Every boy has been in love sometime.").

* Taxonomies of sets are modeled by the topology of the network so that they become the basic skeleton upon which the knowledge base is built. For example, one can directly represent the relationships "Ford is an element of Companies distinct from G.M." and "Mustangs is a subset of Automobiles distinct from Model-T's". One can also associate with a set characteristic properties common to all elements of the set, such as "All Mustangs are built by Ford".

* Procedures may be attached to the network to interface it to other knowledge sources such as conventional data bases or arithmetic algorithms. When called by SNIFFER, these procedures extend the network by creating new nodes and arcs representing the information acquired from the other sources. Links to these procedures are explicitly represented in the network so that their existence and role can be reasoned about and discussed by the system.

* The network provides indices that facilitate associative retrieval of the relationships in which any given knowledge base entity is involved. For example, retrieval of all females that John loves can be indexed through the node representing John, the node representing the set of loving relationships, or the node representing the set of females. The basic mechanism is one that allows immediate access to all of a node's incoming and outgoing arcs that are visible in any given set of spaces.

Characterizing Features of SNIFFER

SNIFFER is a "natural" deduction system (as in Bledsoe, et al., 1972) that is given two net structures as input, one representing a knowledge base and the other representing a query (usually a translation of a question originally stated in English). It treats the query as a pattern and attempts to find instances of the pattern in the knowledge base, or equivalently, it treats the query as a theorem to be proved and attempts to find instantiations for its

existentially quantified variables. Results are returned in the form of sets of "bindings" for the variables in the pattern. For example, the question "Who does John love?" is translated into a net structure representing the pattern "John loves x" (or the theorem (∃x)[Loves(John,x)]), and SNIFFER returns bindings for x such as (x, Mary). Answers may either be retrieved from the knowledge base or derived using knowledge base theorems and procedures.

SNIFFER can be characterized by the following list of features:

* Associative retrieval of relationships from the knowledge base is performed using the K-NET indexing facilities.

* Efficient, special purpose deductive procedures are used for extracting information from the K-NET taxonomies. For example, if the knowledge base indicates that x is an element of the set of Mustangs, that Mustangs are a subset of the set of sports cars, and that sports cars are a subset of the set of automobiles, then SNIFFER can conclude that x is an automobile by using procedures that follow the chain of **elementOf** and **subsetOf** arcs, thereby bypassing the more cumbersome, general-purpose deductive machinery.

* Facilities are included for answering questions and using knowledge base statements composed of conjunctions, disjunctions, and implications, containing arbitrarily embedded universally and existentially quantified variables.

* Queries and knowledge base statements are processed in the "natural" form in which they are input, without converting into a canonical form such as clause form or prenex normal form. This capability eliminates "explosive" conversions (such as converting the disjunction (a∧b∧c) ∨ (d∧e∧f) ∨ (g∧h∧i) into clause form which consists of 27 clauses each containing 3 disjuncts) and unnecessary conversions (such as conversion of a disjunctive question's complex disjuncts when one of its simple disjuncts can easily be shown to be true). In addition, the intuitiveness and heuristic value of the form in which statements are input (as implications, for example) is maintained.

* A logically complete set of natural deduction rules are used that reason backwards from the question. These rules use such techniques as case analysis, hypothetical reasoning, and the establishing of subgoals. For example, to answer a question that is in the form of an implication, SNIFFER might use hypothetical reasoning by assuming the implication's antecedent and then pursuing a proof of the consequent as a subgoal.

* A flexible coroutine-based control structure allows the construction of alternative proofs in a pseudo-parallel manner, with results being shared among the alternatives. Each partial proof has its own local scheduler to determine how its proof attempt should be continued. There is an executive scheduler that uses information supplied by the local schedulers to determine which partial proof is to be given control at each step. The various schedules provide the facilities necessary to allow reasonable heuristic guidance of the total deduction and retrieval process.

* User-supplied procedures may participate in the attempt to find answers in two ways. First, procedures included in the K-NET knowledge base may be invoked to access information in knowledge sources that are external to K-NET. Second, SNIFFER allows the inclusion of user-supplied procedures that extend the system's problem solving capabilities. Such procedures may add heuristics to the deductive strategies or even integrate new knowledge sources into the system, such as data bases and planners. Facilities are available to these procedures for creating alternative proofs, manipulating schedules, altering priorities, and establishing "demons" so that the user can create strategies that augment and interact with those that already exist in the system.

* SNIFFER is implemented as a "generator" (see Teitelman, 1975) so that after returning an answer it can be restarted to seek a second answer to a query. For example, given the question "Who owns a Mustang?" SNIFFER may first produce the answer "John", then be "pulsed" again to produce "Mary", etc. This style of answer production allows the user to examine each answer as it is produced and dynamically determine whether additional answers are needed.

* "No" answers are determined by finding an affirmative answer to the question's negation. For example, if given the question "Does John love Mary?", SNIFFER will attempt to prove "John does not love Mary" in addition to attempting to prove "John loves Mary".

Overview Description of K-NET

In this section we will describe and illustrate how K-NET is used to encode knowledge. Throughout the section reference will be made to the example knowledge base shown in Figure 2, which represents some facts about automobiles.

Taxonomies

Major portions of the semantics of a task domain can often be expressed naturally by a taxonomy of sets that indicates the major sets of objects in the domain and the relationships between the sets. The power of the taxonomy can be enhanced further by the inclusion of statements that specify necessary and/or sufficient conditions for membership in the sets. K-NET provides the following facilities designed specifically for encoding such taxonomies.

S arcs indicate "subset of" relationships. For example, the s arc in Figure 1 from the Men node to the Humans node indicates that the set of men is a subset of the set of all humans.

Most sibling subsets described in taxonomies are disjoint. Arcs labeled ds are used in K-NET to represent this disjointness property in a concise and easily interpretable manner. A ds arc from a node x to a node z indicates that the set represented by x is a subset of the set represented by z and that the x set is disjoint from any other set represented by a node with an outoing ds arc to z. For example, the ds arcs in the figure (i.e., Figure 2) emanating from the Humans and Companies nodes indicate that the set of humans and the set of companies are disjoint subsets of the set of legal persons.

Since each node in most taxonomies represents a distinct entity, and in general an entity can be represented by any number of nodes in a K-NET, arcs labeled de (for "distinct element") are used to indicate that two or more nodes each represents a distinct element of a set. In

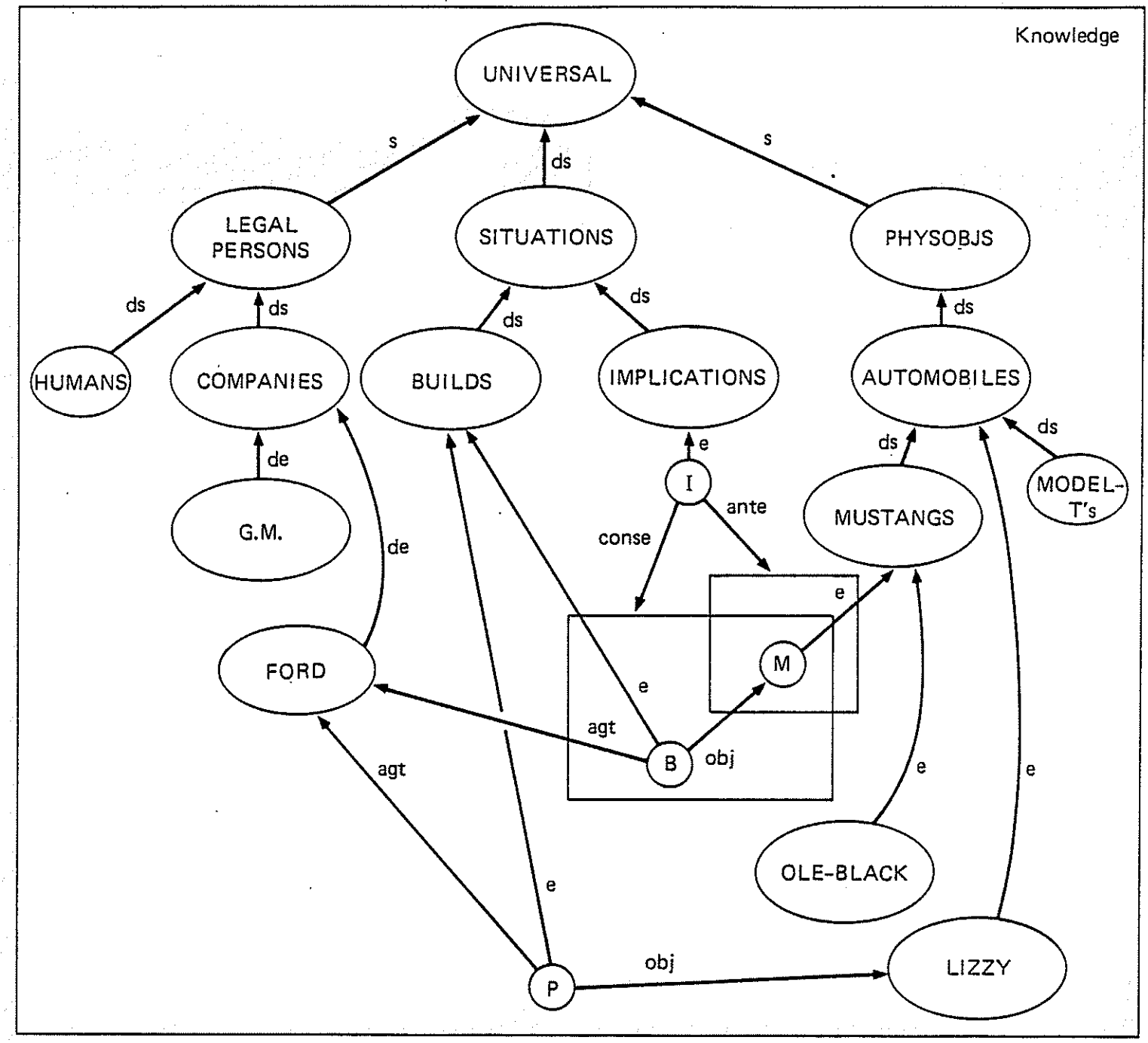


FIGURE 2 AN EXAMPLE KNOWLEDGE BASE

particular, a de arc from a node x to a node z indicates that the entity represented by x is an element of the set represented by z and that the x entity is distinct from any other entity represented by a node that has an outgoing de arc to z. For example, the de arcs in the figure emanating from the G.M. and Ford nodes indicate that G.M. and Ford are distinct members of the set of companies.

E arcs are used to indicate "element of" relationships without making a commitment to distinctness. For example, Fred. Jill, and Mary may be known to be distinct elements of Riders, the set of people that rode to the airport in Fred's car. If some fact is known about the driver of the car and the identity of the driver has not yet been determined, then a node D representing the driver may be linked to set Riders by an e arc. The node D can be used to encode information about the unnamed driver without specifically indicating which of the distinct elements of Riders is the driver.

Situations

SNIFFER assumes that relationships other than elementOf and subsetOf are represented by nodes having outgoing case arcs pointing to the participants in the relationship (such as node P in the figure, which represents the relationship "Ford built Lizzy"). This representational convention allows an arbitrary amount of information to be stored with a relationship (using outgoing case arcs) and allows associative retrieval of the relationship using the network's indexing facilities. Such relationships are grouped by type into sets and these sets are considered to be subsets of the set of all "situations". For example, Builds (the set of all situations in which building takes place) and Implications are disjoint subsets of Situations in the figure, and node P represents an element of the Builds set, a particular building situation in which Ford is the agent and Lizzy is the object built. (The situation represented by P took place over an interval of time from StartTime to EndTime. These time cases would be present in a more complete description of P.)

Spaces and Vistas

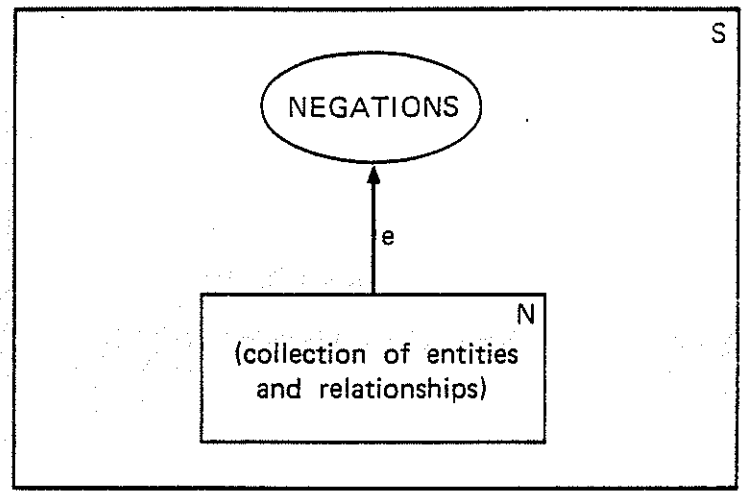
Perhaps the primary feature that distinguishes K-NET from other semantic networks is that a net can be partitioned into subnets, and relationships among the subnets can be explicitly and easily represented (see Hendrix, 1975). All nodes and arcs in a K-NET are "elements" of at least one "space" (i.e., subnet). In the figures, such spaces are depicted by boxes. For example, node **P** in the figure and the **obj** arc from **P** to **Lizzy** are elements of the **Knowledge** space. A space can be (and usually is) a node in some other space. For example, in the figure the **conse** arc from node **I** points to a node in the **Knowledge** space that is itself a space. When retrieving information from a network, it is convenient to have only a specified list of spaces, called a "vista", visible to the retriever. For example, the vista that would typically be used when retrieving information from the space pointed to by the conse arc in the figure consists of the space itself and the **Knowledge** space.

Negations, Disjunctions, and Implications

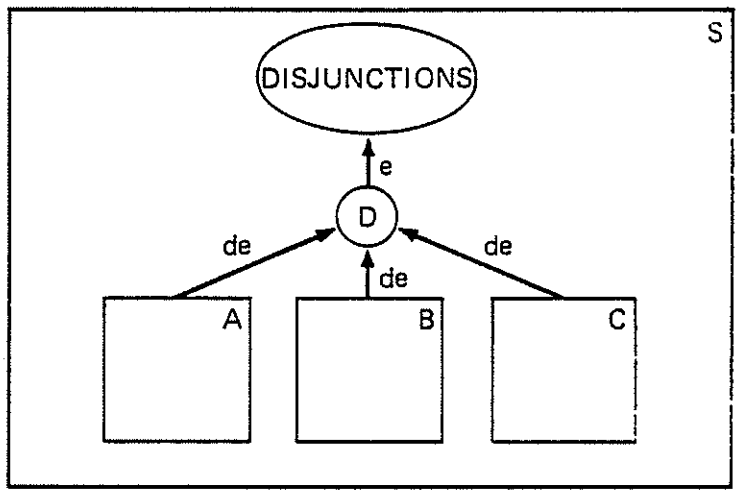
A representation scheme for negations, disjunctions, and implications must allow one or more "worlds" to be described and a relationship to be asserted among the worlds (e.g., that at least one of them is true). K-NET's partitioning facilities provide the required capabilities for creating just such a scheme.

A negation occurring in some space s describes a collection of entities and relationships, and asserts that no collection satisfying the description can exist in the world represented by space s. We represent such a negation as shown in Figure 3a by creating a space to describe the collection, and by adding the created space to space s as a node with an outgoing e arc to negations, the node that represents the set of all negation relationships. For example, the statement "G.M. does not build convertibles" would be represented using a space describing a collection consisting of an entity C, an elementOf relationship between C and the set of convertibles, and a **build** relationship with agent **G.M.** and object C.

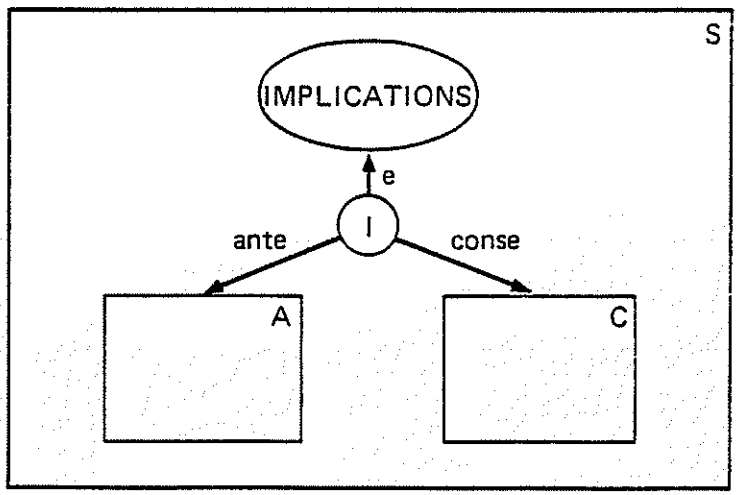
A disjunction occurring in a space s describes alternative collections of entities and



a. NEGATIONS ($\neg$N)



b. DISJUNCTION (A ∨ B ∨ C)



c. IMPLICATION (A → C)

FIGURE 3 ABSTRACTION OF LOGICAL CONNECTIVES

relationships, and asserts that entities and relationships satisfying at least one of those descriptions exists in the world represented by space s. As shown in Figure 3b, we describe each disjunct in a separate space and represent a disjunction as a set of such disjunct spaces.

An implication occurring in a space s describes two collections of entities and relationships, and asserts that if entities and relationships exist in the world represented by space s that satisfy the first of the two descriptions (the antecedent), then entities and relationships satisfying the second description (the consequent) also exist in that world. We represent an implication as shown in Figure 3c by a node with outgoing case arcs to spaces containing the descriptions of the antecedent and consequent. More concrete examples of implications will be presented in the next section.

Quantification

One of the important features of K-NET is that it provides facilities for representing arbitrarily nested existential and universal quantifiers. Existential quantification is a "built-in" concept in the sense that we take the occurrence of an element (i.e., a node or arc) in a space to be an assertion of the existence with respect to that space of the entity or relationship represented by the element. In particular, if an element occurs in the system's "knowledge space", then that element represents the system's belief that a corresponding entity or relationship exists in the domain being modeled.

Existential quantification and negation could be used to represent any universally quantified formula $(\forall x \in X)P(x)$ by making use of the following transformation:

$$(\forall x \in X)P(x) \equiv \sim\sim[(\forall x \in X)P(x)] \equiv_{\_} \sim[(\exists x \in X)\sim P(x)] .$$

The K-NET representation of the transformed formula is shown in Figure 4.

Although this representation is logically sound, it is extremely unappealing intuitively. The following transformation suggests a more attractive representation:

$$(\forall x \in X)P(x) \equiv_{\_} (\forall x)[(x \in X) \Rightarrow P(x)] .$$

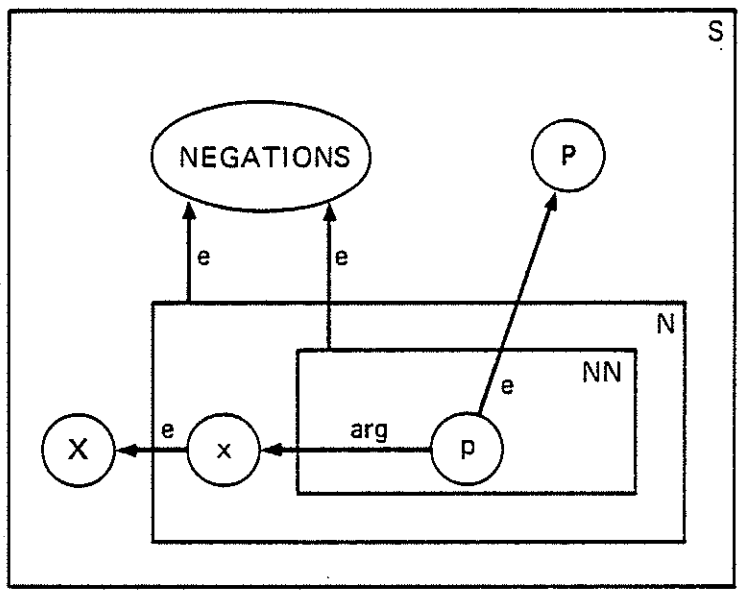


FIGURE 4 $\sim[(\exists x \in X)\sim P(x)]$

That is, any universally quantified formula can be represented as an implication whose antecedent specifies the "typing" of the universally quantified variable and whose consequent specifies the statement that is being made about any entity that satisfies the type restrictions.

A distinguishing feature of the universally quantified variable x in this implication is that it occurs in both the antecedent and the consequent. We have made use of this feature by adopting the convention in K-NET that if a node occurs in both the antecedent and the consequent spaces of an implication, then we consider it to be the representation of a universally quantified variable. This convention is, in fact, used as the primary means of representing universal quantification in our system.

When the main connective of a formula is an implication, it is not necessary to embed the formula in another implication to represent the universal quantification. That is:

$$(\forall x \in X)[Q(x) \Rightarrow R(x)] \equiv$$
$$(\forall x)\{(x \in X) \Rightarrow [Q(x) \Rightarrow R(x)]\} \equiv$$
$$(\forall x)\{[(x \in X) \wedge Q(x)] \Rightarrow R(x)\} \quad .$$

Figure 2 shows the K-NET representation of a concrete example of such an implication, namely the statement "For all M in the set of Mustangs, there exists a B such that B is an element of the Builds situations, the agent of B is Ford, and the object built is M."

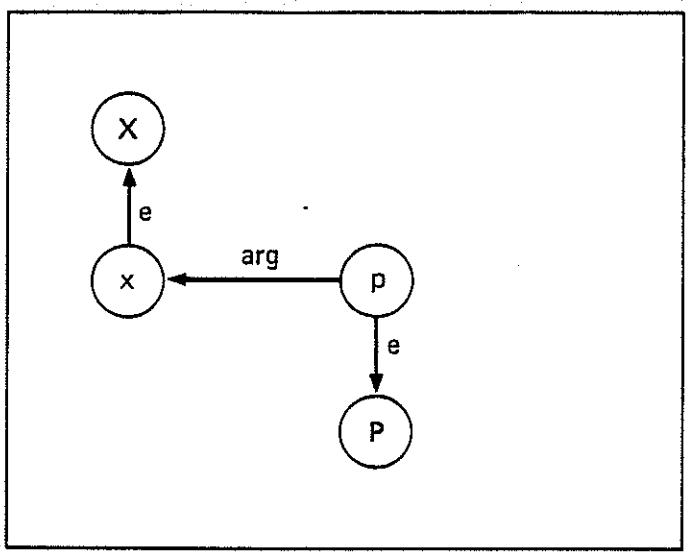
Arbitrary nesting of quantifiers may be achieved by placing implications in the consequent spaces of other implications. For example:

$$(\forall x \in X)(\exists y \in Y)(\forall z \in Z)P(x,y,z) \equiv$$
$$(\forall x)\{x \in X \Rightarrow (\exists y)[y \in Y \wedge (\forall z)(z \in Z \Rightarrow P(x,y,z))]\} \; .$$

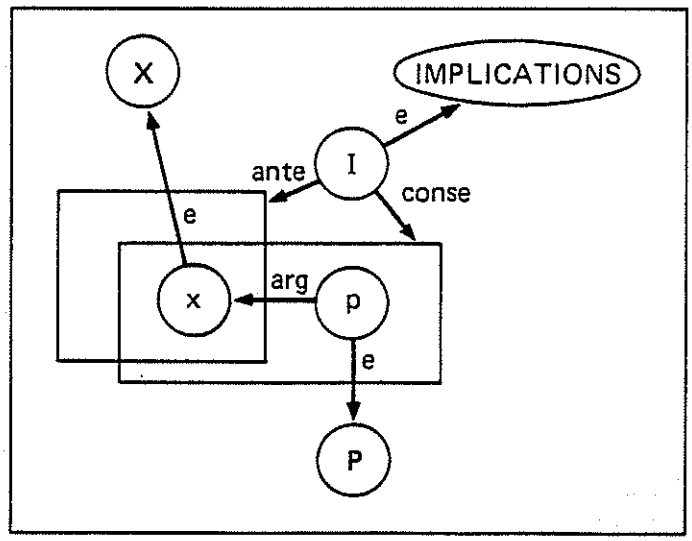
Figure 5 summarizes the conventions for representing quantification by contrasting the K-NET representations of $(\exists x \in X)P(x)$ and $(\forall x \in X)P(x)$.

Procedural Augmentation

For many applications, it is important for the system's knowledge base to include sources of



$(\exists x \epsilon X)P(x)$ or
$\exists x [x \epsilon X \wedge P(x)]$



$(\forall x \epsilon X)P(x)$ or
$\forall x [x \epsilon X \rightarrow P(x)]$

FIGURE 5 EXISTENTIAL AND UNIVERSAL QUANTIFICATION

information such as relational data bases or arithmetic algorithms external to the K-NET nets. (See Reiter, 1977, for another example of an inference system designed to interact with a relational data base.) We have adopted a set of conventions in K-NET for describing links to such external knowledge sources.

The links to external knowledge sources are represented by "theorems" (i.e., implications containing universally quantified variables) in the system's knowledge space that have the form exemplified by the network shown in Figure 6. Such theorems are interpreted to mean that if there is a successful application of the indicated function to a set of arguments that satisfy the description given in the antecedent, then the arguments and the results returned by the function can be used to create relationships and entities satisfying the description given in the consequent.

The particular theorem of Figure 6 indicates that an application of INTERLISP's PLUS function can be used to produce new instances of the Sums relation in the net. This theorem makes it unnecessary for all the instances of the Sums relation to be explicitly represented in the knowledge base. When SNIFFER attempts to match a pattern involving the sum of two numbers, it can use this theorem to form a call of the PLUS function and to translate the results of that call into the desired Sums relationship. The manner in which SNIFFER uses knowledge about the Applications set to create new relationships from the results of procedure calls is discussed below in the section on special purpose binder tasks.

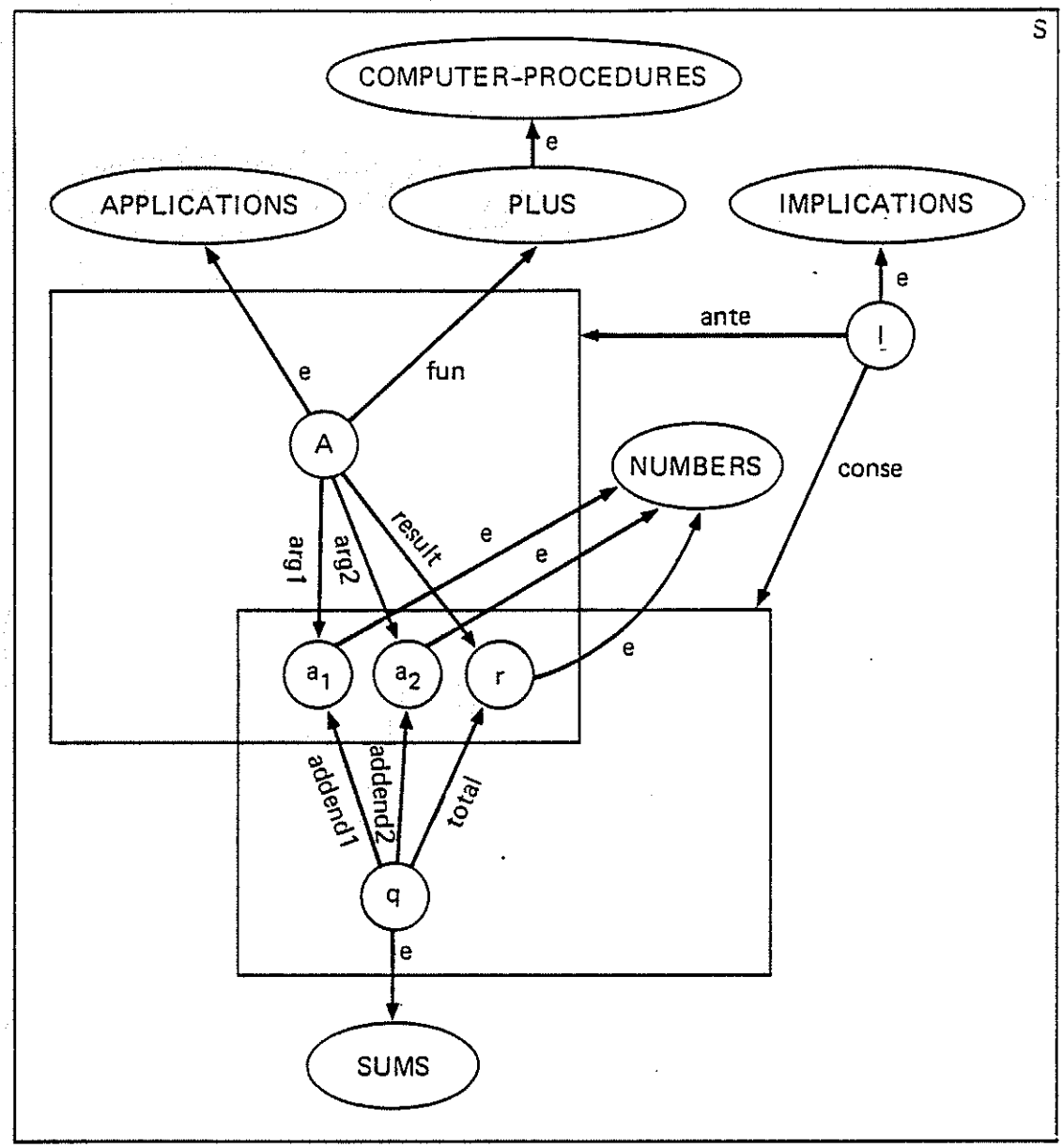


FIGURE 6 LINKING RELATION SUMS TO PROCEDURE PLUS

Overview Description of SNIFFER

This section describes and illustrates the basic features used by SNIFFER in retrieving and deriving information from K-NET structures. We begin by considering how SNIFFER is invoked and by illustrating how it would go about solving two simple problems. Attention is then turned to the overall control structure and to the operations performed by various components.

Introduction

SNIFFER is given as input a vista representing a query (the QVISTA) and a vista representing the beliefs that are to be considered true while answering the query (the KVISTA). Like other vistas, the QVISTA and KVISTA are lists of spaces. In aggregate, the nodes and arcs of the various spaces in the QVISTA describe a set of entities (i.e., objects and relationships) whose existence is to be established in the KVISTA. If a set of such entities can be found to exist, then SNIFFER returns a list of "bindings" that link the QVISTA descriptions to their KVISTA instantiations. Otherwise, SNIFFER attempts to prove that no such collection of entities can exist, so that a negative response can be given.

For example, Figure 7 shows a KVISTA and a QVISTA for the query "What company built Lizzy?". Given this QVISTA, SNIFFER seeks an element of the Builds situations set having both Lizzy as its object and an element of the Companies set as its agent. The Builds situation represented by node P in the KVISTA is found by using the incoming e arcs to the Builds node as an index, and a "Yes" answer is generated with P as the binding for node Z and the Ford node as the binding for node ?X. The "Yes" answer indicates that the question was based on a true premise, and the binding for ?X is the actual value that was sought.

Given the KVISTA and QVISTA shown in Figure 8, SNIFFER must carry out a derivation to answer the query using the KVISTA theorem "All Mustangs were built by Ford." The theorem is found by indexing on the incoming e arcs to the Builds node. A unification process determines that the relevant instance of the theorem is one in which the universally quantified variable M is replaced by Ole-Black. The theorem allows a new Builds situation to be asserted if it can be shown that Ole-Black is an element of the Mustangs set. A

subproblem is created to find that **ElementOf** relationship, and when the subproblem is solved, the new **Builds** situation is asserted and the desired bindings are assigned. In particular, node **?X** is again bound to **Ford** and **Z** is bound to the newly derived **Builds** situation.

Control Structure

As an introduction to SNIFFER's control structure, consider the following simplified description of how the system goes about answering queries. The basic process consists of selecting an unbound QVISTA arc and finding a match for the selected arc in the KVISTA. The matching arc then implies matches for the nodes at each end of the selected QVISTA arc. After each arc is bound, the process is repeated until all the arcs and nodes of the QVISTA have been bound.

This conceptually simple process is complicated by a number of factors. At each step in the process there are typically many alternatives that may be followed. For example, any of the unbound arcs in the QVISTA might be selected for consideration and each of these might be successfully bound to many KVISTA arcs. Another complicating factor is that some structures in the QVISTA will have no matches in the KVISTA, even though their existence is implied by statements in the KVISTA. Deductive machinery must be invoked to derive explicit representations of these implied structures. Within the deductive machinery, choices must be made between alternative strategies for pursuing a derivation and among the collection of KVISTA statements that could possibly be used to derive the desired matching structure.

The control structure that we have evolved for SNIFFER allows these various alternatives to be pursued in a pseudo-parallel "best first" manner. K-NET's partitioning facilities and INTERLISP's coroutines are used to create a system environment that allows each alternative to have its own subproblems, assumptions, and derived results, and for the choices among these alternatives to be guided by both built-in and user-supplied evaluation functions.

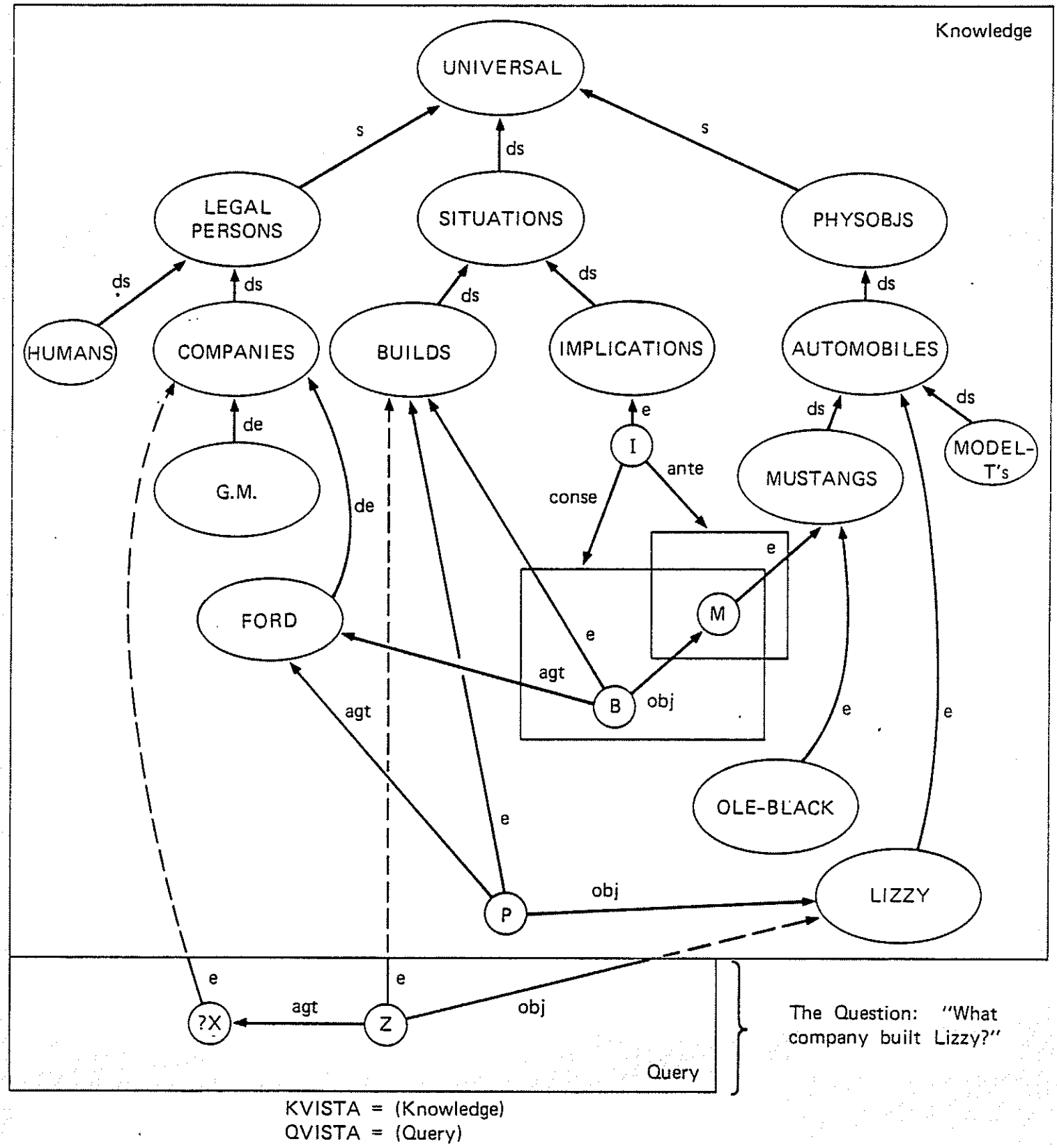


FIGURE 7 WHAT COMPANY BUILT LIZZY?

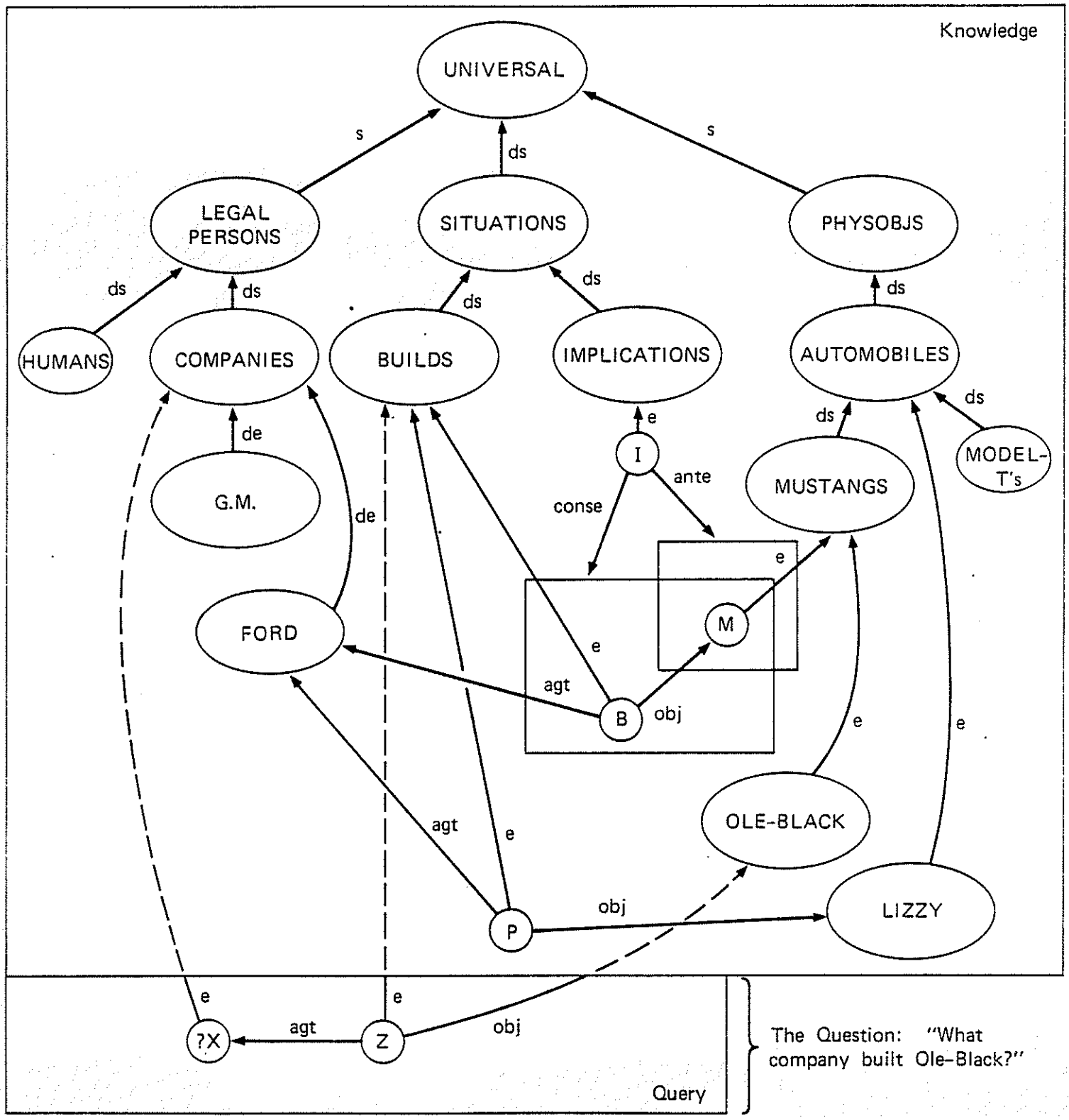


FIGURE 8 WHAT COMPANY BUILT OLE-BLACK?

The Environment Tree and Task Agendas

SNIFFER proceeds by building a tree of alternative proofs, each node of which represents a data environment that includes a set of choices of bindings for QVISTA elements and derivation strategies. Each time a choice is to be made in an environment, an offspring environment is created and the results of the choice are established in the offspring. For example, if a binding for a QVISTA element is found, then an offspring environment will be created in which the binding will be assigned. The search for additional bindings can then be continued in the parent environment, but SNIFFER is committed to the assigned binding in the offspring.

Included in each environment is a task agenda (patterned after the agenda mechanism in KRL-0, see Bobrow and Winograd, 1977) that defines n priority levels and allows a list of tasks to be stored at each priority level. The SNIFFER Executive typically proceeds by selecting an environment to give control to and then running the highest priority task on the selected environment's agenda. Each task is composed of a LISP function and a set of arguments upon which the function is operating. Typical tasks look for KVISTA descriptions matching a given QVISTA description or, if necessary, initiate derivations to deduce new explicit descriptions from implicit descriptions contained in KVISTA "theorems".

The Executive also has its own task agenda that is used to determine what to do at each step. Initially, this agenda has three tasks on it; one to initialize an environment tree to seek "Yes" answers to the query, one to initialize an environment tree to seek "No" answers to the query, and the one mentioned above that selects an environment, runs the task defined by that environment's agenda, and reschedules itself.

The agenda associated with the top environment in an environment tree initially contains a single task that selects for consideration unbound arcs that lie in the QVISTA. Each time the selector task is restarted, it selects another QVISTA arc, creates a "binder" task that will seek bindings for the selected arc, schedules the created task, and reschedules itself.

When a binder task finds a KVISTA arc that is a "candidate" (i.e., potential) binding, it

creates a new offspring environment in the environment tree that is a copy of the parent, assigns the binding in the offspring environment, and reschedules itself in the parent environment. Hence, at any given step, each terminal environment in the tree includes a partially formed alternative answer to the query.

Provisions have been made for attaching "demon" functions to QVISTA nodes and spaces in an environment. Demons attached to a QVISTA node, which are "fired" when a binding is assigned to the node, allow binder tasks to "pause" until other bindings have been assigned that can be used as indices. Demons attached to a QVISTA space, which are fired when bindings have been assigned to all the arcs and nodes in that space, are useful in completing derivations and returning results. For example, demons are attached to each QVISTA space in the initial environment of an environment tree. When the last of these demons fires in an environment, bindings will have been assigned to all QVISTA elements in that environment and an answer can be generated. The last demon causes the answer to be generated by scheduling an appropriate task on the Executive's agenda.

When an offspring environment is created, it inherits copies of its parent environment's data structures, including the agenda, demons, and list of assigned bindings. If a task or demon represents a "paused" coroutine that will be "resumed" when the task is run, then copying it conceptually produces a copy of the coroutine so that the original task or demon and the copy can run independently in their respective environments. For example, if a binder task is in a state such that it will consider relationship R as the next candidate binding and it is copied into an offspring environment's agenda, then the copy will also independently consider R as the next candidate binding. Similarly, a demon can be independently fired in each environment in which bindings for all the space's elements have been assigned. This powerful capability is implemented using the "spaghetti stack" facilities found in INTERLISP (Bobrow and Wegbreit, 1973).

Binder Tasks and User-Supplied Specialists

The Selector task in each environment's agenda selects unbound QVISTA arcs and creates

binder tasks that seek bindings for the selected arcs. The procedures used in the binder tasks embody the system's retrieval and derivational mechanisms.

Domain-Specific Augmentation

The primary way in which SNIFFER can be augmented and adapted to a particular problem domain is by providing additional procedures that can act as "expert" binder tasks for specialized classes of relationships. Such experts may add heuristic guidance to the deduction process or add completely new sources of knowledge.

For example, a binder task for ownership relationships might add heuristic guidance by knowing that objects usually have a unique owner. This task would look for bindings by following indices from the object to its owner rather than from the person to all the objects he/she owns or from the set of all ownership relationships.

Another expert binder task might be written for the relationship between a person and his telephone number. Rather than look for the person/number relationship in the K-NET, this procedure might look it up externally in a phone book file. The procedure would then create new structures in the KVISTA to encode the retrieved information and use this new structure in the binding.

Strategy Selectors

When a QVISTA arc has been selected, it is passed through a set of "strategy selectors", each of which is a function that can create a binder task for the arc and indicate whether additional selectors should be consulted. When a new function for finding bindings is added to the system, a strategy selector is written for it and added to the set of selectors. These strategy selectors provide a generalized form of pattern directed invocation of the binder tasks.

When no "specialist" binder task is available for a selected arc, a general purpose binder task is created that can seek bindings for any relationship or its negation using natural deduction

theorem proving strategies. It uses the net's indexing facilities to first find all atomic statements (i.e., relationships other than disjunctions, implications, or negated conjunctions) that contain possible bindings for the selected arc and then all nonatomic statements that can be used to derive bindings for the selected arc. For example, the general purpose binder task for arc Z--e-->Builds in Figure 8 would consider incoming e and de arcs to the Builds node as candidate bindings.

Ramification

When a binder task finds a candidate binding, it can apply the following "ramification" rules to determine what other bindings are directly implied by the candidate. First, if two arcs are to be bound to each other, then the from-node of the first arc must be bound to the from-node of the second arc and the to-node of the first arc must be bound to the to-node of the second arc. Second, we assume that a node can have at most one outgoing case (i.e., nontaxonomic) arc with any given arc label. Therefore, if two nodes are to be bound to each other and both nodes have outgoing case arcs with a common label, then those case arcs must also be bound to each other. For example, if in Figure 7 arc P--e-->Builds were the candidate binding for arc Z--e-->Builds, then bindings would be implied for nodes Z and ?X, and for the agt and obj arcs.

If a candidate binding implies a binding that is inconsistent with an existing binding (for example, one that assigns two different bindings to some QVISTA node, where ds and de arcs in the taxonomies indicate that the two bindings represent distinct entities), then the candidate can be rejected and another one sought. Hence, this ramification process acts as a powerful and efficient filter for candidate bindings as well as a producer of new bindings.

Self Scheduling

The decision as to which binder task should be given control in any given environment is made by allowing each such task to determine the priority level at which it is scheduled on the environment's task agenda. A task makes this determination by assessing the difficulty

of finding bindings for its QVISTA arc based on estimates of the number of indices (i.e., matching arcs) available in the KVISTA, knowledge about the semantics of the relationship being sought, knowledge about the effectiveness of the task's search method, etc. User supplied specialists may be written that are particularly adept at such assessments. The basic goal of the overall strategy is for the system to first seek bindings for those QVISTA arcs that are most highly constrained.

Deriving Bindings for ElementOf and SubsetOf Relationships

Included in SNIFFER are a set of functions embodying the semantics of the taxonomic relationships **e**, **de**, **s**, and **ds**. These functions provide the following eight services:

> Given a node representing some entity x, they can generate nodes representing entities y such that x is an element of y, y is an element of x, x is a subset of y, or y is a subset of x.
>
> Given two nodes representing entities x and y, they can determine whether x is an element of y, y is an element of x, x is a subset of y, or y is a subset of x. Possible answers are "Yes", "No", and "Unknown".

The algorithms used follow chains of s and ds arcs applying recursive rules such as the following:

> Two sets are disjoint if each of the nodes representing them has an outgoing ds arc to the same node, or if the sets are each subsets of disjoint sets.

These functions are used in SNIFFER wherever information is needed about SubsetOf or ElementOf relationships. In particular, they are used by the general purpose binder task to find candidate bindings for e and s arcs, and during the ramification process to test potential bindings of QVISTA nodes as to whether the bindings can satisfy the ElementOf and SubsetOf relationships specified for them in the QVISTA. Hence, these very important classes of deductions are carried out rapidly and "automatically" whenever they are needed, in a manner that requires none of the standard deductive machinery.

Derivations Using KVISTA Implications, Disjunctions, and Negated Conjunctions

When the general purpose binder task has considered all the "explicit" candidate bindings for a given arc, it uses the network's indexing facilities to find nonatomic statements (i.e., implications, disjunctions, and negated conjunctions†) that describe relationships having the same form as the binding being sought. For example, arc B--e-->Builds in Figure 8 is used as the index for finding an implication containing a "build" relationship. Such nonatomic statements are used as the basis for a derivation of the desired binding.

† Double negations, negated disjunctions, and negated implications are eliminated from both the KVISTA and QVISTA by simplification rules.

Applicability Tests

When such a nonatomic KVISTA statement is found, the general purpose binder task carries out an applicability test to determine if the statement can be used to derive a binding for the given QVISTA arc. This test involves unifying (i.e., matching) the KVISTA statement with the QVISTA statement in which the given QVISTA arc is embedded and, when successful, produces a set of substitutions for universally quantified variables that define the "instance" of the KVISTA statement applicable to finding the desired binding.

Several complications in doing the applicability test arise from the fact that neither KVISTA nor QVISTA statements are stored in a canonical form. For example, a negated relationship in the antecedent of an implication can be used to derive a binding for an unnegated form of the relationship, but cannot be used to derive a binding for a negated form of the relationship. In this section, we will discuss the mechanisms in SNIFFER for dealing with these complications.

Parity of Embedded Relationships

The applicability tester needs to determine what the logical signs are of the relationships (i.e., terms) that a given KVISTA statement can be used to prove. For example, the statement $(\sim x \wedge y) \Rightarrow (\sim u \vee v)$ can be rewritten in the following ways:

$$(y \wedge u \wedge \sim v) \Rightarrow x$$

$$(\sim x \wedge u \wedge \sim v) \Rightarrow \sim y$$

$$(\sim x \wedge y \wedge \sim v) \Rightarrow \sim u$$

$$(\sim x \wedge y \wedge u) \Rightarrow v$$

and can therefore be used to prove x, ~y, ~u, or v. If, then, a binding is being sought for a relationship matching x, this statement may be useful in deriving the binding. However, the statement cannot be used to derive a binding for ~x.

The logical signs of the relationships that a given statement can be used to derive correspond to the logical signs that the relationships have when the statement is converted into disjunctive normal form. For example, the disjunctive normal form of the statement given above is $x \vee \sim y \vee \sim u \vee v$. The logical signs of x, y, u, and v in this form of the statement are the same as those that the statement can be used to prove.

During the conversion to disjunctive normal form, only two conversion rules change a relationship's logical sign. Namely:

$$\sim(\sim x) \equiv x \quad \text{and} \quad (x \Rightarrow y) \equiv (\sim x \vee y) .$$

Therefore, we can compute a "parity" for each relationship in a statement to indicate the logical sign that it would have in the statement's disjunctive normal form simply by counting the number of negation spaces and antecedent spaces in which it is embedded. The parity associated in this way with relationships allows a quick determination of whether a given KVISTA statement can be used to produce the desired binding.

Parity of Embedded Variables

The applicability tester also needs to determine what type of quantifier (i.e., existential or universal) is associated with each variable in the statement. For example, the statement $[\sim(\forall x)P(x) \wedge (\forall y)Q(y)] \Rightarrow (\exists z)R(z)$ can also be written:

$$[(\forall y)Q(y) \wedge \sim(\exists z)R(z)] \Rightarrow (\forall x)P(x) \quad \text{and}$$

$$[\sim(\forall x)P(x) \wedge \sim(\exists z)R(z)] \Rightarrow (\exists y)\sim Q(y) \quad .$$

and can therefore be used to prove $(\exists z)R(z)$ or $(\exists y)\sim Q(y)$ or $(\forall x)P(x)$. If, then, a binding is being sought for an existentially quantified QVISTA node that is a participant in an **R** relationship, this statement may be useful in deriving the binding. However, the statement cannot be used to derive a binding for a universally quantified node that is a participant in an **R** relationship.

The quantification types of the variables in the relationships that a given statement can be used to derive correspond to the quantification types that the variables have in those relationships when the statement is converted into prenex normal form. For example, the prenex normal form of the statement given above is $(\forall x)(\exists y)(\exists z)\{[\sim P(x) \wedge Q(y)] \Rightarrow R(z)\}$. The quantification types of x, y, and z in this form of the statement are the same as those that the statement can be used to derive.

During the conversion to prenex normal form, only two conversion rules change a relationship's logical sign. Namely:

$$\sim(\forall x)P(x) \equiv (\exists x)\sim P(x) \quad \text{or} \quad \sim(\exists x)P(x) \equiv (\forall x)\sim P(x),$$

$$\text{and} \quad [(\forall x)P(x) \Rightarrow y] \equiv (\exists x)[P(x) \Rightarrow y] \quad \text{or}$$

$$[(\exists x)P(x) \Rightarrow y] \equiv (\forall x)[P(x) \Rightarrow y] \ .$$

Therefore, we can compute a "parity" for each variable in a statement to indicate the quantification type that it would have in the statement's prenex normal form simply by counting the number of negation spaces and antecedent spaces in which it is embedded.

Note that this is the same rule that is used for computing the parity of relationships! Therefore, this single, computationally simple rule is used to define a parity for both arcs and nodes. The parity associated with an arc indicates the logical sign of the relationship represented by the arc, and the parity associated with a node indicates whether the node represents a universally or existentially quantified variable.

Matching Embedded Structures

The match process carried out by the applicability tester is a generalization of the ramification process described above and is logically equivalent to unification. An attempt is made to find a set of substitutions that will allow two sets of descriptions to match as follows. The QVISTA contains a description of the relationship that is being sought. When the process begins, a KVISTA statement has been found that describes an existing or derivable relationship. The question being considered is whether a relationship that satisfies the description given in the KVISTA statement will also satisfy the QVISTA description. That question is answered by matching the two descriptions. If the match is successful, it defines a set of substitutions (for universally quantified variables) that must be made in the KVISTA description for it to describe a relationship that would also satisfy the QVISTA description. These substitutions produce an "instance" of the KVISTA statement that can be used as a basis for a derivation. For example, if the selected QVISTA arc is part of the relationship Q(a) and the candidate binding is in the consequent of $(\forall x)[P(x) \Rightarrow Q(x)]$, then the instance $P(a) \Rightarrow Q(a)$ would be created.

The basic rules that are used in doing the match are the following. When comparing the two descriptions, an existential in the KVISTA can match only with an existential in the QVISTA or a universal in the KVISTA, and a universal in the QVISTA can match only with a universal in the KVISTA or an existential in the QVISTA. Remember that nodes that are elements of KVISTA or QVISTA spaces are considered to represent existentially quantified entities. These rules are derived directly from the rules for unification. The key observation is that the derived rules should correspond to the rules used for unification in a refutation proof where the match is being done using the *negation* of the query.

As an example of the use of parity during an applicability test, consider again the query shown in Figure 8. The general purpose binder task uses the arc B--e-->Builds as an index to find implication I as a candidate statement to use in the derivation of a binding for the arc Z--e-->Builds. Since both arcs have positive parity, a "builds" relationship derived from the implication will have the desired logical sign. The unification process produces

pairings for nodes Z, ?X, and M, and for the **obj** and **agt** arcs. All the members of those pairs have positive parity except node M. Node M's negative parity indicates that it is universally quantified and can therefore be paired with an existential KVISTA node having positive parity, namely **Ole-Black**. The resulting substitution of **Ole-Black** for M creates the instance of the implication that is used in the derivation.

Extracting Embedded Structures

When an applicable non-atomic KVISTA statement has been found, the derivation that is initiated can be thought of as one designed to "extract" the desired embedded relationship from the statement so that the relation or its negation can be asserted at the top level of the KVISTA and the desired binding can be assigned. For example, if the candidate binding is in the disjunct x of a disjunction $x \vee y$, then finding a solution to the subproblem "prove ~y" will allow x to be asserted and the binding to be assigned.

Rules for Extraction

The derivation is begun by creating the appropriate instance of the KVISTA statement (as indicated by the applicability test) and then applying the following extraction rules:

To extract	Given	Attempt to prove
x_i	$x_1 \vee \ldots \vee x_n$	$\sim x_1 \wedge \ldots \wedge \sim x_{i-1} \wedge \sim x_{i+1} \wedge \ldots \wedge \sim x_n$
$\sim x_i$	$\sim(x_1 \wedge \ldots \wedge x_n)$	$x_1 \wedge \ldots \wedge x_{i-1} \wedge x_{i+1} \wedge \ldots \wedge x_n$
x	$y \Rightarrow x$	y
~x	$x \Rightarrow y$	~y

Note that the extraction rules for negated conjunctions and for implications are merely rewrites of the rule for disjunctions.

If an instantiated implication contains a universally quantified variable, then that variable becomes part of the subproblem produced by extracting either the antecedent or the consequent and is free to be bound during the process of solving the subproblem. For

example, suppose the original implication is of the form $(\forall x)(\forall y)[P(x,y) \Rightarrow Q(x,y)]$ and the instantiation is of the form $(\forall x)[P(x,a) \Rightarrow Q(x,a)]$ If the consequent is to be extracted, then the subproblem has the form "Find an x such that P(x,a)". The assertion that is made when the subproblem is solved is of the form Q(<binding of x>,a).

Nesting

If the relationship being extracted is embedded in a *nesting* of disjunctions, negated conjunctions, or implications (such as the B(x) in $A(x) \Rightarrow [B(x) \vee \sim C(x)]$), then it is necessary to apply a sequence of extraction rules to complete the extraction. The rules are applied "top down" to the outermost disjunction, negation, or implication first, and all the desired extraction rules are applied before any of the subproblems are worked on. Hence, in the above example, a single subproblem is formed consisting of $A(x) \wedge C(x)$. Solution of this subproblem causes assertion of the desired B(<binding of x>). Doing the complete extraction in one step results in the extraction rules being applied only once, makes available to the deductive machinery all the constraints imposed by all the subproblems, and allows the subproblems to be worked on in whatever order seems the most advantageous.

KVISTA and QVISTA Extension Spaces

Procedures that carry out derivations such as the extractions described above require facilities for creating subproblems, making assumptions, and asserting derived results. We have used K-NET's partitioning features to create such a set of derivation facilities that are available for use by any binder task. In particular, provisions have been made for adding spaces (called "extension spaces") to the QVISTA or to the KVISTA in an environment. KVISTA extension spaces are used for making assumptions and for asserting derived results. QVISTA extension spaces are used for expressing subproblems.

For example, consider an environment E1 where K1 is the current (i.e., most recently added) KVISTA extension space and a binder task for the QVISTA implication $x \Rightarrow y$ is initiating a derivation by assuming x and establishing y as a subproblem to be proved. The derivation is

initiated by creating an environment E2 that is an offspring of environment E1, adding to the KVISTA in E2 a new extension space K2 containing a copy of x, adding to the QVISTA in E2 a new extension space Q2 containing a copy of y, and attaching a demon to space Q2 in E2. When bindings are assigned to all the elements of y, the demon is fired in the current environment (i.e., the environment in which all of the bindings are assigned) and in that environment the demon removes space K2 from the KVISTA, removes space Q2 from the QVISTA, asserts $x \Rightarrow y$ in space K1 (the new current KVISTA extension space), and assigns this newly derived result as the binding for the original QVISTA implication.

In order to maintain the relationship between derived results and the assumptions that were used to derive them, the following three rules are used in creating bindings and asserting results.

The first rule is that in each environment only those binder tasks that are seeking bindings for arcs in the most recently added subproblem are allowed to run. This rule helps prevent duplication of effort among environments and assures that effort within an environment created to pursue a particular derivation strategy will not be spent considering other strategies.

The second rule restricts bindings assigned to elements of any given QVISTA space to be elements of KVISTA spaces that existed at the time the QVISTA space was created. In addition to preventing results derived with the aid of assumptions from being used as if they were independent of the assumptions, this restriction is used to maintain the nesting of quantified variables during derivations, as described in the sections below.

The third rule attempts to assure the widest availability of derived results to as many subproblems in as many alternative proof paths as possible. It specifies that each derived relationship be asserted in the newest KVISTA extension space in the set consisting of the space containing the statement used to initiate the derivation and those KVISTA spaces containing elements that were used as bindings to solve the subproblem created by the derivation. This rule allows a derived result whose derivation does not make use of the

assumptions in recently added KVISTA extension spaces to be added in an earlier extension space and therefore be made available to aid in the solution of subproblems created before the assumptions were made.

Use of Extension Spaces for doing Extractions

During the multiple level extraction process, the results of some subproblems may be used in the formation and solution of other subproblems. To make this possible and to prevent a subproblem's results from being used before that subproblem is solved, we maintain the order of the subproblems and their results by putting each one in a separate space and adding those spaces to QVISTA and KVISTA as extensions in the order that the extraction rules are applied. For example, the extraction of R(y) from

$$P(a) \Rightarrow \{(\forall x \in X)P(x) \wedge (\exists y \in Y)[(P(y) \wedge Q(y)) \Rightarrow R(y)]\}$$

will cause creation of the subproblem, prove $P(a) \wedge P(y) \wedge Q(y)$, and will produce the results $(\forall x \in X)P(x) \wedge y \in Y \wedge R(y)$. The results $(\forall x \in X)P(x)$ and the existence of an entity y that is an element of Y cannot be used in the proof of P(a), but can be used in the proof of $P(y) \wedge Q(y)$. This ordering constraint is maintained by creating extension spaces in the following order:

Q1: a QVISTA extension containing P(a) that accepts bindings from the KVISTA that was current when the extraction was initiated.

K1: a KVISTA extension containing $(\forall x \in X)P(x) \wedge y \in Y$, the results of proving P(a).

Q2: a QVISTA extension containing $P(y) \wedge Q(y)$ that accepts bindings from K1 and the initial KVISTA.

Demons are attached to spaces Q1 and Q2 that fire upon completion of the subproblem. Those demons cause spaces Q1 and Q2 to be removed from the QVISTA, space K1 to be removed from the KVISTA, and the cumulative results, $(\forall x \in X)P(x) \wedge y \in Y \wedge R(y)$, to be added to the then current KVISTA extension.

Special Purpose Binder Tasks

The basic SNIFFER includes a collection of functions that form special purpose binder tasks in addition to the general purpose binder described above. The most important of these embody the derivation strategies for queries containing disjunctions, implications, and negated conjunctions. In this section we will describe this collection of functions.

Proving Disjunctions, Implications, and Negated Conjunctions

QVISTA queries are sometimes nonatomic, for example, consider the questions "Were any Mustangs built by Ford?" and "Are all red mustangs owned by playboys?".

The system's special purpose binder tasks for nonatomic statements occuring in the QVISTA apply a strategy of decomposing the statement into alternative simpler subproblems using the following rules:

To Prove:	Generate n subproblems of the form:
$x_1 \vee ... \vee x_n$	Assume $\sim x_{i+1} \wedge ... \wedge \sim x_n$ and prove x_i.
$\sim(x_{i+1} \wedge ... \wedge x_n)$	Assume $x_{i+1} \wedge ... \wedge x_n$ and prove $\sim x_i$.

To Prove:	Generate the subproblems:
$x \Rightarrow y$	Assume x, prove y. Assume ~y, prove ~x.
$(\forall x \in X)[P(x) \Rightarrow Q(x)]$	Create x', assume $x' \in X \wedge P(x')$, prove $Q(x')$. Create x', assume $\sim Q(x')$, prove $\sim[x' \in X \wedge P(x')]$.

As was the case with the extraction rules discussed earlier, the subproblems created for negated conjunctions and for implications are merely rewrites of those produced for disjunctions. Each binder task selects an order in which to produce its subproblems so that the easier ones are produced first.

Each solution to each of the subproblems produces a set of bindings for the entire original statement being proved. Each time one of these binder tasks is run, it creates a subproblem in a newly created offspring environment and reschedules itself in the parent environment. In the offspring environment it adds a new extension space to KVISTA containing a set of assumptions, adds a new extension space to QVISTA containing an expression to be proved, and attaches a demon to the new QVISTA extension space. When the demon is fired by the solution of the subproblem in the QVISTA extension space, it schedules a task that creates bindings for the entire original expression in the then current environment.

If SNIFFER automatically sought inconsistencies between its knowledge base and assumptions that are made, then it would be sufficient to create a single subproblem from a disjunction. Namely, assume the negation of all the disjuncts except one and then attempt to prove the remaining one. However, since SNIFFER does not automatically check assumptions for consistency, we must define two subproblems from a disjunction. Namely, one that specifies a disjunct to be proved, say x_1, and an assumption $\sim x_2 \wedge ... \wedge \sim x_n$; and a second one that consists only of trying to prove that the assumption made in the first subproblem is false. However, the second subproblem is then attempting to prove the equivalent of the disjunction $x_2 \vee ... \vee x_n$, which itself defines two subproblems, etc. Therefore, in fact, n subproblems are defined and they have the form shown in the rule given above. (Note that in an actual proof it is unlikely that many of these subproblems will be created since what appear to be the easiest ones are established first. Only when the initial ones are found to be difficult to solve do others need to be attempted.)

The subproblem formation rules for implications differs from the rule for disjunctions in that the subproblems created from implications may involve universally quantified variables (represented by nodes that occur in both the implication's antecedent and consequent spaces). In each such subproblem, the nodes representing universally quantified variables are "assumed" in the KVISTA extension space created for the subproblem. They therefore represent an entity in the knowledge vista about which nothing is known except the other assumptions made by the subproblem. If the statement to be proved in the subproblem can

be shown to be true about that entity, then it is true for all entities for which those assumptions are true. Such a proof is sufficient to complete the subproblem and therefore prove the implication.

For example, if SNIFFER is attempting to prove that only insecure people own red Mustangs (represented by the implication "if x is a red Mustang, then x is owned by an insecure person"), and the generator for implications creates a subproblem that assumes the implication's antecedent and attempts to prove its consequent, then the assumption for that subproblem would be that some newly created node x' represents an entity that is a red Mustang, and the statement to be proved would be that the entity represented by x' is owned by an insecure person.

Function Applications

In a previous section we discussed the procedural augmentation of K-NET through the use of the **Applications** set. A special purpose binder task creates elements of the **Applications** set in the KVISTA by calling the indicated function with the indicated arguments. This binder task is needed when a subproblem is created consisting of the antecedent of a KVISTA implication that describes a "procedural attachment" to the network. Such subproblems describe an element of the **Applications** set that can be created as soon as bindings are determined for each of the arguments. If the binder task is called before all the argument bindings have been determined, then it attaches demons to the unbound argument nodes that will restart the binder task when all of them have been bound. When all arguments are present, the procedure is called and new network structures are added to the KVISTA to represent the result.

The use of the **Application** set allows a K-NET to explicitly represent meta-relationships between sets of relationships and the procedures that compute them. If a user has no need to represent such meta-relationships explicitly, then procedural augmentation may be realized much more efficiently through the use of user-supplied binder tasks. For example, rather than include the theorem of Figure 6, a specialist for the Sums relationship set may

be added that knows how to call function PLUS and add new information to the KVISTA as described for **Applications.**

Case Analysis Proofs

There is an important class of problems that the deduction mechanisms we have described thus far cannot solve. Namely, those that require a case analysis proof (See Loveland and Stickel, 1973, and Moore, 1975). For example, consider the problem of proving some relation R given a KVISTA containing $(P \vee Q) \wedge (P \Rightarrow R) \wedge (Q \Rightarrow R)$. The mechanisms we have described would go into an infinite loop attempting to solve this problem. What is needed is a case analysis mechanism that will, for example, attempt to prove R in the case where P is true and then attempt to prove R again in the case where Q is true. Since R can be proved in both those cases and the KVISTA contains a statement indicating that either P or Q is true, the problem is solved.

A major difficulty in creating a design for a case analysis proof mechanism is the development of a procedure for defining the cases. Every nonatomic statement in the KVISTA defines a candidate set of cases (e.g., an implication $x \Rightarrow y$ defines the set $\{\sim x, y\}$). Therefore, the problem of defining the cases can be considered to be one of selecting an appropriate nonatomic KVISTA statement.

We are currently experimenting with the following scheme which appears to be an effective way of making the selection. It is based on the observation that for a case analysis proof to be necessary, it must not be possible (or be impossibly difficult) to complete a proof without the case assumptions. Therefore, in each case the assumptions must be useful at some point in the proof. The key, then, to defining the cases for a case analysis proof is in the recognition during the attempt to construct a standard proof of the need for each of the assumptions in some potential set of cases.

Disjunctions in the KVISTA, for example, are selected to be the basis for a case analysis proof by recording each time one of the disjuncts is extracted (i.e., contains a relationship that

matches some relationship in the QVISTA) during a proof attempt. If all the disjuncts of a particular instance of a disjunction have been extracted, then we can conclude that each of the disjuncts would be a useful case assumption and therefore that the disjunction could be the basis for a potentially successful case analysis proof. The same conclusion can be made when both the antecedent and the consequent of an implication or all the conjuncts of a negated conjunction have been extracted.

When such a "fully extracted" KVISTA statement is found, the first common parent of the environments in which the extractions were initiated is found, and a new task is added to that parent environment's agenda to initiate the case analysis proof. That proof attempts to derive bindings for the portion of the QVISTA for which bindings were being derived each time one of the extractions was done. The task creates an offspring environment and in that environment assumes the first case, establishes the statements to be proved in a new QVISTA extension space, and attaches a demon to the new QVISTA extension. The demon does the same thing for the next case. The last demon asserts the statements that have been proved in each case and assigns the appropriate bindings.

Note that in the example given above, any of the three KVISTA statements could be used as the basis for a case analysis proof (e.g., ~P and R is an acceptable set of cases). Our selection procedure could find any one (or all) of them, depending on the order in which new environments are created in the environment tree.

To achieve completeness, one must also consider cases defined by relationships that occur in the initial QVISTA in both a negated and unnegated form. For example, P(x) and ~P(y) occuring in the QVISTA define a useful set of cases {P(<binding of x>), ~P(<binding of x>)} when the binding of x along one proof path is the same as the binding of y along another path.

Concluding Remarks

The goal of this research is to provide a unified system that has powerful, general

mechanisms and that can be made very efficient for solving the most frequently encountered problems in particular application areas. The central idea is to use specialized representations and deduction schemes where they can be effective, while having a logically complete mechanism to fall back on when the special mechanisms fail.

In producing K-NET and SNIFFER, we have attempted to create convenient hooks for adding specialists, and useful building blocks from which those specialists can be constructed. These hooks include the links to procedures (and hence to other representation structures) that are included in K-NET, the pattern-directed strategy selectors in SNIFFER that are capable of invoking user-supplied tasks, and SNIFFER's agenda control mechanism. The building blocks include the taxonomy derivation functions, the unification machinery, and the facilities for manipulating extension spaces.

We plan to continue our experimentation with various specialist routines, both for the rapid handling of particular types of deduction and retrieval and for the extension of the system to include new types of problem solving activities, including reasoning with uncertainties and about processes. Preliminary experience indicates that the facilities provided make this exploration manageable and productive.

An important goal of future work with SNIFFER is to determine the effectiveness of its control mechanisms, particularly the use of INTERLISP coroutines and multiple level agendas. The use of these mechanisms to coordinate multiple types of problem solving activities is of particular interest to us, as is the use of heuristics to guide the allocation of resources among the various strategies that SNIFFER coordinates. We have only begun to gain experience in these areas. However, the modular control structure of SNIFFER and the strong cross-indexing of K-NET provide a very supportive environment for future explorations.

Acknowledgements

The research reported in this paper was supported at SRI by the Advance Research Projects Agency under contracts DAAG29-76-C-0011 and DAAG29-76-C-0012 with the U.S. Army Research Office. The Xerox Palo Alto Research Center has provided support to the first author during the writing and preparation of this paper. Nils Nilsson has been an important contributor to the design of this system, particularly with regard to the deductive machinery. Ann Robinson, Johnathon Slocum, and Mike Wilber have been major participants in the overall implementation effort. Danny Bobrow has provided important critiques of our efforts to describe the work.

References

Bobrow, D. G., and Wegbreit, B. "A Model and Stack Implementation for Multiple Environments". Communications of the ACM, Vol. 16, No. 10, Oct. 1973.

Bledsoe, W. W., Boyer, R. S., and Henneman, W. H. "Computer Proofs of Limit Theorems". Artificial Intelligence Journal, Vol. 3, 1972, pp. 27-60.

Bobrow, D. G., and Winograd, T. "An Overview of KRL, A Knowledge Representation Language". Cognitive Science, Vol. 1, No. 1, Jan. 1977.

Hendrix, G. G. "Expanding the Utility of Semantic Networks through Partitioning". Fourth International Joint Conference on Artificial Intelligence, 1975.

Loveland, D. and Stickel, M. "A Hole in Goal Trees". Third International Joint Conference on Artificial Intelligence, 1973, pp. 153-161.

Moore, R. C. "Reasoning from Incomplete Knowledge in a Procedural Deduction System". MIT AI-TR-347, Dec. 1975.

Reiter, R. "An Approach to Deductive Question-Answering Systems". SIGART Newsletter, Natural Language Interfaces Issue, Feb. 1977, pp. 41-43.

Teitelman, W. *INTERLISP Reference Manual.* XEROX Palo Alto Research Center, 1975.

Walker, D., Editor. "Speech Understanding Research, Final Report, Project 4762". Artificial Intelligence Center, Stanford Research Institute, Menlo Park Ca., 1976.