WORKING PAPER 87

Representing the Semantics of Natural Language as Constraint Expressions

A Master's Thesis Proposal and Progress Report

Richard W. Grossman

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

January, 1975

Abstract

The issue of how to represent the "meaning" of an utterance is central to the problem of computer understanding of natural language. Rather than relying on ad-hoc structures or forcing the complexities of natural language into mathematically elegant but computationally cumbersome representations (such as first-order logic), this paper presents a novel representation which has many desirable computational and logical properties. It is proposed to use this representation to structure the "world knowledge" of a natural-language understanding system.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0003.

CONTENTS

1 The general problem-domain

This project is primarily concerned with the "representation problem": How complex interrelated symbolic information can be structured so that a computer can use it "intelligently". For this project, the information to be structured is the "knowledge of the world" which a person must bring to bear in order to understand some simple natural language utterances (such as found in children's stories), and the "intelligent" use of such information is exactly this process of understanding. Following the lead of other Artificial Intelligence research into the problem of natural-language understanding (primarily Charniak [1972] and McDermott [1974a]), we can consider the computer to have "understood" an utterance if it can answer relevant questions from it.

For example, consider the following story fragment from Charniak's thesis (the line numbers are for later reference):

(1) Fred was going to the store.
(2) Today was Jack's birthday
(3) and Fred was going to get a present.

Now consider some questions which might reasonably be asked: "Why did Fred go to the store?", and "Who is the present for?" For a human reader the answers are trivially obvious. However, in answering these questions, a significant amount of "knowledge of the world" must be used. The first question requires knowing that presents can be gotten at stores (so that "going to get a present" is sufficient motivation for "going to the store"), and the second requires knowing that one often gives presents to another on his birthday, so that the present

which Fred is going to get is ultimately to be given to Jack. Note that none of this knowledge is even implicit in the story text (as it would be if the story contained "Fred was going to the store in order to buy a present for Jack"), and that the integration of the story input with the world knowledge is so smooth that it seems to humans that such information is in the story.

Also, note that line (3) out of context could mean that Fred is whom the present is for (as in "Fred was happy because he was going to get a present"). Thus one also needs to understand the "local discourse context" in order to make sense of the story. Furthermore, there must exist some sort of "global discourse context" which refers to the general topic under discussion. The statement that "the group had no identity" [Winograd 1971] is understood quite differently depending on whether one is discussing mathematics or sociology.

Thus there appear to be several different kinds of knowledge which must be used in order to understand even very simple utterances, without even considering the need for potentially vast amounts of traditional "linguistic" knowledge to handle syntax. Indeed, all the natural-language systems referenced in this proposal use different representations for each sort of knowledge used, and usually contain many additional ad-hoc formats for storing certain kinds of information.

2 The proposed solution-domain

What this project attempts is to express all these different sorts of knowledge using a common representation. The appendix to this proposal gives the philosophical and technical details of the proposed representation as it is currently conceived, and should be consulted if one wishes to verify the claims made for it here. The appendix also compares this "constraint expression" (CE) representations to other ones such as Planner [Hewitt 1968] and first-order logic.

2.1 Experts

The advantages and disadvantges of attempting to use a common representation can be elucidated by comparing such an approach with the "black box experts" approach. The "expert" paradigm says that for each particular kind of problem involved in understanding language one builds a self-contained "expert" program which contains the knowledge necessary to solve it. The experts can call on other experts to solve subproblems or pass the buck. These experts are "black boxes" in that their internal workings are not public: The only access to the knowledge each contains must be through some particular message-passing protocol. Thus for example line (3) might have the expert for the word "get" detect the ambiguity in "get a present" (either "receive oneself" or "get to give to another"), and then it could call the "local discourse" expert to see which of the two interpretations makes the most sense. The local discourse expert might be able to resolve the problem immediately, or it in turn might call other experts (perhaps to

do some deductive inference to decide whose birthday it is).

2.2 Some history

This approach has many advantages (which it is why it is used in the systems of Winograd [1971], Charniak [1972], and McDermott [1974a]), among the most important of which is its modularity: One can attack design problems one-at-a-time by building a new expert whenever some new phenomenon has to be taken care of. Of course one problem is that many of the old experts have to then be changed to include calls upon this new expert -- after a while the interesting details of how the knowledge is used become swamped by the details of the particular communication protocols used between various pairs of experts.

Many years ago (when such experts were being coded in Fortran or machine language), such a "complexity barrier" was quickly reached, which caused much interest in uniform logical representations such as first-order logic. Such a uniform representation avoids the problem of "who knows what" by dumping all known facts into a homogeneous set of axioms instead of hiding them in experts. Unfortunately, first-order logic was designed to axiomatize the foundations of mathematics, not to compute with. Anyway, partly in reaction to the Procrustean bed of first-order axiomatization, Hewitt designed some ways to lubricate the interface between experts and thus free the system's designer from having to consider all the low-level details of inter-expert communication. The micro-Planner language and philosophy which resulted [Sussman, Winograd, and Charniak, 1970] allowed the creation

of such systems as Winograd's and Charniak's.

However, the complexity barrier was again reached -- being free from concern with low-level interaction details, the designer could concentrate on more intricate higher-level interactions, which in their turn became too numerous and involuted to understand. Thus there is again a push for a more uniform type of formalism (See McDermott [1974b] and Winograd [1974] for a deeper discussion of such issues. It is only fitting that the designers of expert-based systems, who have personally run into the complexity barrier, should have good insight into the dimensions of the problem). This of course does not entail a return to mathematical logic -- it is clear that any new formalism must be designed with great attention being paid to its computational aspects (some of which are discussed in the appendix).

2.3 Advantages of a common representation

Apart from the historical cycle of section 2.2, there exist sound methodological reasons for preferring a common representation in the first place. For one thing, one of the main reasons why Winograd's system is so outstanding is that it exploits the strong interactions between the classically separate linguistic domains of syntax, semantics, and pragmatics (world knowledge). Winograd's thesis contains a good discussion of why this is much better than the previously-used techniques of "doing" the syntax of a sentence, taking all 4386 resultant parses, and then "doing" semantics on each one to resolve the ambiguities. By closely interweaving the syntactic, semantic, and

pragmatic analyses, one can vastly reduce the combinatorial explosions. My thesis will not deal with the syntactic aspects of language, in that it will use pre-parsed input such as McDermott and Charniak do. Thus it will not address the syntax-semantics interaction, but it will be concerned with the semantics-pragmatics one (my solution of course being that the "semantics" and the "pragmatics" can interact strongly because they are essentially the same "kind" of knowledge -- linguistic semantics is exactly the world-knowledge which we have about lexical entities, represented in the same manner as other world knowledge. Some details of this are given below).

Also, handling apparently different "kinds" of knowledge within a common representation helps clarify how they interact. Just as using a common representation for space and time allows relativistic physics to explore interactions which would not otherwise be considered, so too does the embedding of different "kinds" of knowledge in a common medium allow an easier examination of their possibly interesting interactions.

3 The conceptual layers

The system to be computer-implemented as a result of this project can be viewed in terms of six layers. The lowest layer consists of mechanisms to do the CE label propagations (see appendix). At this lowest level the system is committed to using a CE representation, assuring the availability of the clean semantics and modularity discussed in the appendix.

The second layer consists of the particular CE nodes which are used. The appendix mentions the class-partition node, the singleton node, the model-theoretic "world" node, the binary relation node, and the typical-member node. These are all implemented in terms of the previous level, and any new node types which may turn out to be necessary can be similarly implemented. This layer is responsible for the logical expressive power discussed in the appendix.

The third layer consists of "low-level constructs" such as might be found as "primitives" in some logic. The appendix mentions taxonomies, boolean connectives, Conniver-like "contexts" [McDermott 1973], domain-range restrictions, transitive relations, and N-ary relations. These (and any new ones which are needed) are implemented directly in terms of simple combinations of the level-two nodes.

The fourth layer consists of "high-level constructs" which make a strong commitment regarding the ontology of the universe. Some of these constructs are discussed in section 4. They consist of such things as frames, defaults, states, events, and the representation of time. This layer also provides a mapping from linguistic entities (such

as nouns, adjectives, verbs, and case frames) into the appropriate high-level constructs.

The fifth layer consists of the particular world-knowledge to be encoded in terms of layers three and four. For example, to handle the story fragment given in section 1, it is necessary to have knowledge about children, birthdays, birthday parties, gift-giving, gifts (such as where they are gotten), stores, "going", and "getting". This knowledge will of course be expressed in a form which does not bias it too heavily in favor of the particular examples used: For instance, there will be knowledge of "going" in general, not just knowledge of "going to the store". Section 4 gives some detailed examples of such knowledge. The primary commitment of this layer is to the particular domains of discourse used for examples in the thesis.

The sixth and topmost layer consists of a program to "use" the CE data-base. The appendix discusses the CE data-base as a rather passive storehouse of knowledge, which must be prodded by some user in order to actually accomplish anything. In terms of the appendix, this layer is responsible for creating new fragments of CE network (to represent the story fragments), for setting up the different kinds of initial labeling conditions, and for reacting to the results of the label propagations (such as contradictions). This layer is the interface between the CE data-base and the outside world. Thus this layer's primary commitment is to the particular story fragments which will be used as examples in the thesis (The kinds of high-level processing which will be applied to these examples is illustrated in

section 4). Thus it should be possible to dissect the final system and say that the stuff in layer six is particular to this one thesis, but that the rest is of more general applicability.

Note that this topmost layer is fundamentally different from the others. Layers one through five represent increasing levels of structural complexity within the CE data-base. The top layer however represents the the user of this data-base. It also provides a convenient place to house the various ad-hoc processes which will doubtlessly turn out to be necessary. Indeed, the ad-hoc stuff is exactly that knowledge which can not (yet) be represented within the CE data-base. For example, the fact that the high-level constructs do not yet include "procedures" means that all procedural knowledge must be housed here (or else that "procedures" must be implemented within levels one through four). If a large mass such of ad-hoc material does in fact accrete here, it will indicate the areas in which the existing data-base scheme is deficient and thus provide a focus for possible future work.

4 Using the high-level constructs

This section presents a rather detailed analysis of the kind of knowledge structures which are needed in order to understand story fragments such as the one given in section 1. What is desired is to represent in the CE data-base both the hearer's pre-existing knowledge (about birthday parties, etc.) and the knowledge which is conveyed by the linguistic constructs in the story. Consider the sentence in line (1): "Fred was going to the store."

4.1 Entities (noun groups)

Line (1) describes an "event" (Fred going to the store) involving two "entities" (Fred, and the store) and a time reference (past progressive). Entities are represented by points (classes) in the CE network. In this case, both entities are singular so they are represented as singleton object classes, which we can call Fred-37 and store-38. It is known that store-38 is indeed a store, so we need the fact that store-38 is a subclass of all STORES. This is handled using a partition node, which we will abbreviated here by [store-38 → STORES] -- the brackets serve to delimit pieces of CE network from this surrounding text. The fact that store-38 is <u>the</u> store instead of <u>a</u> store gets into rather complex linguistic issues which will not be dealt with here (but which will be discussed in the thesis). Now all we know about Fred from the story is that it is an object named "fred": The is handled by a binary relation node which says that the NAME-OF Fred-37 is "fred". As a linear notation for this we will say

[<NAME-OF Fred-37 fred>], where "fred" is a point which represents the name "fred" (as opposed to representing the objects which have that name). Note that none of this says that Fred-37 is human or even is even a physical object -- such facts are part of the hearer's pre-existing knowledge which is already encoded in the data-base.

4.2 Events and the case-frame hierarchy

The event "going to" is more complex than the static entities. One way of representing such an event is to use logical predicates, such as (GO-TO Fred-37 store-38), which is in fact the approach taken by Charniak's and McDermott's systems. Note, however, that there is considerable question as to how many arguments such a GO-TO predicate takes. In "Fred was going from his home to the store" we seem to need a "source" argument; in "Fred was going to the store via Main Street" we need a "path"; in "Fred was going to the store rapidly" we need a "speed"; and so on. Now, this "variable number of arguments" problem disappears in the CE representation: An N-ary relation is always represented as a series of binary relations between the instance of the relation and its arguments. For example, the class GOINGS represents all instances of the "going" relation. Then the instance in line (1) is in this class, so we make up a new name (going-39) and state that [going-39 → GOINGS]. To associate the arguments, we say something like [<OBJECT-OF going-39 Fred-37>], [<DESTINATION-OF going-39 store-38>], and so on (such as [<SOURCE-OF going-39 whatever>] if required).

Now in fact such structures are called "case frames" [Fillmore

1968] and are being used both by some linguists and some AI language-understanding projects (see section 5). Note that the structure of the CE representation itself (which is based on rather abstract, non-linguistic considerations) has pretty much forced the re-invention of such things. This kind of serendipity has occurred often with CE, and is one of the reasons I am confident that this is a productive line of research.

Now, it turns out that many verbs other than "going" have a similar case structure in that they can take a source, destination, path, object, etc: Consider "coming" "taking" "moving" "sending" and "receiving". To capture this generalization, we can create the common class PTRANS (for "physical transfers", following Schank [1972]), and have all these particular kinds of physical-transfer relations be subclasses. In addition, each of the different kinds of PTRANS may have additional information associated with it by using additional arguments.

Furthermore, it turns out that all events have some structure in common. Thus at the top of the hierarchy of events should be something that captures the general structure of all events. Calling this top point DOINGS, the common structures include "intensity" modifiers ("He did it with vigor"; "He did it rapidly"), the time of the event (which can be very complex, as in "he has been wanting to do it for a week, already"), and the cause of the event.

4.3 Events and difference descriptions

However, we still need to deal with the internal structure of events: What does it mean for an object to be PTRANSing from a source to a destination? In general, I choose to model the meaning of events in terms of state changes. A state can be represented by a CE "world": Each state "contains" the facts which are true in that state. Thus each event has associated with it a before state and an after state: The before-state contains those facts which are true before the event (but not after), and the after-state contains those facts which are true after (but not before). Anything not mentioned in these states is thus known to be unchanged by the event. So part of the structure for PTRANS affirms that for every ptran-N which is in PTRANS, the LOCATION-OF the OBJECT-OF ptran-N is the SOURCE-OF ptran-N in the before-state, while it is the DESTINATION-OF ptran-N in the after-state. That is, the meaning of an instance of PTRANS is that the object changes location (where LOCATION-OF a physical object is a representation of its physical location). The "for every ptran-N" in the above description means that a typical-member node is used to say this about a typical ptran-N, so that the description then applies to all instances of PTRANS (which includes all instances of GOINGS, TAKINGS, etc).

4.4 Events, Minsky's frames, and defaults

The general structure which has evolved in this discussion so far can be described as follows: A complex relation (such as "physical

transfer") is represented via a class of all instances of it (PTRANS). A typical-member node is used to describe the structure of a typical member of the class (ptran-N). The attributes of the typical ptran-N are specified by a series of binary relations (such as OBJECT-OF) which specify the "arguments" to the complex relation. Furthermore, the classes can be arranged in a hierarchy (or even more complex structure) so that structurally similar complex relations can have their similarities factored out and made explicit. This structure is in effect an implementation of Minsky's [1974] idea of "frame systems", so here is another instance of serendipity.

An additional aspect of Minksy's frames is that they employ "defaults" -- each argument slot may have associated with it a "loosely bound" (ie. easily replaced) default value, which is to be used in the absence of any other information. This use of defaults allows quick "jumping to conclusions" with the later possibility of filling things in more cautiously. The notion of "default" is closely related to those of "exception" and "probability", and in fact the same CE mechanism can handle all three. This mechanism uses the CE "world" construct to impose an ordering on the "reliability" which the system assigns to any particular fact. The ordering is imposed using a subclass hierarchy, such that world W1 represents more reliability than world W2 iff W1 contains W2 (ie. is a superclass). Section 5.4 of the appendix discusses such hierarchies as a means of expressing orderings with respect to transitive relations such as "on". Now, individual facts are tied into this hierarchy using world nodes, such that if Wf is the

world of some fact and Wi is a world in the hierarchy, then [Wi → Wf] means that Wf's fact is at least as reliable as Wi. That is, if what Wi represents is reliable enough for a given purpose, then so is the fact that Wf represents.

This can be made clearer by looking at the processing which goes on within such a hierarchy. Since the higher reliability is towards the top of the hierarchy, a +w label (see appendix) placed on a point representing a given level of reliability (such as W2) can propagate upwards to all levels of reliability which are known to be "higher" than it (such as W1). Furthermore, any world node having its Wf contain such a W2 (as in [W2 → Wf]) will also be reached by the +w. Thus we have hierarchy of "contexts" such that if some Wi is "enabled" by putting +w on it, then all higher contexts and all facts implied by such contexts will also be enabled; otherwise these facts will not be enabled. Now, the +w label which is actually used is "+inf", which means that these facts are to be enabled for the duration of the current inference (appendix, section 4). Thus by starting this +inf at different points in the reliability hierarchy, different sets of facts can be enabled for use by the inference. So starting the +inf at a lower point in the hierarchy will enable more facts than starting it at a higher point (since a lower point such as W2 automatically propagates its +inf to all higher points, but not conversely).

Now reconsider the issue of probability. By associating less-probable facts with worlds lower in the hierarchy, they will only be enabled when +inf starts at least that low. Thus the lower down the

+inf starts, the more "gullible" the system is in terms of allowing the use of less-reliable (low-probability) facts. This then allows the user of the CE data-base (ie. layer 6) to set whatever level of gullibility it desires. So for first-pass crude processing it can use a very gullible setting (which enables all the frame defaults), and then switch to a less gullible setting whenever its suspicions are aroused (as might be the case if there were a contradiction between a frame default and other data). As more and more is known about something, the system can afford to be less gullible in terms of accepting the truth of vague generalizations (as embodied in the defaults), since it has more "quality" data to work from and can afford to ignore the defaults. This same mechanism can handle hedged facts, such as "Most birds can fly" or "The rain in Spain stays mainly in the plain" -- the facts are simply given a world node which is assigned the proper place in the reliability hierarchy.

4.5 Disambiguation -- Domain-Range restrictions

The preceding parts of this section have dealt with representing information after it has been fully translated into the proper CE structures. This final part touches on the issue of semantic disambiguation in order to examine part of the translation process itself.

Consider the phrase "going to" in "Fred was going to the store". Here, it clearly means a form of PTRANS, where the destination is the physical location of the store. But in "Fred was going to his

doom", the destination is more a state (of Fred) rather than a physical location. In "Fred was going to help me", the destination involves participating in some other event (ie. helping me). Now a standard linguistic analysis of this would conclude that there are (at least) three different senses of "going to", and that it must be fully disambiguated before one can begin to understand a sentence in which it occurs.

In opposition to such an analysis, I claim that all three senses of "going to" are more similar to one another than they are different -- much of the information conveyed by "going to" is the same regardless of the context and "sense" in which it is used. This means that the part of the meaning of "going to" which is common to all three senses can be used as soon as "going to" is encountered -- one need not wait for the phrase to be disambiguated before one can start drawing some useful conclusions. For "going to", the major common meaning involves the existence of a state in the future involving the object (Fred) and the destination. Thus as soon as "going to" is encountered, it is immediately known that its meaning involves such a future state.

Now, given that much, it is still necessary to distinguish among the three senses as more information comes in. To name the three disambiguated senses and the ambiguous one, let GOINGS-0 be the ambiguous form, and GOINGS-PHYSICAL, GOINGS-STATE, and GOINGS-EVENT be the three senses in the order presented above. Part of the hearer's world knowledge includes the fact that GOINGS-0 is partitioned into the three senses: Every instance of a frame for GOINGS-0 is an instance of

exactly one of the disambiguated senses. The facts which are common to all the senses (such as the use of a future state) are hung off GOINGS-0; the facts peculiar to one sense are hung off that sense. When "going" is first encountered, a frame is set up as an instance of GOINGS-0. Then "disambiguation" of this involves assigning the frame to one of the more specialized senses. This can be accomplished using domain-range restrictions (appendix, section 5). These would say that [<DESTINATION-OF GOINGS-PHYSICAL PHYSICAL-OBJECTS>], [<DESTINATION-OF GOINGS-STATE STATES>], and [<DESTINATION-OF GOINGS-EVENT EVENTS>] respectively for the three senses. That is, the destination of a GOINGS-PHYSICAL is always a physical object, and similarly for the other two. As more information comes in, these restrictions force the choice of one of the three senses. Suppose the destination turns out to be "the store". Most things which the system knows about will be arranged into a global taxonomy (such as figure 2-1 in the appendix). In such a taxonomy, physical objects, states, and events will be mutually exclusive. Now, "the store" is a store, which is indeed a physical object. Thus it is <u>not</u> a state or an event, so the second and third choice for the partition of GOINGS-0 are eliminated (they receive -x labels). Thus GOINGS-PHYSICAL, the only remaining choice, is forced. All this happens within the framework of CE label propagations -- no other kind of processing is necessary.

The paradigm that comes out of such examples is that of "incremental disambiguation" -- instead of something being categorically "unambiguous" or "ambiguous", it is known to a greater or

lesser amount of precision. Indeed, the whole idea of "unambiguous" becomes faintly ridiculous -- there is always more which can be known about any particular thing. The important criterion is that enough be known about the thing in order to be able to make useful inferences.

5 Relation to other work

One purpose of this research is to take several of the ideas about language understanding (and AI in general) which have appeared in the last few years and to embed them in a semantically clean representation, so that their interactions can be studied and experimented with.

The general CE philosophy (layer 1) that computations should be performed by pushing labels around a network has been derived from the works of Lamb [1966, 1969], Quillian [1967, 1969], and Waltz [1972]. Since all three of these systems involve local constraint propagation, they share with CE the feature of modularity in both the logical and computational senses (see appendix). However, unlike Lamb's and Quillian's systems, CE has a clean semantic in terms of exactly what the primitive labeling operations are, and how they interact. In addition, unlike all three, CE has sufficient expressive power to handle a very wide variety of logical inferences in a reasonably simple manner. Of course these three systems were designed to meet different goals from those embodied in CE, and to work in different domains. Thus this research can be seen in part as an extension of such techniques to a more complex domain. Also, like Waltz, I am attempting to take a complex problem ("common sense" logical inference) and reduce it to an essentially trivial algorithmic procedure, much as Waltz did for his area of scene analysis.

The choice of the domain of natural language understanding in general and childrens stories in particular represents a conscious

attempt to build on the work of Charniak [1972]. The two major differences between this research and Charniak's are: (1) His was much more ambitious, addressing a wide variety of linguistic and world-knowledge issues, and (2) This research is interested in doing things cleanly, while Charniak's pioneering efforts were quite properly devoted to doing things at all. Many surface differences between Charniak's work and this research ultimately derive from my desire for "clean" solutions. That is also the major difference between this research and that of McDermott [1974a], who also created a Charniak-like system. For one thing, Charniak and McDermott used "procedural" representations because such representations provide a lot of usable power; I chose a "declarative" representation because it facilitates understanding the underlying semantics of the representation and the interactions between different chunks of knowledge.

Since Minsky's [1974] frame theory is also concerned with such interactions from the standpoint of a basic representation, it is not surprising that this research directly relates to his. Indeed, a large portion of the work to be done on this project involves a reasonably large-scale implementation of some of his ideas which have heretofore not been embodied in executable computer programs.

The use of "difference descriptions" for representing events is derived from Winston's thesis [1970]. Using such descriptions makes explicit the possibilities for an event changing something, and so may be a good solution for the "frame problem" of McCarthy [McCarthy and Hayes 1969].

The use of case analysis for verbs is currently a topic of interest within the AI community. Martin's [1974] work on natural language is a prime example, as well as the in-progress M.I.T. theses of Mitch Marcus and Cal Drake. Martin's system also exploits the semantic commonalities of different word senses discussed in section 4.5 -- the meaning of "put an idea in one's head" has many similarities to that of the physical sense of "put into".

Appendix --

Data Base Applications of Constraint Expressions

CONTENTS

1 Introduction

1.1 Goals

This appendix introduces a representation for information, based on "constraint expressions" (CEs). This CE representation has evolved (over the last year or so) in response to the following three design goals:

First, The representation should be "natural" with respect to the domain of information which is to be expressed. By that fuzzy, overworked term I mean that the representation should be a positive help in structuring the domain's information in useful ways, instead of being a hindrance. In particular, things which are structurally similar in the domain should be structurally similar in the representation. For example, LISP is a far more natural representation for recursive symbol manipulation algorithms than a universal Turing machine is, in part because LISP's control structures allow one to write structurally similar programs for structurally similar algorithms: All algorithms with the same general form "chew down a list and perform a common operation on each sublist" can be naturally written using the same recursive structure. With Turing machines, however, one would be hard pressed to find any similarity among the representations of such algorithms. Of course, in a different domain (such as automata-theoretic proofs) the Turing machine might turn out to be the more "natural" representation. Therefore, since "naturalness" is a function of the domain, it is necessary to explain the intended domain for the CE representation.

Broadly speaking, the intended domain is that of "natural language understanding." In particular, the CE representation forms the basis of my forthcoming S.M. thesis, which deals with the understanding of the sort of discourse found in children's stories. Now, interesting structural similarities show up in natural language because the same locally meaningful structure (such as a noun phrase) can occur as a constituent in many different higher-level structures. Now, since the meaning of a structure (such as "Macy's employees") is pretty much independent of where this structure fits in to some higher-level structure, the representation of such a meaning should have this same property. That is, "Macy's employees" should be represented the same in all of: "John Doe is one of Macy's employees"; "Is John Doe one of Macy's employees?"; and "List the Macy's employees who will retire this year." By representing the constituents (such as "Macy's employees") the same, the structural similarities among these three sentences are directly reflected in the representation. This emphasis on "local meaningfulness" means that the CE representation is highly modular, and that all channels for interaction are represented explicitly in the data-base. Also, the fact that statements, questions, and commands are all represented in the same manner is one of the powerful features of the CE representation -- this appendix mentions some of the limitations found in some representations which do not have such a feature.

The second design goal is that the representation must have sufficient expressive power, in a computational as well as abstract

logical sense: It does no good to have an abstractly "powerful" representation if it is computationally difficult to use it. First-order logic is the canonical example of a computationally poor representation -- the time required to compute an inference grows exponentially with the complexity of the inference and the amount of available information (in the form of axioms). This seems rather strange, in that the more one knows, the easier it should be to answer questions. Thus the emphasis in this appendix is more on the computational aspects of the CE representation, rather than its purely logical ones.

However, this is not to slight the desirability of having a deep theoretical understanding (logical as well as computational) of the representation. Sadly, it often seems that there are two opposing factions working on the representation problem: On one side are some logicians, who are almost exclusively concerned with achieving a deep understanding of their representations; on the other side are some computer scientists, who concentrate on achieving computational power without much understanding of the underlying semantics. The disadvantage of such computer hacking is that when something works (or, more often, doesn't), it is very difficult to glean more general information such as "why", or "where exactly is the weak link."

Indeed, I personally made the greatest progress with CE when I stopped hacking for a while and attempted to discern the basic semantics of what was really happening. Doing this was facilitated by the third design goal, which is that the basic representation should

have a clean semantic base by using the possibly complex interactions of a few simple primitives, rather than on primitives which are themselves inherently complex. The fact that the representation concentrates on "local meaningfulness" is a big help here, since that implies that the global semantics of an expression can be easily analyzed in terms of the local semantics of the primitives. (Another reason for the third design goal is that I eventually plan to use the representation for complex learning tasks, which would be nigh well impossible if the primitives are too complex. The features of CE which relate to learning are beyond the scope of this document). Finally, a system with a simple semantic base is much easier to implement (Just compare the compilers for LISP and PL/I!).

1.2 Details

Section 2 presents the basic ideas behind CE by using it to represent taxonomies. This representation for taxonomies is compared with first-order logic, Planner [Hewitt 1968, Sussman 1970], and Codd's relational data-base scheme [Date & Codd 1974].

Section 3 slightly extends the CE representation of section 2 to handle all Aristotelian syllogisms. Procedures for finding all the individuals with a given property are presented.

Section 4 extends the representation of section 3 to handle all of propositional (0-order) logic. The nature of the model theory for this logic is examined, with references to the problems of consistency and completeness.

Section 5 extends section 4 to handle binary relations (in terms of mappings). It is shown how the CE representation can easily handle transitive relations. The obvious extension to N-ary relations is discussed. Some examples are presented, and compared with Codd-like representations.

Section 6 extends section 5 to handle arbitrarily nested quantifications, giving CE the logical expressive power of omega-order logic and the attendant necessity for brute-force processing. Some brute-force examples are presented, and it is explained why such complexity has up to this point not been needed.

2 Taxonomies

2.1 Other representations

Figure 2-1 shows an example of a taxonomic classification of physical objects. The asterisks (called points) represent classes of objects. The constructions connecting the points are partition nodes -- each signifies that the classes below the bar form an exclusive and exhaustive partition of the class above the bar. (The "..." indicates that a partition node has other subclass points which are not shown). For example, the class of LIVING things is partitioned into exactly the three subclasses of PLANTS, ANIMALS, and WEIRDIES. Now, the visual connectivity of the figure makes it very easy for humans to draw certain kinds of conclusions from it. For example, it is obvious that all redwoods are plants, since the class of redwoods is a subclass of trees is a subclass of plants. Also, no redwoods are bacteria, since all redwoods are plants, all bacteria are weirdies, and plants and weirdies are mutually exclusive.

In general, the problem of representing taxonomies is an important one, for at least three reasons. First, humans do it well and do it often: Biologists classify specimens; programming language designers define data-types ("numbers" are partitioned into "integer," "real," and "complex."); operations analysts (among others) use decision trees; and the notion of "subclass" alone appears often in various formalisms -- for example, Codd [Date & Codd 1974, p. 24] would like to be able to state that "everyone who supplies any part is a known supplier".

Noting that Codd's formalism has trouble even with simple subclass, let us consider how some other formalisms handle taxonomies. This is the second reason for interest in taxonomies: Most formalisms do it poorly. Consider first-order logic. The partitioning of LIVING things into PLANTS, ANIMALS, and WEIRDIES can only be done by something like this:

$$\forall x\ [\mathrm{LIVING}(x) \equiv \mathrm{PLANT}(x) \vee \mathrm{ANIMAL}(x) \vee \mathrm{WEIRDIE}(x)]$$
$$\forall x\ \neg[\mathrm{PLANT}(x) \wedge \mathrm{ANIMAL}(x)]$$
$$\forall x\ \neg[\mathrm{ANIMAL}(x) \wedge \mathrm{WEIRDIE}(x)]$$
$$\forall x\ \neg[\mathrm{PLANT}(x) \wedge \mathrm{WEIRDIE}(x)]$$

The first line of this seems alright -- the problem is that we need all those negation assertions. Indeed, a theorem prover would find it not completely trivial to prove "no redwoods are bacteria", which is annoying since there clearly does exist a trivial procedure ("following the lines", as was done above). Representing the above facts in Planner is even worse, because Planner-like languages do not even <u>have</u> a built-in negation mechanism, usually relying on something to the effect that "if I can not prove it is true, then it is false."

The third and most important reason for discussing taxonomies here is that the CE representation (which is about to be explained) <u>does</u> in fact represent and process them efficiently.

2.2 The CE representation

Consider the process of putting individual objects into the various classes (represented by points) in figure 2-1. Let names in all lowercase letters denote particular objects (while uppercase names still denote classes). We will "put objects in classes" by labeling the

classes with the names of the objects: For an object "x" and a class "C", let the label "+x" on C mean that x is in C; let the label "-x" on C mean that x is not in C. Clearly, it is contradictory to label C with both "+x" and "-x"

Now, if the diagram in 2-1 consisted of just the points, with no partition nodes, there would be complete freedom in labeling any particular point "+x" or "-x" (but not both) without fear of contradiction. By including the partition nodes in the diagram, what we do is constrain the possible patterns of labeling the points, hence a partition node is a kind of "constraint expression". For example, if +x is on TREES, it is constrained to also be on PLANTS (since all trees are plants). Thus the more we know (in terms of additional constraints), the fewer arbitrary interpretations we can make. In general, figure 2-2 enumerates the four ways (t1 thru t4) in which a partition node can apply a constraint and force the propagation of label assignments:

(t1) If one subclass is labeled +x, then the superclass must be labeled +x and all other subclasses must be labeled -x. For if an object is in one partitioning subclass of the superclass, then it is clearly in the superclass itself, and can not be in any of the other mutually exclusive subclasses.

(t2) If the superclass is labeled -x, then all subclasses must be labeled -x. For if an object is not in the superclass itself, it can not be in any subclass.

(t3) If the superclass is labeled +x, and all-but-one of the

subclasses are labeled -x, then the remaining subclass must be labeled +x. For if an object is in the superclass and is definitely not in all-but-one of the subclasses, then it must be in the remaining subclass.

(t4) If all the subclasses are labeled -x, then the superclass must be labeled -x. For if an object is not anywhere in the exhaustive partitioning of the superclass, then it is not in the superclass.

Note that t1 and t3 are in some sense "duals", as are t2 and t4. Also note that another way of looking at the partition node constraints is to say that a partition node makes certain labelings illegal. Figure 2-3 shows the three basic illegal labelings around a partition node, from which one can indeed derive t1 thru t4.

3 Syllogistic logic

3.0 Introduction

The purpose of this section is to show in rather gory detail some of the interesting processing that can be applied to a CE network in order to answer questions. The framework of Aristotelian syllogistic logic was chosen for this section because the inferences are easy for humans to follow, and the inferences are easy for the CE representation to compute.

3.1 Universal affirmations: "all A are B"

Given the label propagating machinery of the previous section, it is trivial to do syllogistic inferences of the form "all REDWOODS are TREES, all TREES are PLANTS, thus all REDWOODS are PLANTS" (see figure 2-1). One method is to initially label REDWOODS with +x. Then, by t1, label TREES with +x. Finally, by t1 again, label PLANTS with +x. Now, since "x" is an arbitrary redwood, we know any arbitrary redwood is a plant; ie. all redwoods are plants.

Since the use of such "arbitrary" objects can be confusing, I prefer a different method of doing such inferences. Assume that the statement is false. Thus there is at least one redwood which is not a plant. Without loss of generality, call it "x" -- "x" is now that particular (although unknown) redwood. So, label REDWOODS with +x (since by hypothesis x is a redwood), and label PLANTS with -x (since by hypothesis x is not a plant). By propagating the labels as above, we get +x on PLANTS. But, this is a contradiction: x can not both be a

plant and not be a plant. Thus the initial hypothesis must be incorrect, so indeed all redwoods must be plants. Note that instead of using t1 to push the +x from REDWOODS to PLANTS, we could just as well have used t2 to push the -x from PLANTS to REDWOODS, or we could have used both and let the contradiction occur somewhere in between (at TREES, in this case).

3.2 Universal negations: "no A are B"

Similarly, label propagation can handle syllogisms involving negation: "All trees are plants, no weirdies are plants, thus no trees are weirdies (and no weirdies are trees)". As above, assume the conclusion is false. Then there exists something which is both a tree and a weirdie. Call it "x". Label both TREES and WEIRDIES with +x. Then, by t1, propagate the +x to PLANTS. Then, by t1, +x on PLANTS constrains there to be -x on WEIRDIES. Since WEIRDIES is now both +x and -x, we have the contradiction, so the hypothesis is false. Thus no trees are weirdies. As above, different orders of applying constraints can cause the point of contradiction to occur at different places.

3.3 Existential affirmations: "some A are B"

Consider: "Some employees are felons, all felons are crooks, thus some employees are crooks". As above, the goal is to represent the premises as a network of constraint expressions, and then prove the conclusion by setting up an appropriate initial labeling. However, a network consisting solely of partition nodes can not make existential

statements: Given a network with a point representing the class of EMPLOYEES, nothing prevents that class from being completely empty. The partition node provide constraints of the form "if an object is in one of the subclasses, then it must be in the superclass and not be in any other subclass" -- nothing says that there has to in fact be an object there at all, but only that if there is one then it must be in the superclass, etc.

Thus we need a new kind of constraint, one which insists that there in fact be an object (or objects) in a particular class. We do this by making special note of classes containing exactly one object. Such singleton classes are represented by network "points" which are drawn as small squares. As an example, figure 3-1a says that there are exactly three EMPLOYEES -- Smith, Jones, and Lee. In keeping with the convention of naming objects with lower-case letters, points representing singleton classes are given lower-case names (except perhaps for the first letter).

Two details need to be emphasized about this particular formalism for objects. First, like OWL [Martin 1974], this formalism does not consider objects to be set-theoretic elements of classes, but rather to be classes themselves (with the added property of being singletons): "The object 'Smith' is in the class EMPLOYEES" is represented as a subclass relation and not a class-membership one. Thus this is a "one level" system -- it is impossible to have classes whose 'elements' are themselves classes, whose elements in turn .... This eliminates a counter-intuitive source of conceptual complexity

(consider Russell's paradox), without harming the system's usable power.

The second detail is that two distinct object-points in a network do not necessarily represent distinct objects. Just as two non-object classes can be the same (for example, when each is constrained to be a subclass of the other), so may two object classes be constrained to be the same. The fact that two (or more) objects are distinct must be represented explicitly. In figure 3-1a, the three employees are constrained to be mutually exclusive with each other, and hence are distinct. In figure 2-1, any objects which appear at the bottom of such a taxonomy will all be distinct by virtue of the hierarchy of mutual exclusions which connect them. Thus there is little overhead in requiring explicit representation of object distinctness, while allowing for the case that superficially 'distinct' points in a network really represent the same object (such as "the morning star" and "the evening star").

Now, figure 3-1b represents the premises that some particular employee (arbitrarily named 'emp1') is a felon, and that all felons are crooks. Note that in a representation such as Planner or Codd's, two atomic objects are always considered to be distinct if they have different print-names. Thus "emp1" would considered to be distinct from all of Smith, Jones, and Lee; while in 3-1b it is clear that "emp1" is in fact exactly one of those three. Now, since all known objects are represented explicitly in the CE data-base, the problem of showing that <u>there exists</u> some employee who is a crook is equivalent to <u>finding</u> one

(or more) such objects. That is, we need to find an object which is both an EMPLOYEE and a CROOK. We of course know that the desired object is 'emp1', but the problem is to implement the necessary inferences in terms of label propagations. Several such implementations are possible.

The obvious brute-force technique is to enumerate all the objects in the data-base and test each one. Testing an object is easy: To see if an object is an employee, mark the object +x, mark EMPLOYEES -x, and look for a contradiction during the label propagation. Similarly, we can test the object to see if it is a crook. Of course, for a large data-base on conventional hardware such a scheme would be much too expensive.

One way to eliminate this brute-force enumeration in some cases is to keep a "short term memory" of objects recently mentioned or created. Then if the conclusion is presented immediately after the premises are given, 'emp1' can be tried immediately. However, in a real data-base application it is usually necessary to answer questions using information that is not explicitly mentioned, and hopefully without using brute-force enumeration.

So, instead of enumerating everything known and testing each one for employeehood and crookedness, it would be much better to enumerate only the known employees (or known crooks), and then test each of these for crookedness (or employeehood). Enumerating all the objects in a class (such as EMPLOYEES) is easy: Label EMPLOYEES with -x, then and every object to which a -x propagates is a known employee.

In this case, the -x will propagate (using t2) to all of Smith, Jones, Lee, and empl. To show that every object which is reached by a -x is in fact an employee, attempt the inference for the general case: Label any found object (Jones, for example) with +x, and label EMPLOYEES with -x. Then clearly a contradiction will occur, since the -x label can propagate from EMPLOYEES to the found object (which is labeled +x). Thus the object is a subclass of EMPLOYEES (by the reasoning used for showing "all A are B"). This proves that all found objects are indeed employees, but does not prove that all employees will be found -- section 4 contains a brief discussion of this issue of "completeness".

A final way of finding crooked employees is to do the enumerations in parallel: Label EMPLOYEES with -x, and CROOKS with -y. Any object which is reached by both these labels is by necessity both an employee and a crook. Note that this is less efficient for sequential hardware than the previous technique is, but it is not hard to envision various asynchronous processing schemes (using cellular automata) which make the parallel-enumeration approach win.

3.4 Existential negations: "some A are not B"

The machinery of the previous subsection can also handle syllogisms such as: "Some employees are felons, all felons are crooks, no good-guys are crooks, thus some employees are not good-guys". Figure 3-1c adds this constraint that no good-guys are crooks (ie. they are mutually exclusive classes). All the techniques for existential affirmations apply here too. For example, we can enumerate all

employees as before, and test them to see if they are not good-guys (by deriving a contradiction from labeling both the found employee and GOOD-GUYS with +x). This is analogous to the handling of universal negations in 3.2. Similarly, to enumerate the non-good-guys, start a +x at GOOD-GUYS and use all objects which are reached by -x. The proof that this works is isomorphic to that in 3.3.

4 Propositional logic

4.0 Introduction

This section uses the CE representation to handle the full range of propositional (0-order) logic. The same processes which apply for simple syllogisms also apply here, although the complexity of the interacting label propagations increases as the complexity of the data-base does.

4.1 Model theory: Propositions as classes

Propositional logic deals with 'propositions' (abbreviated by the letters P and Q), each of which has a 'truth value' (true, or false). The 'propositional connectives' (negation, conjunction, disjunction, implication, etc.) are considered to be functions on the domain of truth values: For example, the compound proposition "P and Q" (or P∧Q) has a truth value of "true" iff both P, Q have truth values of true. We shall see later on how this somewhat obscure notion of "truth function" can be better interpreted in terms of "constraints".

For now, consider that CEs deal with classes, not propositions, so it is necessary somehow map all propositions into classes. This is done by traditional model theory: The class corresponding to a proposition is that of all the possible worlds (models) in which the proposition is true. Then, the propositional connectives become operations on classes. For example, the class corresponding to P∧Q is the intersection of the class for P with the class for Q. That is, P∧Q is true in exactly those worlds where both P is true and Q is true.

Similarly for P∨Q, and ¬P. P⊃Q deserves special mention because it can be represented as "P is a subclass of Q", which is a single partition node. However, class relations corresponding to the other connectives are not quite so simple.

4.2 Union, intersection, etc.

The following list of class relations specifies for each one what behavior (in terms of propagating labels) it should have:

Intersection (P∩Q), figure 4-1a.

(i1) If +x on both "inputs", then propagate +x to the "output".

(i2) If +x on the output, propagate +x to both inputs.

(i3) If -x on either input, propagate -x to the output.

Union (P∪Q), figure 4-1b.

(u1) If -x on both inputs, propagate -x to the output.

(u2) If -x on the output, propagate -x to both inputs.

(u3) If +x on either input, propagate +x to the output.

Note that union and intersection are exact duals, as expected.

Complement (¬P), figure 4-1c.

(c1) If +x on either side, propagate -x to the other.

(c2) If -x on either side, propagate +x to the other

These (plus the subclass relation) correspond to the standard primitive connectives used in propositional logic. All of the 16 possible boolean connectives can be made from this (redundant) set of primitives. One additional class relation deserves mention, because it points out the distinction between viewing class relations as

"functions" and viewing them as "constraints". This relation is that of "exclusive or": An object is in P@Q iff it is in P, or in Q, but not in both. For any two arbitrary classes P and Q, P@Q contains exactly those objects which are in exactly one of them. This is also the behavior of a partition node, except that with a partition node the two "inputs" are additionally constrained to in fact have no objects in common. Thus the "function" viewpoint takes arbitrary "inputs" and produces some "output" (which is constrained to bear the appropriate relationship to the inputs); the "constraint" viewpoint abhors arbitrariness because that loses propagation information. Indeed, the addition of all the class relations of figure 4-1 is just a notational convenience: Each of them may be defined in terms of a small network of partition nodes (using 3 or fewer nodes); they add no new expressive power to the CE representation. So perhaps the centuries-long gap between syllogisms and modern propositional logic would have been considerably shorter if Aristotle had picked the right primitives!

4.3 Using the data-base (with an example)

The techniques for using a general propositional-logic CE data-base are an extension of those for processing syllogisms. A syllogistic data-base presumably contains a large number of syllogistic 'facts' (such as "no good-guys are crooks") in the form of CEs. The user then queries the data-base by providing syllogistic statements which the system is to prove or disprove (including the case of proving something exists by actually finding one). Note that the 'facts' are represented

by CEs, while the users 'queries' are represented by imposing initial labelings on the network and watching for interesting conditions (such as a contradiction) to occur during the constrained label propagations. It is important that the 'facts' and 'queries' have exactly the same expressive power, even though they are represented differently (facts as pieces of network, queries as initial labelings). This means that anything which the user can put into the data base he can later get out, and conversely.

However, the use of the boolean class relations allows one to express complex 'facts' as CEs which can not be represented as simple initial labelings. For example, the right-hand side of figure 4-2 expresses the facts that the class of fortunate-ones is exactly the union of the classes of wise-ones and lucky-ones, and that all unfortunate-ones are unhappy-ones. Now suppose the user wishes to know if everyone who is both unlucky and unwise is therefore unhappy. This can be represented in terms of a simple initial labeling by constructing the intersection class (named C here) of unlucky-ones and unwise-ones, and then asking if "all C are unhappy-ones". The left side of figure 4-2 shows this construction, and the initial labeling (+x on C, -x on UNHAPPY-ONES). This inference does indeed succeed. One way of producing the contradiction is as follow: The +x on C yields +x on both A1 and A2 by constraint i2; +x on A1 yields -x on WISE-ONES by c1; similarly, +x on A2 yields -x on LUCKY-ONES; then these two -x labels yield -x on FORTUNATE-ONES by u1; this yields +x on UNFORTUNATE-ONES by c2, finally yielding +x (and a contradiction) on UNHAPPY-ONES by t1.

Although cumbersome to write out in English, the derivation can be done quite simply by actually marking the appropriate +x and -x labels directly on the network diagram. This is of course how a computer implementation can do it (where "marking" means "add the label to the list associated with this class-point"). Note that this is <u>not</u> how traditional logic does such inferences: Theorem proving involves applying "rules of inference" to a large bag of axioms and derived statements, putting the derived results of the rules into the bag in turn (where they may then be used as input to the inference rules....) Thus theorem proving creates and copies items which are much more complex than the simple CE labels. Also, note that the inference of figure 4-2 includes the derivation of one of DeMorgan's laws (that $[\neg P \wedge \neg Q] \supset \neg[P \vee Q]$), for which most logical systems have to generate rather long proofs.

4.4 Implication revisited -- representing models explicitly

One item which has been glossed over so far is the issue of nested implications. Propositions such as $[P \supset [Q \supset R]] \supset [[P \supset Q] \supset [P \supset R]]$, although ugly, actually do appear in standard treatments of logic. (In fact, this thing is actually an <u>axiom</u> of a system presented by Mendelson [1964]). However, using the subclass relation for implication does not allow such nesting, since a "logic box" (like those in figure 4-1) for a nested implication needs to connect to three points, while the subclass relations just connects to two. This subsection discusses human intuitions about nested implications, presents an intuitively

satisfactory representation for them, and discusses ramifications of this representations in terms of "context", and knowledge about knowledge. This representation is also used in section 6, dealing with nested quantifications.

One way of allowing nested implications is to represent all nested ones by using the boolean truth-function for implication, while still representing the top-level implications using the subclass relation. This does not seem satisfactory for two reasons. First, any use whatever of the implication truth-function goes against my intuitive understanding of what "implication" is, anyway. I balk at the idea that P⊃Q "really means" the truth-function [¬P]∨Q: I vastly prefer "If I am getting wet then it is raining" over "Either I am not getting wet, or it is raining, or both". Now this might just be an artifact of English, but I doubt it. Second, note that 'right-nestings' of implication seem much "simpler" than 'left-nestings', although the truth-functional representation does not make any distinction. An example of a right-nesting is "If it is raining, then if I am outside then I am getting wet"; a sample left-nesting is "If it is true that if I am outside then I am getting wet, then I should not be outside".

Now, the only reason for mentioning these objections to truth-functional implication is that there is a good CE representation for nested implications to which the objections do not apply: It still uses the subclass relation (even for nested implications), and right-nestings are processed differently from left-nestings.

This representation allows an explicit correspondence between

the proposition that "A is a subclass of B" and the class of models for that proposition. Figure 4-3a shows the representation for this: It states that M1 is the class of all worlds in which A is a subclass of B. This kind of CE is is called a "world node". Now, let P, Q, and R be world-classes as above. Then figure 4-3b represents R⊃[P⊃Q], while 4-3c represents [P⊃Q]⊃R. Note that M2 and M3 are each a world-class for a subclass relation between the world-classes P and Q. Thus it is possible to nest these to any depth, although the label propagating computations get more involved as the depth increases.

Before examining these computations, consider what the objects in the world-class M1 represent. Each corresponds to one possible world wherein it is true that A is a subclass of B. Logically speaking, each world (or "model") has an "interpretation" which associates with each class in the network exactly those objects which it contains (in that world). However, for our purposes it is not necessary to be this precise, so we can consider a "world" to be a "possible universe" or "situation" in the physical (and metaphysical) sense. Note that the world-classes need not be finite or even denumerable: We are only interested in how world-classes relate to each other, not in their fine structure.

These world-classes have many uses which are beyond the scope of this appendix to discuss in detail, but which are mentioned here to give some concrete examples. First, different physical situations can be modeled as world-classes. One can consider an "event" to be something that changes the world, so each event has associated with it

a "before" world-class and an "after" one. The data-base can hold different facts about these different worlds, so it is possible to answer questions about the past and about hypothetical futures (perhaps based on proposed courses of action). Second, different states of knowledge can be represented by different world-classes. By associating a world-class with each sentient being, it is possible to explicitly represent what the being does and does not know about something (Such as "John knows Bill's phone number, but I don't think that Bill realizes it"). Of course, the different beings can be the same being at different times: "Yesterday I learned ...." As a final example, different worlds can represent different "contexts" of information within the data-base. If it is known that some desired information is in a particular context (such as "mathematics context" or "Macy's payroll register context"), then it is more efficient to only propagate labels within that context. Such a context world-class operates by "enabling" parts of the data-base (and perhaps explicitly disabling other parts).

Indeed, this "enabling" operation is exactly how labels propagating thru world-classes can affect things. To allow a label to be distinguished on the basis of which world is responsible for generating it, associate with each label a "world tag". When putting an initial labeling on the CE network, use the world tag "inf" (for "current inference"). Each world tag can also act as a label in its own right (in which case it, too, is associated with a world tag of its own). To clarify this, consider the example in figure 4-3d. We want to

infer that [P⊃Q]⊃[P⊃Q]. This is a nice example because it is trivial yet it shows the processing for both left-nesting and right-nesting. M4 is the world-class for the left-hand P⊃Q and M5 is the world-class for the right-hand one. Thus to infer [P⊃Q]⊃[P⊃Q] we must show that M4⊃M5.

To show that M4 is a subclass of M5, mark M4 with +x and M5 with -x, and attempt to derive a contradiction. The world tags for the +x and -x are both "inf": We write +x/inf and -x/inf when it is necessary to show the world tag. The +x sitting on M4 means that the associated inference (P⊃Q) is true in the world "x"; the -x sitting on M5 means that its associated inference (also P⊃Q) is false in the world "x". This is clearly a contradiction, but there remains the task of showing how the label propagating system can realize it. This is accomplished by further processing of the -x label. The -x says that the P⊃Q inference is false in the world "x". Now, <u>if</u> it can be shown that the inference is in fact true in "x", then we have a our contradiction in "inf" (because it is "inf" which is asserting that the P⊃Q must be false in "x") Showing that the inference is indeed true in "x" is easy: Label P with +y/x and Q with -y/x, and try to derive a contradiction in "x". Since there is a +x at M4, the subclass relation P⊃Q is true in world "x". Thus a -foo/x on Q can propagate to P, and a +foo/x on P can propagate to Q (as with constraint t1). Doing either of these (with the -y/x on Q or the +y/x on P) derives the contradiction.

Thus the general procedure for hacking labels at world nodes is as follows (Refer to figure 4-3a). If a +x reaches M1, then propagate any +foo/x which reaches A from A to B, and propagate -foo/x from B to

A. This works because the subclass relation is enabled in world "x". If a -x reaches M1, generate a sub-inference: For a new label "y", put +y/x on A and -y/x on B. If this sub-inference produces a contradiction, then the parent inference (which put the -x on M1) is also contradictory: The parent inference says that the sub-inference must be false in world "x", but it turns out that the sub-inference is actually true in "x". These two different processes (for +x and -x on M1) may help explain why right-nested implications seem simpler (to me at least) than left-nested ones.

4.5 Logical consistency and completeness

The issues of consistency and completeness discussed here are presented in terms of propositional logic, but the conclusions drawn can be seen to apply to all the CEs mentioned in this appendix.

A system of inference is logically consistent iff everything which can be inferred is true. This is usually demonstrated by showing that the various inference rules used do not individually cause inconsistencies, so (by induction) any combination of them also preserves consistency. "Truth" is defined in terms of a model theory, and then it is shown that each inference rule preserves such "truth". Well, we do not have to go to that trouble, because the CE "inference rules" (concerning label propagations) are themselves a direct consequence of the "model theory" (concerning class relations). This correspondence is shown for each constraint propagation rule (such as t1-t4) when that rule is first presented, so there is no need to repeat

everything here. Thus CE inference is consistent since the "inference rules" themselves are purely model-theoretic to start with.

Logical completeness, however, is another matter entirely. Consider the rather degenerate example in figure 4-4. Clearly A and B are the same class (since both are the union of A1 and A2), and similarly B and C are the same. Indeed, it is trivial to show that A is a subclass of B (and conversely) by labeling A with +x and B with -x (or vice-versa) and then deriving a contradiction using constraints u2 and u1. Similarly, B and C are trivially subclasses of each other by labeling +x, -x and using constraints i2 and i1. Furthermore, it is trivial to show that C is a subclass of A (labeling +x, -x and applying constraints u2, u1, i2, and i1 to produce a contradiction at B). However, with the propagation scheme presented so far it is not possible to show that A is a subclass of C. This happens because the +x on A and the -x on C both get "blocked" -- no constraint is immediately applicable.

Now, it is unclear whether or not structures with this "dual blocking" property are ever encountered in practice with any "reasonable" data-base. If it turns out that that such structures do present a practical problem, there is a conceptually simple way to achieve full logical completeness. However, even though this method (about to be presented) is conceptually simple, it is computationally disastrous even for highly-parallel hardware. The problem is that full logical completeness always requires an exponentially-explosive combinatorial enumeration of cases. (Karp [1972] gives strong

mathematical evidence that a complete proof procedure for propositional logic must always take at least exponential time). Anyhow, the super-idiot version of the method uses the "model theoretic" approach: For a CE network containing N+2 class-points (with 2 of them being initially labeled), generate all 2^N combinations of assigning the label +x or -x to each of the remaining classes. Then for each of these 2^N combinations of N label assignments, see if it violates any node constraints. If each of the combinations violate at least one constraint, then the initial labeling is indeed contradictory. This method is analogous to that of doing propositional logic problems by enumerating the entire truth-table and checking each entry.

The ordinary-idiot version of the method is still exponentially explosive, but with a smaller explosion rate. This method involves doing all possible label propagations until everything gets stuck. Then pick any unlabeled point and assume x (or whatever) is in the class. So label the point +x and finish the inference. If that wins, then assume the point does <u>not</u> contain x (by labeling it with -x). Finish the inference again, using this different assumption. Since one of these cases must hold (x must be either in the class, or not be in it), and since the inference wins for both cases, the inference must win in general. The exponential explosion comes in because both of those "finish the inference" processes may in turn themselves require the testing of both cases of some other point, and so on recursively.

Consider applying this method to point B of figure 4-4. If B is labeled with +x, we get the contradiction using i2 and i1; if B is

labeled -x, we get the contradiction using u2 and u1; so we win. The general point is that full logical completeness can be easily achieved, if one is willing to put up with an exponential explosion. For a very large data-base, then, it is impossible to achieve full logical completeness anyway, so there is little point in worrying too much about it.

5 Functions and relations

5.1 Binary relations: Image

The above schemes of CE inference deal only with questions of class relations. It is only as powerful as propositional logic, meaning that it can not handle relations in general ("Aristotle is taller than Socrates") or any quantification beyond the simple "All X's are Y's", "No Y's are Z's", etc.

The problem of handling general relations is of course very basic -- even simple attribute-value systems (such as SIR [Raphael 1964]) have the ability to represent such relations, an ability which is of course needed for a practical data-base system.

What is wanted is a way of expressing such relations in a manner which allows inferences to be made by the same sort of local label-propagation used in the preceding sections. The basic notion to be used for this is that of the image of a class under a relation. For example, "my parents' hobbies" is the class which is the image of the class "my parents" under the relation "hobby of". We use the following notation for such a "relation node":

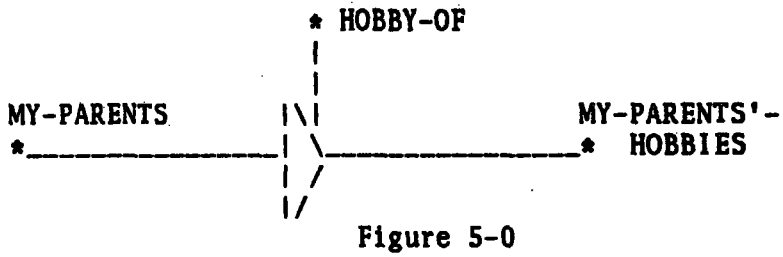

Figure 5-0

Now, we must be very precise as to what exactly the "image" is. Let R be the image of D under F. (In 5-0, the R is "my parent's hobbies", the D is "my parents", and the F is "hobby of"). Consider F

as being a class of ordered pairs ‹d,r›, pairing an object in the domain and an object in the range. There is no uniqueness constraint -- many d's may map onto the same r (playing golf is a hobby of many people), and many r's may be in the image of a single d (my father has many hobbies). Thus F is a general relation and not just a "function" in the sense of there being a unique object in the image for any given object in the domain.

To insure that R is at least as big as the image, we insist that if ‹d,r› is an element of F, and d is an object in D, then r must be an object in R. That is, R contains every object which participates in the relation F with any object in the domain. To insure that R is at most as big as the image, we insist that for every r in R, there is a ‹d,r› in F such that d is in D. That is, R does not contain any extra objects which do not participate in the relation F with any object in D.

Now this all may seem overly nit-picky, but in fact such a precise definition of "image" is exactly what is needed in order to formalize the propagation constraints for relations. (Parenthetically, it was not particularly easy to get things to this level of crispness -- I had the general idea of using the image under a relation for a long time, but a useful formalization had to wait until my semantic understanding caught up with my hacking)

5.2 General domain-range specification

One nice feature of using the image in the above form is that it allows compact specification of the domain and range of a particular relation. Domain-range is very important in a lot of applications. For example, a business data-base might expresses that the domain and range of the relation SUPPLIES-PART are exactly the known SUPPLIERS and PARTS, respectively. Figure 5-1 states this fact and in addition that that 3M (a supplier) supplies all the parts. That is, any part which is supplied by any supplier is also supplied by 3M.

Another use for domain-range specifications is that a programming system might use a different function for square-root depending on whether the argument is an integer, real, or complex number. Thus these 3 sub-functions for the different domains would have different domain-range specifications. In general, conditional expressions can be viewed as being conditional on the domain of the terms in the expression: For factorial, the range is (1) when the domain is (0, 1); the range is the recursive part of the definition when the domain is the integers > 1; the range is null when the domain is anything else. Thus an intelligent system trying to evaluate a factorial using such information would see which domain constraint(s) the argument satisfies, and then do the thing specific to that domain.

5.3 Constraint propagation

Given the above definition of the image, we want to formalize some labeling constraints. To do this, we need a new kind of label for

relations: Let the label +⟨x,y⟩ on a relation mean that the relation relates x and y; let -⟨x,y⟩ mean that the relation definitely does not hold between x and y. Note that these can propagate like ordinary +z and -z labels (using t1-t4): Doing this, a relation can be treated like any other class: For example, the relation "parent of" can be defined as being partitioned into "mother of" and "father of". Thus we have not introduced a new kind of label so much as a new kind of object, the ordered pair. Now, using our definition of "image", we have these propagation rules:

(f1) From +x on D, +⟨x,y⟩ on F, infer +y on R. That is, since R is the entire image of D under F, y must be in R if x is in D and F relates x,y.

(f2) From +y on R, generate a new object g0037 and infer +⟨g0037,y⟩ on F and +g0037 on D. That is, since R is no larger than the image of D under F, for every y in R there must be some domain object which bears the relation F to it. We do not know which object this is, so we use a new name. This is a little obscure, but will hopefully be clarified by the examples below.

(f3) From -y on R and +x on D, infer -⟨x,y⟩ on F. This is a negation of (f1): If -y on R, then either -x on D or -⟨x,y⟩ on F; thus if also +x on D (and thus not -x on D), it must be -⟨x,y⟩ on F. More intuitively, if x is in the domain and y is not in the image, then ⟨x,y⟩ can not occur in the relation.

(f4) From -y on R and +⟨x,y⟩ on F, infer -x on D. This is the other case of the negation of (f1): If -r on R, then either -x on D or

$-\langle x,y\rangle$ on F. Since not $-\langle x,y\rangle$ on F, it must be $-x$ on D. Again, if y is not in the image but $\langle x,y\rangle$ is in the relation, then x can not be in the domain (otherwise y would have to be in the image).

(f5) If D is an object (singleton class), then for $+x$ on D and $-\langle x,y\rangle$ on F, infer $-y$ on R. Since D is a singleton, and since x is in that singleton, then only x can be in the singleton. Thus if x is the entire domain, and x does not relate to y, then y can not be anywhere in the range.

(f6) Similarly, if F is an object (contains only a single ordered pair), then for $+\langle x,y\rangle$ on F and $-x$ on D, infer $-y$ on R.

5.4 Transitive relations

As an example of making inferences using some of the above constraints, this subsection concerns a representation for transitive relations. The fact that "taller", for example, is transitive, tends to be a bit of a nuisance. Either something must be built into the system code to recognize transitive relations and treat them specially (as in SIR and OWL), or some complex thing such as

$$\forall x \forall y \forall z\,[\mathrm{TALLER}(x,y) \wedge \mathrm{TALLER}(y,z) \supset \mathrm{TALLER}(x,z)]$$

must be given to a theorem prover (which may choke on it).

Since transitive relations are so common (virtually all English adjectives, for example, have meaningful comparative forms), then I assume that for humans they are both usefully expressive and easy to compute with. It would be nice to have a clean formal representation with the same feature. Consider: For most relations like "taller"

there is associated a "quantity", in this case, "height". In common-sense terms, A is taller than B if A's height is greater. Attempts (such as by Schank [1972]) to use this fact by assigning numerical quantities seem very forced -- what, after all, are the measure units for happiness, goodness, or beauty? Also, if the issue of units is ignored by assigning a dimensionless number, then statements such as "Joe's height is greater than Sam's happiness" would be sensible, which they obviously are not (except in a metaphorical sense: If Joe is a midget, it is just a cute way of saying that Sam is unhappy).

So, our scheme will be to have a relation HEIGHT (for example) which maps from the domain PHYSICAL-OBJECTS to the range HEIGHTS. The HEIGHTS will be ordered by the superclass relation -- if H1, H2 are heights, then H1 is greater than or equal to H2 iff the class H1 is a superclass of H2. Note that these classes H1, H2 are <u>not</u> classes of objects having some particular height -- they are classes which represent the ordering of abstract heights. (What exactly the <u>objects</u> are in those classes is unimportant. One way of looking at it is that each abstract height is a class of tokens with numbers on them, and the "measured height" in inches is the maximum number to be found among all the tokens in the class. The use of maximum insures that all superclasses of a given height must have a "measured height" at least as big).

Since the CE inference scheme already knows about the transitivity of "superclass", it will know about the transitivity of "greater height" since that is represented as superclass. Figure 5-2a

says that A is taller than B -- A's HEIGHT is a superclass of B's HEIGHT. Similarly, figure 5-2b says that B is taller than C. Finally, figure 5-2c represents A's height and C's height. (The "stacking" of relation nodes in 5-2 means that all the stacked nodes have as their F points the same class, in this case HEIGHT. This cuts down on the number of crossing lines in the network diagrams). Now, the goal is to show that A's height (point AH2) is a superclass of C's height (point CH2). As with inferring "all redwoods are plants" (from section 2), we label CH2 as +x and AH2 as -x. If a contradiction is obtained, then there is no such x, so all of CH2 is contained in AH2.

Before forging ahead with the label propagations, consider again exactly how the data-base is used. We want to test the assertion that A is taller than C by using a data-base which presumably contains some relevant information. Here, figures 5-2a and 5-2b represent existing knowledge in the data-base. Figure 5-2c is new stuff which has been built solely for the purpose of testing the assertion. In a sense, a user who wishes to use the data-base has access to certain "terminals" (points) in the data-base to which new stuff can be attached. That is, it would be meaningless to ask "is A taller than C" if the user had no access to the names A and C, about which the data-base presumably contains some facts. Note that the user of the data-base does <u>not</u> have to know anything about B, or even that B exists at all.

Further, the user does not have to know that the heights of A and C are already explicitly represented in the data-base (by AH1 and

CH1, respectively). In the process of label propagation, we will implicitly infer that the points AH2 and CH2 can be identified with AH1 and CH1 for the purposes of the current inference. Such identifications can easily be made explicit and given to the user, so that in the future the user has a more direct way of refering to A's height (using the existing AH1) than by constructing another thing such as AH2. Using the existing AH1 and CH1 is more efficient both in terms of space (since the new stuff need not be created) and in terms of time (since the inference that AH2 is identical to AH1 need not be rederived each time it is needed).

Now, the inference for figure 5-2 can best be thought of as happening in three stages: The first pushes +x from CH2 to CH1; the second pushes it from BHC to BHA; and the third pushes it from AH1 to AH2, causing the desired contradiction since AH2 is already labeled -x. The three stages involve using identical constraints, so it is only necessary to explain one of them in detail. Consider the second stage: From +x on BHC we get +<g0341,x> on HEIGHT and +g0341 on B by constraint f2, where g0341 is some new label; then from these two labels we immediately get +x on BHA by constraint f1! Intuitively, this happens because two different occurrences of the image of a given class under a given relation (such as BHC and BHA both being the image of B under HEIGHT) must be identical, thus the +x should always be able get from one to the other. So three applications of this do all the work, with some interstage glue being provided by constraint t1 (which gets stage one's output +x from CH1 to BHC, and gets stage two's output

from BHA to AH1). Note that this particular inference for figure 5-2 is by no means unique -- just as for "all redwoods are plants", using different constraints would produce a different inference, with the final collision of + and - occuring at some other point in the figure.

5.5 A comparison with Codd's system and Planner

Codd [see Date & Codd 1974] proposes a "relational model" for data-bases. Such a data-base consists of a collection of N-ary relations, each of which is expressed as an enumerated set of N-tuples. In the case of "normalized" relations (which is what we are concerned with here), the slots of each N-tuple are filled with atomic data items (such as employee names or part numbers). For all tuples in a given relation, the i-th slot of each is filled with the same "type" of atomic data. Thus a relational data-base can be written out as a collection of tables: Each relation is a different table, the rows of which each represent one N-tuple, and the columns of which are the different slots. For example, one relation (table) might be "supplies part", where the first slot (column) names a supplier, and the second one names a part which the supplier supplies.

This sort of data-base scheme is conceptually very much like the non-procedural portions of the Planner data-base [Sussman 1970], with two major exceptions. The first is that Planner represents N-tuples in a slightly different manner. Each tuple explicitly names the relation of which it is a member (such as (SUPPLIES-PART 3M WIDGET)), and data-base retrieval is done on the basis of slot position within

the tuple instead of on the basis of the slot (column) name. That is, Codds system can be asked for "all suppliers who supply widgets" (in some formal notation), while Planner is asked to find all tuples which match (SUPPLIES-PART $? WIDGET). Besides this minor notational difference, there is a second, more important difference. This is that Planner has slightly more expressive power. In Planner, the relation's name (in the tuple) has the same status as any other slot, so one can ask for all matches against ($? 3M WIDGET), which (in a suitably structured data-base) might mean "all relations which occur between 3M and widgets" (for example, 3M may use or recycle or engrave widgets, in addition to supplying them). Codd's system can not do this because a "relation" (table) can not be used as a search key -- it 'contains' manipulable atomic data without itself being manipulable as a unit.

This difference between Codd's system and Planner is also the major difference between both of them and the CE representation. Briefly, in the CE representation <u>everything</u> is manipulable: Classes, objects, ordered pairs (2-tuples), relations (classes of ordered pairs), and even the structure of the CE network itself are all explicitly manipulable (The network structure can be manipulated by selectively enabling different constraints via worlds). This is why the CE representation has such a clean semantics and can represent both statements and questions in the same manner.

Furthermore, the fact that Codd-like systems represent everything in terms of <u>explicitly enumerated</u> finite sets drastically restricts the expressive power. Consider figure 5-1, which states that

3M supplies all parts. To say such a thing in Codd's system would require that the "supplies parts" table have a separate 2-tuple for each part which 3M supplies. Even worse, if 12 different companies each supply all of the 100 different parts, this involves 1200 table entries. The problem is that there is no way to talk about a class of things (such as "all parts") as a unit. In addition, the use of explicit enumeration requires that the system have complete information about the contents of every set: The only way to state that "all parts which company A supplies are also supplied by company B" is to know exactly what those parts are and explicitly list each one as being supplied by both A and B. This clearly does not capture the expressed generalization about all parts (as opposed to each of the currently-known individual parts). Since the desired generalization is not directly expressed, the system must not only generate all those table entries to express an approximation to the general fact, but it must examine them all if asked "does B supply all the parts which A does". Using a CE network, both the generalization and the question are trivial to express, using a single subclass constraint.

5.6 N-ary relations

In Codd's system, it is trivial to express N-ary relationships, since N-tuples are a primitive of the system. Even though N-tuples are not primitives in the CE representation presented in this appendix, it is easy to represent them in an intuitively-satisfactory manner. Consider an example taken from Date and Codd [1974]. They represent

"part" as being a relation among the slots P# (part number), PNAME (part name), COLOR, and WEIGHT. The P# is included for the purpose of acting as a "primary key", which is something that uniquely identifies the tuple. Now note that there is nothing that one can point to here and say "this is a part". The thing that comes closest is the P# (since there is a 1-1 correspondence between parts and part numbers), but the "part number" is not the part itself. It seems to me that individual parts can exist independently of their known attributes (such as color, weight, and number). Of course, the tuple itself can be considered to be the part, (with the slots giving each part's attributes), except that there is no way in Codd's system to refer directly to a given tuple -- that is why the "primary key" is needed as an indexer.

In the CE representation, each part is represented explicitly, and the various attributes are each represented as a binary relation. Figure 5-3a enumerates some of the kinds of parts, and gives the attribute values for widgets. Considering just the partition nodes for the moment, note that things can be specified at exactly the desired level of detail. It is possible to refer to all PARTS, to all of some particular kind of part (such as all WIDGETS), and even to particular widgets (W1, W2, W3, and W4). Note that Codd's representation is restricted to one level of detail, since there is no representation for higher-level groupings (classes) of objects. In this example, Codd's system is restricted to the level of WIDGETS -- it is impossible to refer to either a more general concept (such as PARTS), or a more

particular one (such as W2).

Concerning the relation nodes which supply the attribute values, note that they too can be specified at exactly the right level of detail. Since all widgets have the same part number, part name, color, and weight, the specification of these attributes need only be done once (for the entire class). For example, the relation node for COLOR-OF says that the image of the entire class of widgets under the relation color-of is the single object "pink". That is, all widgets have the same color. If it is desired to create a transitive partial ordering for weights (as subsection 5.4 does for heights), just add a relation which maps from a particular weight (such as 3-tons) to the class which represents the corresponding "abstract weight". (5.4 uses a 1-level mapping from individuals to abstract heights instead of a 2-level one from individuals to individual heights to abstract heights in order to simplify the example. The identical processing goes on in the 2-level case).

Now, it is indeed possible to do the same kind of inference on this structure as we have been doing all along. Suppose a user wishes to know the color of part type W2. The user does not have to know that W2 is a kind of widget, or anything else. All the user needs is access to the network point for W2. Given this, the construction in figure 5-3b constrains the class FOO to be the color of W2. Now, label FOO with +x -- if this +x label reaches a class, then it is known that FOO is a subclass of that class. So in particular, if +x reaches an object, then FOO is a subclass of that object. The only way in which

one object can be a subclass of another is if they are the same, so in fact the object which the +x reaches must be the color of W2.

One way in which +x can get from FOO to 'pink' is by the same series of constraints used in each of the "stages" in section 5.4: +x on FOO yields both +⟨g0342,x⟩ on COLOR-OF and +g0342 on W2 (by constraint f2); the +g0342 propagates from W2 to WIDGETS by constraint t1; and finally the +g0342 on WIDGETS and the +⟨g0342,x⟩ on COLOR-OF combine to yield +x on 'pink' (by constraint f1).

5.7 Inverse relations -- "active" and "passive"

One unusual feature of the CE representation for binary relations is that they are treated asymmetrically. For example, every object in the range must have at least one related object in the domain (by constraint f2), but not vice-versa. Thus there is a problem if we need to use both the "active" and "passive" forms of a relation (such as "the companies which supply the part" and "the parts supplied by the company").

These two forms of a relation can be related by introducing a new constraining node, the "inverse node". Figure 5-4 contains an inverse node (drawn as a bisected diamond) which constrains the relations "supplies part" and "part supplied by" to be inverses of each other. So point A represents the class of all parts which 3M supplies, while point B represents all the suppliers which supply a widget. This inverse node constrains every ⟨x,y⟩ instance of the relation SUPPLIES-PART to have a corresponding ⟨y,x⟩ instance of

PART-SUPPLIED-BY, and vice-versa. Thus the general label propagating constraints for an inverse node and arbitrary objects x and y are:

(v1) If +⟨x,y⟩ on either side, put +⟨y,x⟩ on the other.

(v2) If -⟨x,y⟩ on either side, put -⟨y,x⟩ on the other.

6 Logical quantification

6.1 Implicit quantification

With any sort of deductive inference system, processing becomes more complex (or even impossible) when the data-base facts and queries are allowed to contain complex logical quantifications. This section examines how the CE representation can be extended to handle such things (using a fair amount of brute-force enumeration, however).

But before doing this, consider what kinds of quantification can already be handled using CEs. As seen in section 2, the constraints embodied in the harmless-looking "partition node" need explicit quantification in order to be represented in first-order logic. Indeed, the intuitively simple notion of "subclass" is usually defined by something like: "A is a subclass of B iff for all x, if x is in A then x is in B". This explicit universal quantification is made implicitly in the CE representation by allowing <u>any</u> label of the form +x to go from A to B (by the first part of t1), and similarly for the other propagation constraints. Thus there is a certain power of quantification inherent in the dynamic processing (via label pushing), which therefore does not need to be expressed explicitly in the static representation.

For example, consider figure 6-1. The top two relation nodes state that supplier-1 supplies both widgets and frobs (Each is a subclass of the class of all the different kinds of parts which supplier-1 supplies). The bottom relation node represents the stuff added for the query "what parts does supplier-1 supply": The class FOO

is exactly what we want (the entire image of supplier-1 under the relation supplies-part). So, as usual, to find relevant subclasses of the given class FOO, we mark FOO with -x, and propagate. Any point reached by -x is then a subclass of FOO. Since all objects (singleton classes) obviously contain themselves, we can implicitly "hard-wire" a permanent +supplier-1 label to the class 'supplier-1'. Then by constraint f3 on the bottom node we can put -<supplier-1,x> on SUPPLIES-PART. Then constraint f5 on the two top nodes produces -x on A1 and A2, respectively. Finally, constraint t2 propagates the -x from A1 and A2 to WIDGETS and FROBS, and we are done. If in addition we want to go down to the ultimate atomic objects (such as the 4 types of widgets in figure 5-3), then we can propagate the -x labels further.

6.2 Explicit quantification

Of course, there exist queries which are too complex to be handled by the built-in implicit quantification mechanisms. Since the most general way of handling a "find all" or "find one" query is to actually construct a class which contains what is desired (and then start a -x from it), it is first necessary to be able to construct complex quantificational facts before being able to answer questions about them.

Consider "FOO is the class of all suppliers each of which supplies all parts". Now it is easy to state for a particular supplier that the supplier supplies all parts (for example, figure 5-1), but is not so easy to capture the general case. One possible approach is

diagramed in figure 6-2: The subclass of suppliers (FOO) does indeed supply all the parts. However, nothing prevents FOO from just consisting of some supplier-A and supplier-B, which together supply all the parts, but which individually do not. What 6-2 represents is that "FOO is a class of suppliers, which among themselves supply all parts".

What is needed is some way to refer to a "typical" supplier (such as 3M in figure 5-1) which individually supplies all parts, and then say "let FOO be the class of all of those". Indeed, English makes it clear that such a typical object exists (at least at the surface level): In "list all suppliers such that the supplier supplies all parts", the definite description "the supplier" clearly refers to a typical object. OWL, in fact, is strongly biased towards the use of such descriptions by typical example, as is Winston's [1970] scene-analysis program.

Figure 6-3 shows the CE representation for such a typical object. This is a copy of part of figure 5-1 (stating that 3M supplies all parts), with the addition that the object "typ-supplier" is the desired typical object: It, like 3M, supplies every kind of part. The two other lines (with arrows going out of and into the node) are respectively the desired 'destination class' of all of those objects which behave as the typical one does, and the 'source class' from which such objects may be drawn (in this case, from the class of suppliers). This kind of node is called a "typ node".

Now as usual, the important point is not so much that such typical objects can be statically represented, but that in fact some

useful computations can be done using them. To find suppliers each of which supply all parts, we put the customary -x on FOO, and see where is propagates to. The way a typ node processes a -x on the destination-class point is as follows: Enumerate objects one at a time from the source class; for each one, test to see if it meets the description of the typical-object; when one does, then it is a desired object in the destination-class. To enumerate the objects in the source class, we use the customary method of putting -y (a new label) on the source class and noting those objects which a -y reaches. In this case, it will reach all suppliers.

To test each such object to see if it meets the typical-object description, assume the contrary and try to derive a contradiction. In this case, assuming the contrary means assuming that there is a part which the enumerated object does not supply. This is done by generating a new label z (to represent the hypothetical part), putting +z on PARTS, putting -‹typ-supplier,z› on SUPPLIES-PART, and temporarily "binding" typ-supplier to the enumerated object. This "binding" means that anything which refers to typ-supplier (as a point or a label) will really refer to the enumerated object.

As an example of this rather involved process, consider the case when the enumerated object happens to be 3M. PARTS gets the label +z, and SUPPLIES-PART gets -‹typ-supplier,z› (which by virtue of the "binding" is really -‹3M,z›). Finally, 3M (being an object class) has its hard-wired +3M label. Well, thank heavens we are done: Constraint f5 applied to 3M's relation node forces a -z on PARTS, and we have the

contradiction.

Clearly, this process is rather expensive: It involves a brute-force enumeration and the maintenance of temporary bindings. This is unfortunate, but there is some consolation in the fact that other data-base formalisms have to go through pretty much the same thing. The moral is that one must be prepared to pay a stiff computational price if one insists on asking quantified questions.

6.3 Nested quantification

Of course, things can always get worse -- in addition to the computational problems with doing explicit quantification at all, there is the problem of in what order things should be done when multiple quantifiers are used. A sufficiently high-level query language (such as Zloof's [1974] "query by example") can remove the burden of ordering from the naive user, but within the data-base itself the order must be specified. (Zloof's system and the other Codd-like systems do not have to worry about quantified facts in the data-base, because they use a data-base containing only completely specified finite tables of atomic facts).

Within the CE framework presented in this appendix, the orderings among quantified typical-objects can be made explicit by using worlds. The "world node" used in section 4 provides an explicit representation for all worlds in which a given subclass relation holds, and the same principle can be applied to the "typ nodes" of this section. Figure 6-4 shows a typ node in all its glory. The in-world

line specifies when the constraints represented by the typ node apply -- if +w is on the in-world then the constraints apply for labels generated by w; if -w is on the in-world than the constraints definitely do not apply. The out-world is a filtered subclass of the in-world, much as the destination-class is a filtered subclass of the source-class: +w can go from in-world to out-world iff the source class is not empty. That is, the out-world is activated iff there is indeed a source object to be bound to the typ-object.

Thus by chaining the out-world of one typ node to the in-world of another, the order in which they operate can be specified: The first node will pick a source object, bind it to the node's typ-class, and then fire up its out-world, letting the second node now bind one of its own source objects. This order corresponds to the "scoping" or lexical nesting of quantifiers in mathematical logic, and indeed the same amount of non-intuitive complexity can be produced. One simple 2-level example is shown in figure 6-5: It states that every girl who has a brother who has a pet likes it (the pet, that is). A typ node for which no in-world is specified is always enabled, just as the one in figure 6-3 is. The final out-world connection to the world node states that in all worlds in which such objects can be found (the girl, her brother, and his pet), the pet is contained in the class of things which the girl likes.

If this were a query to be processed (ie. "Is is true that ...."), it would require 2 nested loops of enumerations: One for the girls, and one for each girl's brothers. This is somewhat painful, but

appears to be the only way to answer such questions. So the conclusion from all this is clear: For facts and queries which can not be handled by the CE representation's implicit quantification, it must go through the same kind of brute-force enumeration which the other data-base schemes use.

7 Summary

This appendix has presented a data-base representation based on Constraint Expressions. It has the following features:

(1) It is highly modular, in several senses. First, the primitives of the representation themselves are semantically modular, in that any given fragment of CE network has the identical local meaning regardless of context (a feature which first order logic, for example, does not share, since a subexpression of a logical formula may have different meanings depending upon the quantifiers within which it is scoped). Second, the processing of inferences is computationally modular, in that each node can be viewed as an active process which waits for certain label patterns to appear on its attached points, and propagates new labels when such a pattern occurs. Thus the representation can reap the full benefits of using parallel-processing hardware (even to the extent of actually building the data-base expressions themselves out of cellular automata). Finally, the growth of the power of the representation is evolutionarily modular. The sequence of sections in this appendix corresponds to the sequence in which the various different kinds of constraint nodes were developed. It is important that every new increment of expressive power represents an extension of what previously existed -- no changes in the previous primitives have been necessary.

(2) It has great expressive power, in both the logical and computational sense. Inferences not involving complex quantification are easy to compute, without resorting to an exhaustive enumeration of

cases. Of course, such an exhaustive search is needed if abstract logical completeness is desired, but the "incomplete" system is still extremely powerful. Also, it has great user-oriented expressive power in that anything which can be stated as a fact to the data-base can also be used as a query, and vice-versa. Systems based on a fully-instantiated finite model theory (such as the non-procedural portions of Planner, and the various Codd-like data-base schemes) do not have this feature, and in fact are unable to express any sort of "general" fact whatever.

Finally, it has a clean semantics. The combination of this and (2) is very unusual: First-order logic has the semantics, but is computationally disastrous even for small data-bases; The procedural aspects of Planner and Conniver [Mcdermott, 1973] have the computational power, but do not (at the current state of the art) have any unifying semantic base. Having a clean semantics aids both the task of implementing the primitives of the system on a computer, and that of encoding a large data-base into the implemented representation.

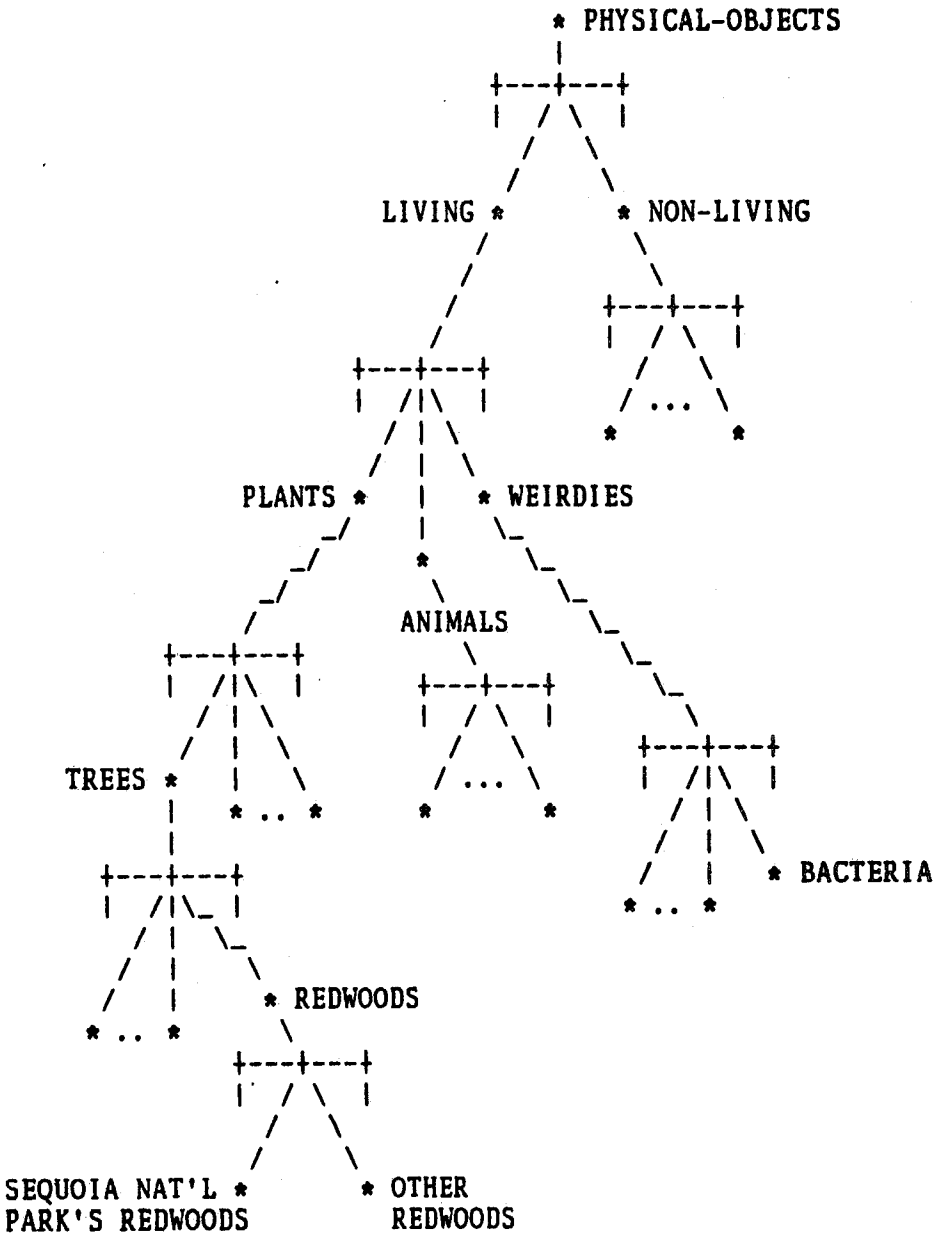

Figure 2-1 -- Sample Taxonomy

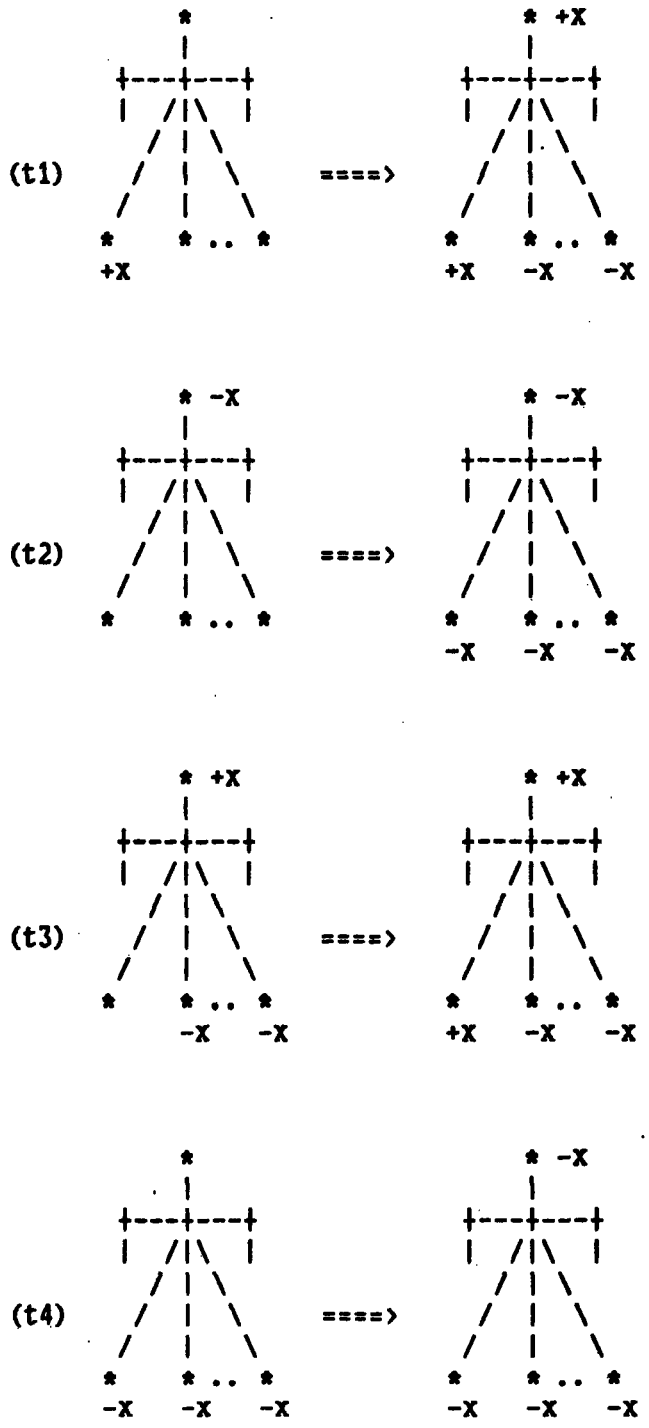

Figure 2-2 -- The 4 label-propagation constraints

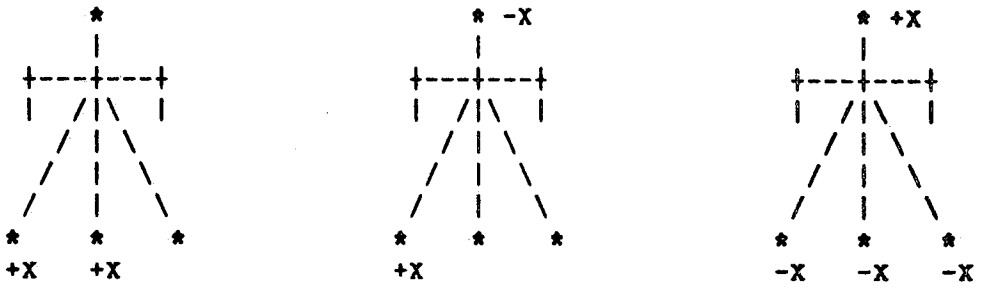

Figure 2-3 -- The 3 illegal labelings

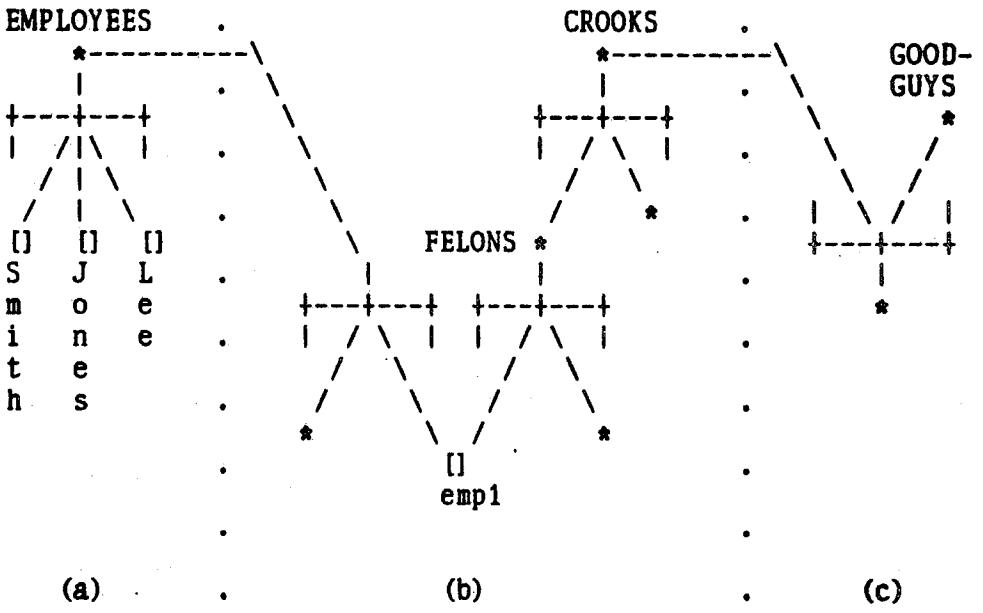

Figure 3-1 -- The use of "objects"

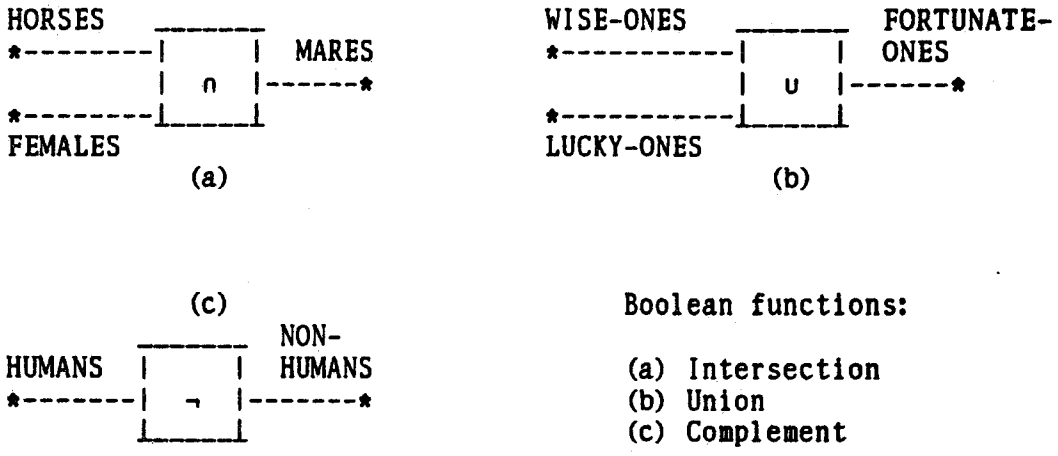

Figure 4-1 -- Boolean functions

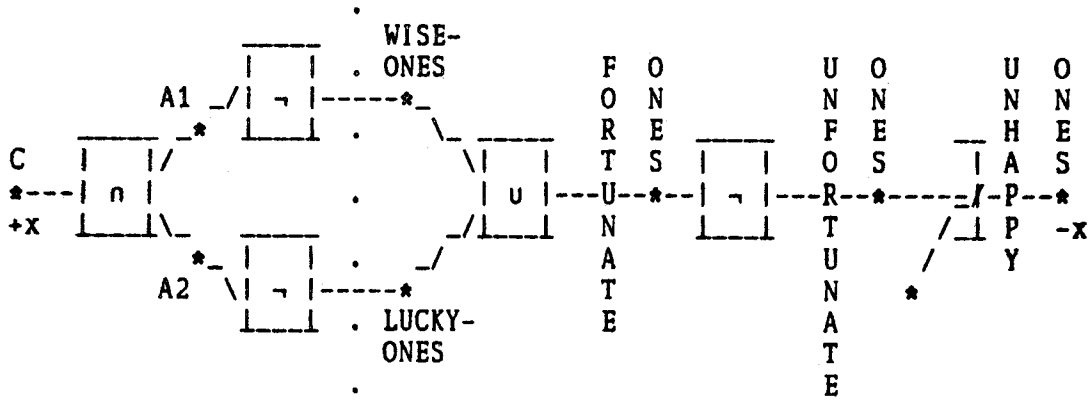

Figure 4-2 -- Example using boolean functions

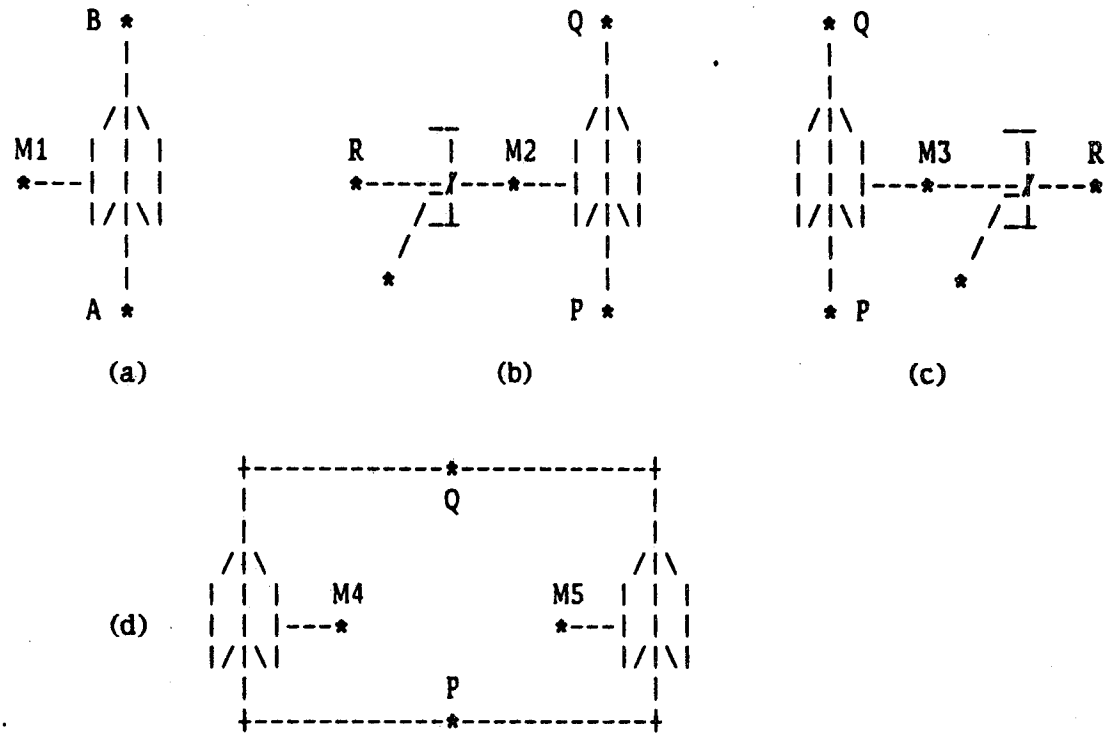

Figure 4-3 -- The "world node"

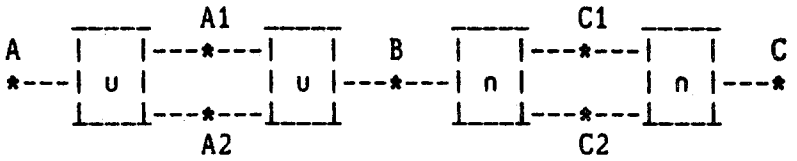

Figure 4-4 -- An example of incompleteness

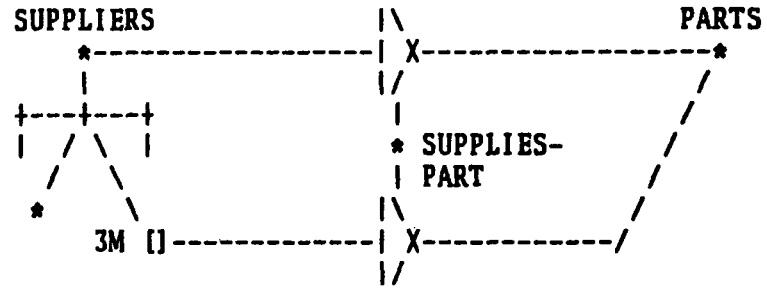

Figure 5-1 -- Domain-Range

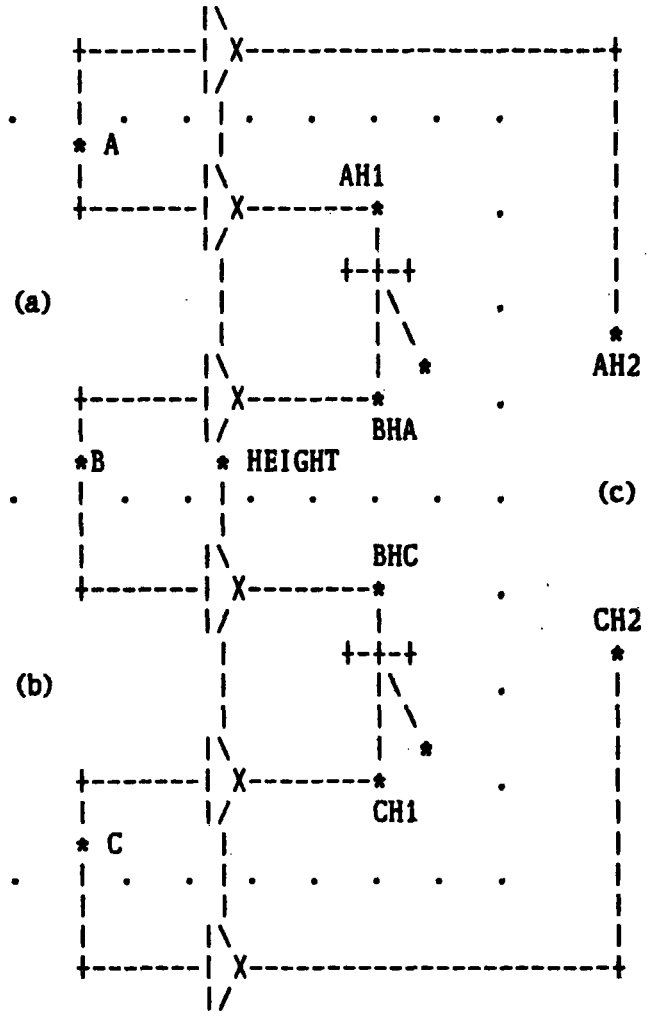

Figure 5-2 -- Transitivity

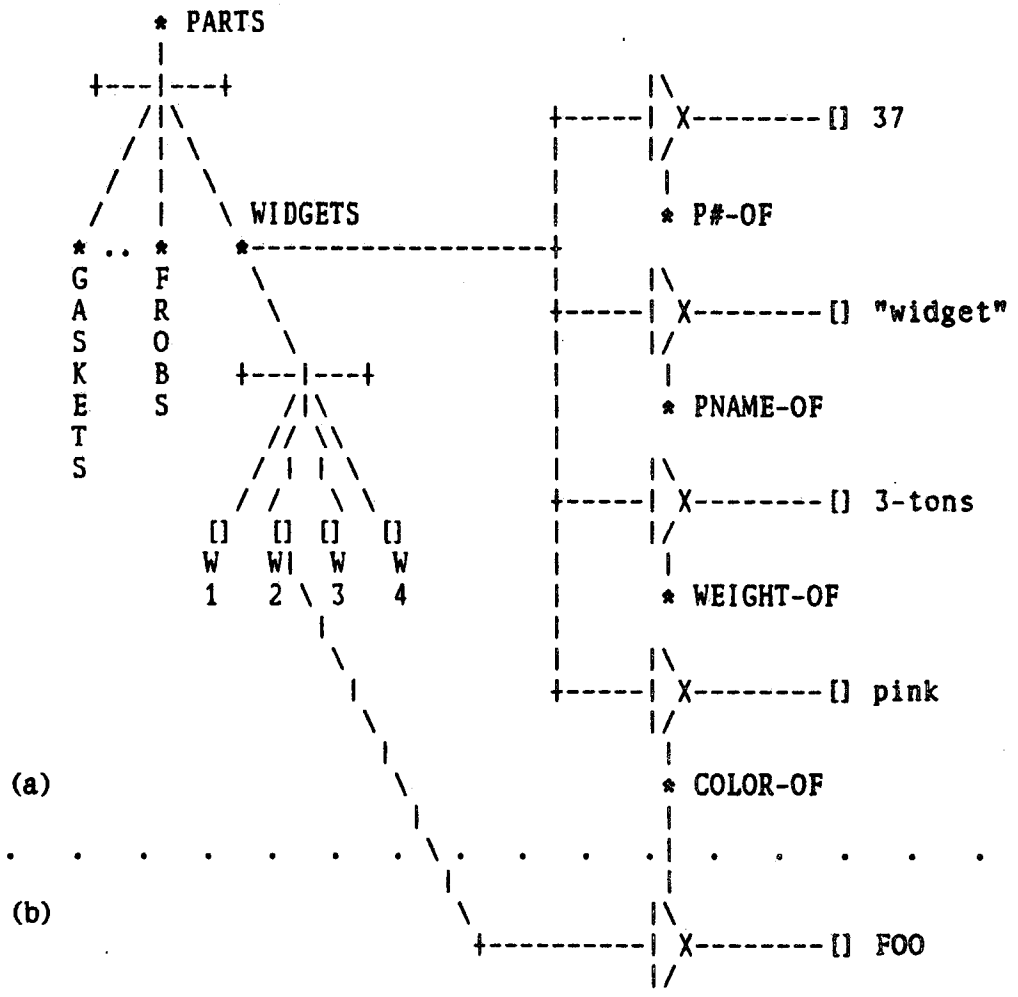

Figure 5-3 -- N-ary relation

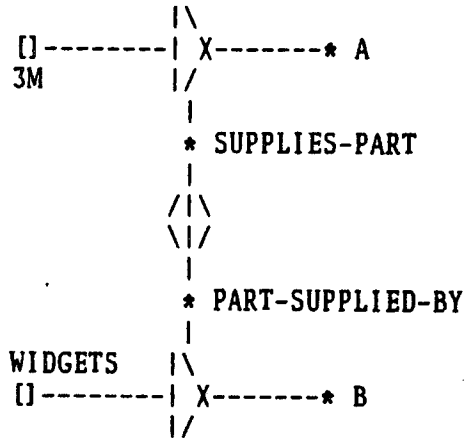

Figure 5-4 -- Inverse relation

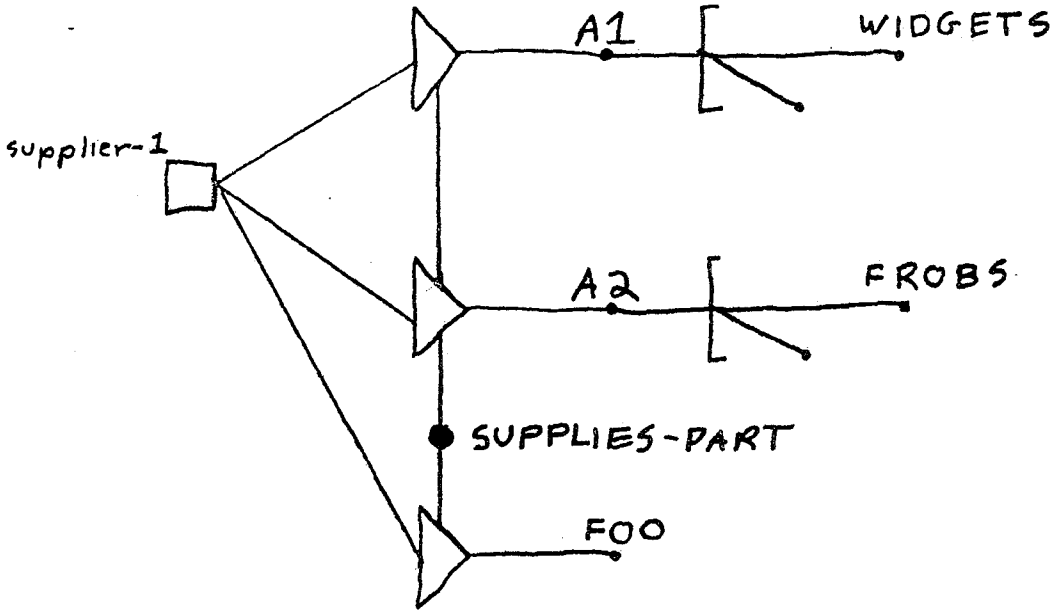

FIGURE 6-1

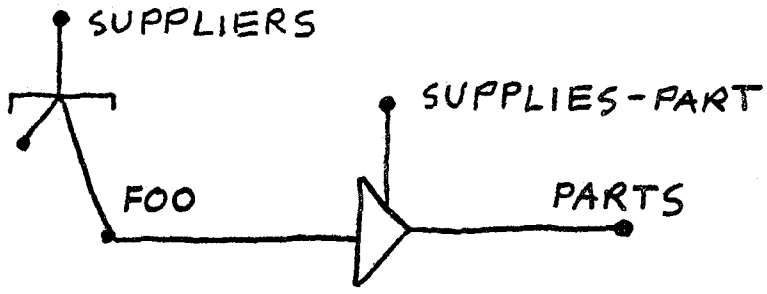

FIGURE 6-2

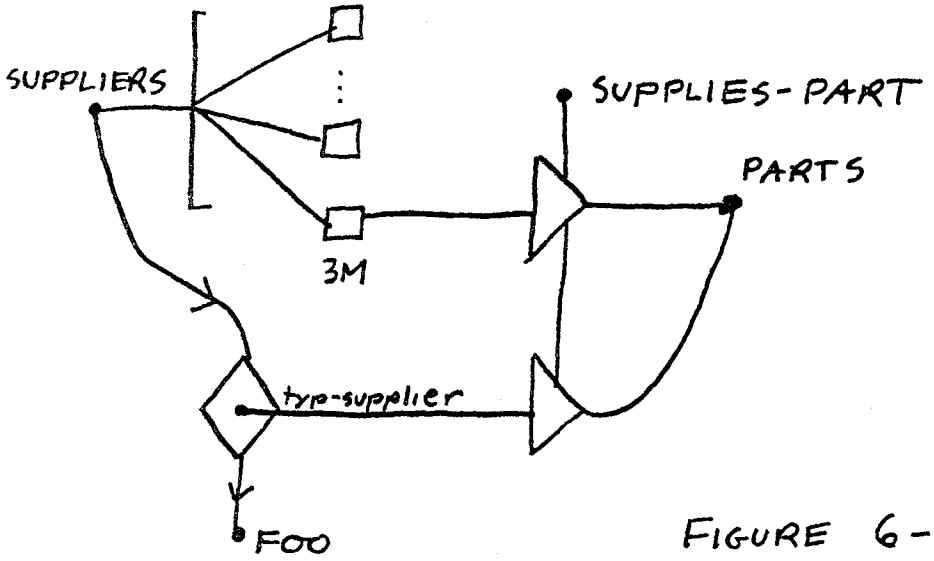

FIGURE 6-3

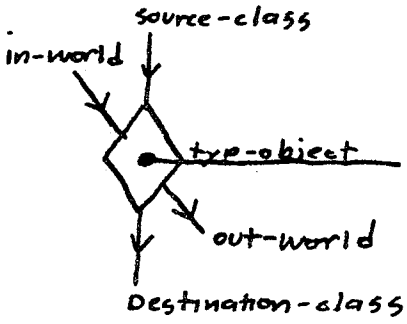

FIGURE 6-4

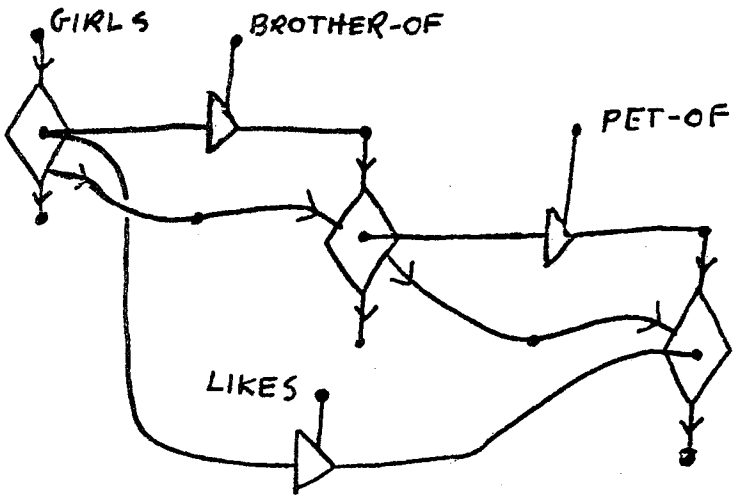

FIGURE 6-5

Bibliography

1. Black, F., "A Deductive Question Answering System" [1964], in Minsky, SEMANTIC INFORMATION PROCESSING.

2. Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as Relational Expressions: SQUARE", IBM Research Report RJ 1291 (#20240), October 1973.

3. Charniak, Eugene, "Towards a Model of Children's Story Comprehension", M.I.T. AI Lab technical report TR-266, December 1972.

4. Date, C. J., and Codd, E. F., "The Relational and Network Approaches: Comparison of the Applications Programming Interface", IBM Research Report RJ 1401 (#21706), June 1974.

5. Fahlman, Scott, "An Hypothesis-Frame System for Recognition Problems", M.I.T. Ph.D. thesis proposal, 1973.

6. Fillmore, C. J., "The Case for Case", in Bach & Harms (eds.), UNIVERSALS IN LINGUISTIC THEORY, Holt, Rinehart, & Winston, Inc.: Chicago, 1968.

7. Hewitt, Carl E., "PLANNER", MAC-M-386, Project MAC, M.I.T., October 1968, revised August 1970.

8. Karp, Richard M., "Reducibility among Combinatorial Problems", in COMPLEXITY OF COMPUTER COMPUTATIONS, R. E. Miller & J. W. Thatcher, ed., Plenum Press: New York, 1972.

9. Lamb, Sydney M., OUTLINE OF STRATIFICATIONAL GRAMMAR, Georgetown U. Press: Washington, D.C., 1966.

10. Lamb, Sydney M., "Linguistic and Cognitive Networks", Yale U. Linguistic Automation Project memo, June 1969.

11. Martin, William, "OWL", M.I.T. Project Mac internal notes, 1974.

12. McCarthy, John, and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in Meltzer & Mitchee (eds.) MACHINE INTELLIGENCE 4, American Elsevier: New York, 1969.

13. McDermott, Drew V., and Gerald J. Sussman, "The CONNIVER Reference Manual", version 2, M.I.T. AI Lab, 1973.

14. McDermott, Drew V., "Assimilation of New Information by a Natural Language Understanding System", M.I.T. AL Lab technical

report TR-291, February 1974[a].

15. McDermott, Drew V., "Advice on the Past-Paced World of Electronics", M.I.T. AL Lab working paper 71, May 1974[b].

16. Mendelson, Elliot, INTRODUCTION TO MATHEMATICAL LOGIC, Van Nostrand: 1964.

17. Minsky, Marvin (ed.), SEMANTIC INFORMATION PROCESSING, M.I.T. Press: Cambridge, Ma., 1968.

18. Minsky, Marvin, "Form and Content in Computer Science", JACM, January 1970.

19. Minsky, Marvin, "Frame-Systems: A Framework for Representing Knowledge", forthcoming, 1974.

20. Quillian, M. Ross, "Semantic Memory" [1967], in Minsky, SEMANTIC INFORMATION PROCESSING.

21. Quillian, M. Ross, "The Teachable Language Comprehender", CACM, August 1969.

22. Raphael, Bertram, "SIR: A Computer Program for Semantic Information Retrieval" [1964], in Minsky, SEMANTIC INFORMATION PROCESSING.

23. Reich, Peter A., "Symbols, Relations, and Structural Complexity", Yale U. Linguistic Automation Project memo, June 1968.

24. Reich, Peter A., "The English Auxiliaries: A Relational Network Description," Yale LAP memo, November 1968.

25. Robinson, J. A., "A Machine-Oriented Logic Based on the Resolution Principle", JACM, October 1965.

26. Schank, Roger C., "Conceptual Dependency: A Theory of Natural Language Understanding", COGNITIVE PSYCHOLOGY, 1972, v. 3, pp. 552-631.

27. Sussman, Gerald, Terry Winograd, and Eugene Charniak, "Micro-Planner Reference Manual", M.I.T. AI Lab memo 203, July 1970.

28. Waltz, David L., "Generating Semantic Descriptions from Drawings of Scenes with Shadows", M.I.T. AI lab technical report TR-271, November 1972.

29. Winograd, Terry, "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language", M.I.T. AI Lab technical report TR-84, February 1971.

30. Winograd, Terry, "Frame Representations and the Declarative/Procedural Controversy", forthcoming, 1974.

31. Winston, Patrick H, "Learning Structural Descriptions from Examples", M.I.T. AI Lab technical report TR-231, September 1970.

32. Zloof, Moshe M., "Query by Example", IBM Research Report RC 4917 (#21862), July 1974.