REPRESENTING "PLACE" IN A FRAME SYSTEM

by

Mark Jay Jeffery

S.B., Massachusetts Institute of Technology
(1975)

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

Massachusetts Institute of Technology

January, 1978

Signature of Author___ _____

Department of Electrical Engineering and Computer Science, January 26, 1978

Certified by___ _____

Thesis Supervisor

Accepted by___ _____

Chairman, Departmental Committee

REPRESENTING "PLACE" IN A FRAME SYSTEM

by

Mark Jay Jeffery

Submitted to the Department of

Electrical Engineering and Computer Science

on January 26, 1978 in partial fulfillment of the requirements

for the Degree of Master of Science.

ABSTRACT

This thesis explores hierarchy as a knowledge representation strategy. I will examine the use of hierarchy to represent knowledge about place, a natural candidate given the importance of the inclusion relationship between places. I shall show that for a first order theory of place, hierarchy proves economical and powerful, effectively expressing regularities of place implicit in statements such as "Boston is in Massachusetts" and "Cambridge is a city". But I shall also show that for an extended theory of place knowledge, hierarchy proves insufficient, burdened by the handling of incomplete, uncertain, and exceptional information, and the effects of data addition and deletion. Finally, I shall propose methods by which a hierarchical representation scheme can cope with dynamic knowledge bases in an *interactive* environment, an essential capability for the practical use of such representations.

Thesis Supervisor: Ira P. Goldstein

Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgements

I thank God for the amazing set of people and circumstances that made the writing of this thesis possible.

First, there's my Mom and Dad -- for many years, they have loved me and given themselves for me, that I might have every opportunity to explore knowledge, love people, and love God.

My roommate Harry has a special place -- there's nothing like a "Happy Hawaiian" to make a difference during thesis time.

I count it a special privilege to have had an advisor who read, re-read, and re-re-read this often-edited paper. Ira helped me to find something I was interested in, found me support so I could work on it, helped me to see what I'd done after I did it, and made sure I wrote it down well enough so others could read it. These were not easy tasks!

All my friends who have consistently expressed their concern for me deserve a special award for perseverance -- for how many months (semesters!) have they heard "I'm almost done"? I'm especially thankful for their prayers in the midst of the many personal struggles of this last semester.

The computational resources of the MIT AI Laboratory are extraordinary (all things considered), and what would I have done without the help of the unusually approachable, patient, and competent people there? A special thanks to Mark Miller and Matt Mason who listened to me discover my thoughts, or at least let me think they were my thoughts. Finally, it was Bruce Roberts who made it possible to approach the system with some degree of hope, and the time he took to help me clarify and redirect my thinking was invaluable.

I'm thankful that I live in a country with places like MIT where I can pursue this kind of research. Perhaps someday I'll be able to make a contribution which will help give others the same opportunity to pursue their goals as I have had to pursue mine.

Finally, I thank God for a sound mind and body, and for his patient discipline in helping me apply myself to the task of writing this thesis. His presence has made all the difference in the world.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

Table of Contents

I. What's It All About? 6

Common Sense About Place 6
Putting It All Together: A "Near" Scenario 8
Relation to Previous Work 10
Organization of This Document 12

II. A Basic Theory of Place 13

The Inclusion Relation 13
Place Generalization: Rank 14
The Rank-inclusion Property 16
A Simple Concept of Near 17

III. A Basic Frame Implementation of Place 20

The Inclusion Relation 24
Procedural Attachment 27
Place Functions 33
Everything in Its Place 37

IV. Refining the Theory of Place 38

Exceptions of Inclusion 38
Exceptions of Rank 40
Refining Near: Directionality, Adjacency, and Size 45
New Issues Raised 54

V. Refining the Frame Implementation of Place 57

Ignorance Is Not Bliss 57
Modification Is Worse Than Addition 65

VI. User Interaction with a Hierarchical System 72

Supplying New Information 72
Top-level Control 83

Appendix A Actual "Near" Scenario 87
Appendix B Sample Place-frames 88
Appendix C Sample Place System Functions 93

Bibliography 99

Table of Figures

Chapter II -- A Basic Theory of Place

Fig. 1 Definition of Nearness 18

Chapter III -- A Basic Frame Implementation of Place

Fig. 2 The "A Kind Of" Hierarchy of the PLACE System 23
Fig. 3 Using SUB-SUPER Links to Capture Inclusion 26
Fig. 4 The Place-frames Inclusion Hierarchy 28
Fig. 5 SUB and SUPER Pointers Are Added as Pairs 29
Fig. 6 Restructuring Inclusion Relationships 30
Fig. 7 Eliminating Hidden Inclusion Relationships 31
Fig. 8 The Formal Algorithm 31
Fig. 9 Recursive Application of the Rank-inclusion Principle 33

Chapter IV -- Refining the Theory of Place

Fig. 10 Possible Inclusion Relations with PARTLY-IN 39
Fig. 11 Refining Near: Directional Information 46
Fig. 12 Refining Near: Adjacency and Directional Information 46
Fig. 13 Refining Near: Adjacency, Size, and Directional Information 46
Fig. 14 Refining Near: Adjacency, Direction, Shape, Orientation, and Position 47
Fig. 15 Directional Information and Inclusion 48
Fig. 16 An Exception to Dominant Directionality 49
Fig. 17 Inappropriate Directional Modifiers 50
Fig. 18 Appropriate Directional Modifers 50
Fig. 19 Using Nearness to Suggest Adjacency 51

Chapter V -- Refining the Frame Implementation of Place

Figs. 20, 21, & 22 Using Rank-inclusion to Merge SUB-SUPER chains 59
Fig. 23 The Initial Merging Algorithm 60
Fig. 24 A Missing AKO 61
Fig. 25 The Merge Is Completed 61
Fig. 26 Multiple Missing AKOs 62
Fig. 27 The Desired Merge 62
Fig. 28 Subtleties of "Passing Through" 63
Fig. 29 The Desired Merge 63
Fig. 30 Conditions Necessary for a Merge 63
Fig. 31 Also Sufficient for a Merge 63
Fig. 32 Previously Unmerged Chains 64
Fig. 33 A Merge Was Possible 64
Figs. 34 & 35 An Simple Example of Merging 66

I. What's It All About?

This thesis explores hierarchy as a knowledge representation strategy. I will examine the use of hierarchy to represent knowledge about place, a natural candidate given the importance of the inclusion relationship between places. I shall show that for a first order theory of place, hierarchy proves economical and powerful, effectively expressing regularities of place implicit in statements such as "Boston is in Massachusetts" and "Cambridge is a city". But I shall also show that for an extended theory of place knowledge, hierarchy proves insufficient, burdened by the handling of incomplete, uncertain, and exceptional information, and the effects of data addition and deletion. Finally, I shall propose methods by which a hierarchical representation scheme can cope with dynamic knowledge bases in an *interactive* environment, an essential capability for the practical use of such representations.

Common Sense About Place

Place knowledge clearly involves some element of hierarchy. For example, a person may refer to his or her home in terms of the city it's in, or the metropolis, state, section, or even country it's in, such as Center Line, Detroit, Michigan, Midwest, or USA. The context set by the relative generality of the kinds of places being talked about determines which generalization is appropriate:

Within the home city of Center Line:	"When I come home, mom fixes dinner."
In Detroit, referring to Center Line:	"At home, I don't have to be careful to get home before dark."
In Mass., referring to Michigan:	"Back home, we passed a Bottle Bill."
In the East, referring to Midwest:	"Back home, it's a milk shake."
In Europe, referring to the USA:	"Back home, the water's OK to drink."

From a representational perspective, a system is needed which conveniently handles speaking about places at various levels of generality.

These levels also have interrelationships associated with them that are implicitly used in reasoning about places. For example, if a person is in a particular city, then that person will be in the same country no matter where he or she is in the city. This is almost always a valid deduction, although there are exceptions, such as Berlin. These relationships (and their occasional exceptions) must be made explicit if they are to be used wisely by the computer.

Finally, each level of specificity has associated with it various properties. For example, countries form a disjoint class, but sections, like New England and East Coast, can overlap. Levels differ in typical size which in turn affects typical means of travel: in rooms, one walks; in cities one takes cars, buses, and taxis; in countries one takes cars, buses, trains, or planes. These various means of travel in turn have implications for scheduling, such as travel time, cost, and preparation, as well as considerations like gas, keys, ticket, luggage, license, visa, or good weather. An adequate representation for place will have to provide means to represent the properties of each level.

Thus we see that a representation for place demands the accommodation of various levels of generality, the ability to assert relationships between those levels, and the association of particular properties with each level.

Putting It All Together: A "Near" Scenario

A frame representation language was employed to implement a system which meets these demands. This system is called the PLACE system, and it is based on a hierarchy of frames with attached procedures. To illustrate this system and some of the strengths and weaknesses of hierarchy that it explores, an interaction appears below which exercises the system's knowledge of near. An Anglisized account of the PLACE system's reasoning is provided for comprehensibility. The LISP version of this interchange is found in Appendix A.

The PLACE system considers two places to be "near" one another in a given region if they both are in a common sub-region:

Is Boston near Cambridge relative to Massachusetts?

> *Yes, because they are both in the Metro-Boston sub-region of Massachusetts.*

Is Boston near New York relative to North America?

> *Yes, because they are both in the East Coast sub-region of North America.*

The conceptualization of near based on hierarchy can give rise to errors. For example, the existence of the sub-region East Coast which contains both New York and Boston causes the system to consider them as being nearer to each other than Montreal and Boston, which are not contained in a known sub-region:

Is Boston near Montreal relative to North America?

> *No, because they have no sub-region of North America in common.*

In fact, there is not much difference, compared to the size of North America -- Boston is 185 miles from New York, and 260 from Montreal.

The inadequacy of a NEAR predicate based solely on hierarchy is even more obvious in the following example, where two cities separated only by a river are not considered near:

Is Detroit near Windsor relative to North America?

No, because they have no sub-region of North America in common.

We shall see that the constraints imposed by hierarchy make it very difficult, if not impossible, to handle these situations with a purely hierarchical scheme. Perhaps the simplest solution is to tell the computer to make a specific exception to the general hierarchy. For example, the system could be told to consider Detroit to be near Windsor relative to any area of type state or province, or larger:

Assert Detroit is near Windsor relative to
including areas of type state or province.

Done.

Is Detroit near Windsor relative to Michigan?

Yes, because they are known to be near
relative to including places of type state.

While this approach requires special knowledge for each pair of places, it does allow a better approximation of the meaning of near.

Hierarchy integrates well with these exceptions. For example, it allows the exception to apply automatically to the hierarchy of places within Detroit and Windsor:

Is Cobo Hall (a building in Detroit) *near Windsor Mutual*
(an imaginary building in Windsor) *relative to Ontario?*

Yes, because they are in places known to be near
relative to including places of type province.

A difficulty, however, with making explicit exceptions is that they must be asserted at the proper level of generality. The system should have been told that Metro-Detroit is near Metro-Windsor, relative to the size of a state or province. That would prevent errors of the following sort:

> *Is Center Line* (a suburb of the Metro-Detroit area) *near*
> *Fairfield* (an imaginary suburb of Windsor) *relative to Michigan?*
>
> *No, because they have no sub-region of Michigan in common.*

Chapters II and III of this essay develop a first order theory of place knowledge and a corresponding frame representation. Chapters IV and V criticize this first order theory, discussing more refined concepts that place knowledge must encompass, and more complex processes on frames that the representation scheme must accommodate.

Relation to Previous Work

The "room theory" of David Rumelhart (1974) is similar to the concept of near employed in the present system: two places are near if they are in a sub-region of the context-defining place. Rumelhart defines that theory as follows:

> The room theory posits the existence of a psychological room relative to which distances are reckoned. ...the room corresponds to the smallest geographical region that encompasses both the reference location of the conversants (normally their physical location) and the locations of the places in question. Places that are close to one another are those whose distance is small relative to the size of the size of the room. ... The rule is find the smallest room which just includes the reference location and the answer location. The appropriate answer is the next smallest geographical unit which contains the location in question, but excludes the reference location. (Rumelhart, 1974)

The simplicity of this concept of near gives it wide applicability and ease of implementation, making it a reasonable starting point for a first order theory of near.

Murray Denofsky's work (1976) explores the concept of "near" as a judgment about physical distance. He does so not just from the perspective of place, such as buildings in a city, but also for many other entities, such as pages in a book, letters on a page, or cars on the road. This near weighs factors such as purpose of the judgment, dimensions of the object, absolute distance, and relative distance to other objects, ranges, and standards. Denofsky's approach is quite quantitative in that it depends on having specific numbers available for its computation, whether they be actual values, or default values. The PLACE system does not depend on numerical information, but only on the inclusion relationships involved. Hence it is less accurate, but more often applicable when dealing with partial information. Extending the PLACE system so that it can use numeric information when it is available would not be too difficult.

Ben Kuipers (1976, 1977) has explored place representation at the level of streets and neighborhoods, with an emphasis on being able to learn and find routes between places given a local and incomplete (but correct) description of the streets and intersections of the area. A hierarchy of specific places (Boston --> Massachusetts --> USA) is used to determine travel routes between large areas, such as the East and West Coasts. These routes are refined at progressively lower levels in the hierarchy until an exact route, say from the Stanford AI Lab to the MIT AI Lab, is found.

My current work differs from Kuipers' in that it uses an abstraction of hierarchy of specific places (city --> state --> country) for the purpose of reasoning about place and travel. For example, information about the rank of a place in this abstract hierarchy is used to deduce facts about inclusion relationships (MIT is a building complex, and hence can't contain a city), and inclusion relationships are used to put

restrictions on what rank a place can have. There is also a concern for dealing with incorrect as well as incomplete information, and for assisting the user with options and examples for modifying that information. Finally, the issue of detailed route-finding, which is a major element of Kuipers' work, is not considered in my work.

Terry Winograd (1975) examines a frames approach for embedding a hierarchy of time in a frames system. Many of the problems and advantages of his approach apply to the current system, including those concerning the use of procedurally-attached knowledge and inheritance. The NUDGE system (Goldstein and Roberts, 1977) is a knowledge-based scheduling system which attempts to integrate common sense about time, place, meetings, activities and people using just this type of system. The PLACE system finds practical use as a module in the NUDGE system.

Organization of This Document

Chapter II explores the formalization of knowledge about place, identifying several major regularities. Chapter III describes the implementation of this theory in a frame-based hierarchical system. The limitations of the place theory are explored in chapter IV, and suggestions are made concerning improvements. Chapter V demonstrates the limitations of a frame representation by exploring the impact of incomplete and changing data on the hierarchical structure of the data base. Finally, chapter VI explores methods for dealing with interactive classification within a hierarchical system, and discusses some issues of user interaction and control which are easily mishandled in such systems.

II. A Basic Theory of Place

In this chapter, I define formal relations "IN" and "NEAR" to capture the regularities of place implicit in the corresponding ordinary English relations. In order to make full use of the reasoning power these regularities provide, exceptions to them which the ordinary use of "in" and "near" allow will not be considered. We will pursue these exceptions as we refine the theory in chapter IV.

The Inclusion Relationship

The most fundamental concept needed for dealing with place is that of inclusion -- Detroit is in Michigan, the USA is in North America. The semantic sense of "in" that is important here is the relation "is a part of", not the relation "physically inside". For example, the city of Center Line is completely surrounded by the city of Warren, but it would generally be judged incorrect to say "Center Line is in Warren". Hence in formally defining IN, we make the distinction that even though one country or city is totally surrounded by another, it is not IN it unless it is a part of it.

Another regularity usually exhibited by the use of "in" is that it is a binary relation. Either one place is *completely* within another, or it isn't in it. Hence we add to the definition of IN that it means "completely in", not "partly in". This is not always the case -- part of Kansas City is in Missouri, and part is in Kansas. This situation is not expressible with our definition of IN as "completely in".

Combining the two characteristics of IN that we have discussed, we arrive at the definition that IN means "completely a part of". Consequently, if P' is IN P, then it is reasonable to think that any place P" IN P' is also IN P -- all subparts of P' are IN P. For example, if Boston is IN Massachusetts, then any place IN Boston is IN

Massachusetts. This is equivalent to saying that IN is a transitive relation. Transitivity is a very powerful reasoning tool to have, so cases such as Kansas City will be handled as exceptions.

Place Generalization: Rank

Another regularity of place is the typical use of place generalization. To reflect that regularity, I shall introduce the concept of "rank", which refers to the level of generalization. The most basic levels seem to be those of room, building, city, state, and country. Generalizing slightly and including some less often used categories, the following rank levels were selected:

1. meeting-place	smallest kind of place considered; it has the property that it takes no time to travel within a single meeting-place, and people at the same meeting-place are always aware of one another.
2. building-area	sub-parts of a building, such as an entry or floor.
3. building	
4. complex	collection of several buildings
5. city	
6. metro-area	collection of several cities in close relation
7. political-unit	subdivision of a country, such as state, province.
8. section	area of a country, such New-England, Midwest.
9. country	
10. continent	
11. world	for completeness.

The properties of generalization that this categorization seeks to capture are the following: (1) the typical levels of generalization encountered in everyday dealing with place are present; (2) the complete range of place sizes is covered; and (3) the classification of places and classes of places is unique. Hence every place P has a single rank R(P) corresponding to the level the place falls into, "world" being the highest rank. The first property of including all common levels is an obvious requirement for any first

order theory. The second property of spanning all sizes makes it possible for the properties of places which depend heavily on size (such as travel time and nearness) to vary as smoothly as possible, with no large jumps. Finally, the third property of uniqueness means that there is no ambiguity in determining rank-dependent properties; hence reasoning about places can proceed straightforwardly.

There are several other *inherent* properties of rank which we will exploit. First, as has already been mentioned, each rank has factors associated with it which determine nominal travel means and time. For example, it just a fact that building have stairs, elevators, and sometimes escalators, while cities have cars, buses, taxis, and sometimes subways. Secondly, ranks have a natural inclusion order -- a place is never in another place of lesser rank: a country is never IN a city; a city is never IN a building. Hence the general relation holds that if the rank of P is greater than the rank Q, P cannot be in Q. This order property will prove useful in reasoning about inclusion relationships.

In addition to these basic properties, other useful properties can depend on rank information. For example, if it is known that the members of a rank are mutually exclusive, then it can be checked that at most one member of that level contains a given place. For example, Boston can not be IN both Maine and Massachusetts, since states are a mutually exclusive category. If it is known that a particular level spans another (that at least one member of that level contains all places of a certain set), additional deductions can be made. For example, all cities of the USA are IN states, since states span the USA. If a level's known members are also known to form an exhaustive set, further deductions are possible, since the members are then known to span that level. For example, knowing that you knew all the states IN New England would let you deduce that Michigan isn't

IN New England, since it wasn't in your list. As it turns out, the members of most ranks are mutually exclusive and collectively exhaustive, a comment perhaps on how people organize their descriptions of the world.

Dividing places into levels also creates expectations for inclusion relationships, since it is known which levels are of ranks above and below a given one. This information can be used in asking more intelligent questions. Not only can one avoid asking questions which violate the rank order property, such as "What building is Boston in?", but one can ask questions in terms of the most appropriate rank, such as "What building is the AI LAB in?", instead of "What continent is it in?".

Lastly, the levels enable one to generalize up or down to the rank appropriate for a given place context. For example, one would say "the drinking age in Michigan is 18", not "the drinking age in room 214 is 18".

The Rank-inclusion Property

The concepts of rank and inclusion interact in an interesting way -- with few exceptions, regions of one rank form boundaries along the boundaries of regions of lesser rank. For example, countries such as the USA are divided into states, and no states are split by a country boundary. Cities are not split by state or country boundaries (Kansas City is one of a small number of counter examples), buildings are seldom split by city boundaries, and a room is generally not split among several buildings. I call this regularity in the structure of places the "rank-inclusion" property.

If a place can not be split by the boundaries of places of greater rank, then it acts as a unit -- either all in or all out. Hence if part of a place is known to be IN a place

of greater rank, then the entire place must be IN the greater ranked place. This is a powerful reasoning aid. For example, if Detroit is known to be IN the USA, and Detroit is also known to be IN Michigan, then Michigan must also be IN the USA. Otherwise the USA would be spliting a region of lesser rank, Michigan, into two parts -- a part containing the Detroit part of Michigan, and the rest of Michigan. To summarize, if any part of P is IN Q, and the rank of P is less than the rank of Q, then all of P is IN Q. An implication of this is that if two places P and Q both contain a third place, and the rank of P is less than the rank of Q, then P is IN Q. For example, the USA and New England both contain Boston; since the rank of New England is less than that of the USA, New England is IN the USA.

The rank-inclusion property is violated by places like Kansas city, or buildings that straddle state lines. We take up these exceptions in chapter IV.

A Simple Concept of Near

Evolving an adequate understanding of the concept near is pivotal to developing a practical theory of place. Complexity arises because its meaning varies with context: my room is near his room; Vermont is near New Hampshire; Japan is near India. The formal definition of "near" used here is based solely on the ideas of rank and inclusion as presented thus far. One place is used to establish the context, and nearness is defined in terms of the internal place structure of this context-establishing place unit (this follows the definition suggested by Rumelhart, 1974). More specifically, places are considered to be near each other if they all lie in some sub-unit of the context-establishing place unit: places P(1)...P(n) are NEAR one another relative to Q if there

exists a Q' IN Q such that P(i) is IN Q' for i=1 to n.

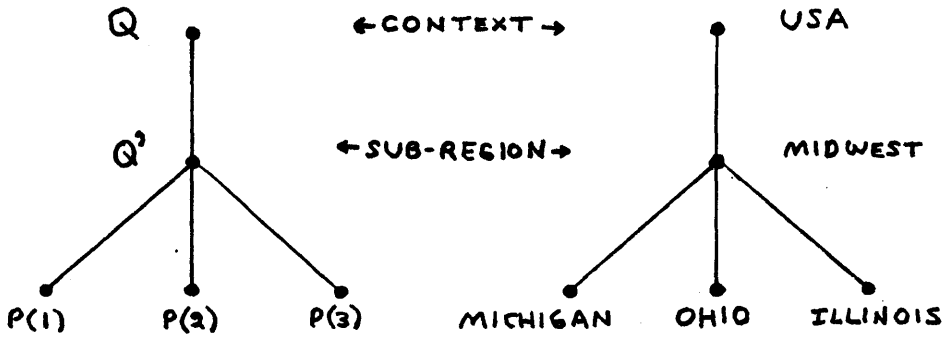


Fig. 1 Definition of Nearness

Michigan is NEAR Illinois relative to the USA since they are both IN the Midwest. Similarly, Detroit is NEAR Lansing relative to the Midwest, since they are IN the same state; but Detroit is not NEAR Chicago, since there is no sub-unit of the Midwest that contains both Detroit and Chicago. One could be invented, however, such as "Great Lakes States".

This NEAR relation depends solely on the inclusion relationships that are known, and uses no distance metric. Hence it is easy to implement, and often gives reasonable results, as long as only general magnitudes of distance are all that matters (see Denofsky, 1975, for a more metric oriented approach).

This approach has its shortcomings: while Detroit and Windsor are separated only by the Detroit River, they would not be found to be "near" each other using this approach, and no region can be invented which allows that deduction that does not also violate the rank-inclusion property. We will take this up in chapter IV.

We have examined the concepts of "in" and "near" and defined two corresponding formal relations IN and NEAR which capture some of their regularities. IN is defined to mean "completely a part of", and is transitive. NEAR is defined in terms of IN. Place types have been organized into a collection of ordered ranks which reflect the properties of general categories of places, and which constrain inclusion relationships. In the next chapter, I will explore a frame representation for these concepts.

III. A Basic Frame Implementation of Place

The simple theory of place described in chapter II was implemented as the PLACE system in the frame representation language FRL developed by Goldstein and Roberts (Goldstein and Roberts, 1977; Roberts and Goldstein, 1977). As was anticipated, the hierarchical orientation of the language was well-suited to the hierarchy of place relationships. Knowledge could be placed at one place in the knowledge base, corresponding to its level of generality, and be accessed automatically, via inheritance, when needed by other parts of the system. This modularity and matching with appropriate level of generality made it easy to understand and modify the system. Finally, procdural attachment proved useful for automatically maintaining the structural integrity of the hierarchy when adding new places. In this chapter, I shall describe how this fortuitous matching between FRL and place relationships was carried out. In chapters IV and V we shall see how uncertain, incomplete, exceptional, and changing information adversely affect the effectiveness of the place hierarchy and FRL's maintenance of it, and chapter VI proposes methods for handling the special difficulties one encounters when dealing interactively with a hierarchical system.

In FRL, knowledge representations are built from individual packages of information, called frames, which are nested association lists. Each frame has a number of slots which reflect various aspects of the thing the frame represents. For example, BUILDING might have slots for number of floors, instances of specific buildings known by the system, travel information for getting around in a building (stairs, elevators, walking), and common facilities of a building, such as lavatories. Each slot in turn has

facets, which categorize the kinds of information the slot contains. For example, one facet type is reserved for the actual *value* of a slot ($value); another is used for *procedural knowledge* which says what to do when a new value is added to that slot ($if-added); and yet another specifies *requirements* that any value for that slot must satisfy ($require). There are other facets which we will describe latter. If there is no $value facet under a certain slot, a value can be inherited from a frame that the current frame is a kind of. For example, an office has the facility of light-fixtures because its "a kind of" (AKO) slot has value ROOM, and ROOM contains the value LIGHT-FIXTURE under its facility slot. A number of simplified sample frames appear below; a larger listing of frames appears in appendix B.

The OFFICE frame demonstrates the nested association list structure of frames, and the use of the value and default facets:

```
(OFFICE     (AKO              ($VALUE      (ROOM)))
            (INSTANCE         ($VALUE      (NE43-319) (NE43-823)))
            (FACILITIES       ($VALUE      (DESK) (WASTE-BASKET)))
            (ACTIVITY         ($DEFAULT    (PAPER-WORK)))
            (AVAILABLE        ($DEFAULT    ((INTERVAL 9AM 5PM)))))
```

The PLACE frame is the most general one in the place hierarchy. All places can inherit its slots and their associated facets. Note the use of the if-added and require facets:

```
(PLACE      (AKO              ($VALUE      (THING))
                              ($IF-ADDED   ((CHECK-IMPROPER-SUPERS))
                                           ((ADD-INSTANCE)))
                              ($IF-REMOVED ((REMOVE-INSTANCE))))
            (INSTANCE         ($VALUE      (ROOM) (CITY) (SCHOOL)...)
                              ($REQUIRE    ((RESTRICT-AKO 'PLACE))))
            (SUPER-PLACE      ($IF-ADDED   ((CHECK-IMPROPER-SUPERS))
                                           ((SUGGEST-AKO)))
                              ($REQUIRE    ((> (RANK :VALUE)
                                               (RANK :FRAME))))))
```

The CITY frame illustrates the embedding of travel information in a frame:

```
(CITY      (AKO              ($VALUE      (PLACE)))
           (INSTANCE         ($VALUE      (CAMBRIDGE) (DETROIT)))
           (SUPER-PLACE      ($REQUIRE    ((>= (RANK :VALUE)
                                               (RANK 'CITY)))))
           (INTRA-TRAVEL     ($DEFAULT    (CAR  (TIME: 20 MINUTES)
                                                 (COST: 35 CENTS))
                                          (TAXI (TIME: 20 MINUTES)
                                                 (COST:  4 DOLLARS))
                                          (BUS  (TIME: 25 MINUTES)
                                                 (COST: 25 CENTS)))))
```

The CAMBRIDGE frame contains a small amount of very specific information; other information can be inherited from the more general CITY and PLACE frames:

```
(CAMBRIDGE (AKO              ($VALUE      (CITY)))
           (SUPER-PLACE      ($VALUE      (METRO-BOSTON)))
           (SUB-PLACE        ($VALUE      (MIT) (HARVARD) ... )))
```

As we have seen from the examples, each place-frame is "a kind of" (AKO) one of the rank-determining levels in the place hierarchy, by inheritance, if not directly. For example, PAPA-GINO'S is AKO RESTAURANT which is in turn AKO BUILDING. Each rank level, such as CITY or COUNTRY, is AKO PLACE. The AKO tree that this produces appears on the next page (Fig. 2). The tree is abbreviated, but serves to illustrate the principles involved. It is this AKO tree structure which is used for inheriting basic properties of the various categories, such as appropriate travel means, and which permits over-riding those properties in more specific cases by placing information further down in the tree.

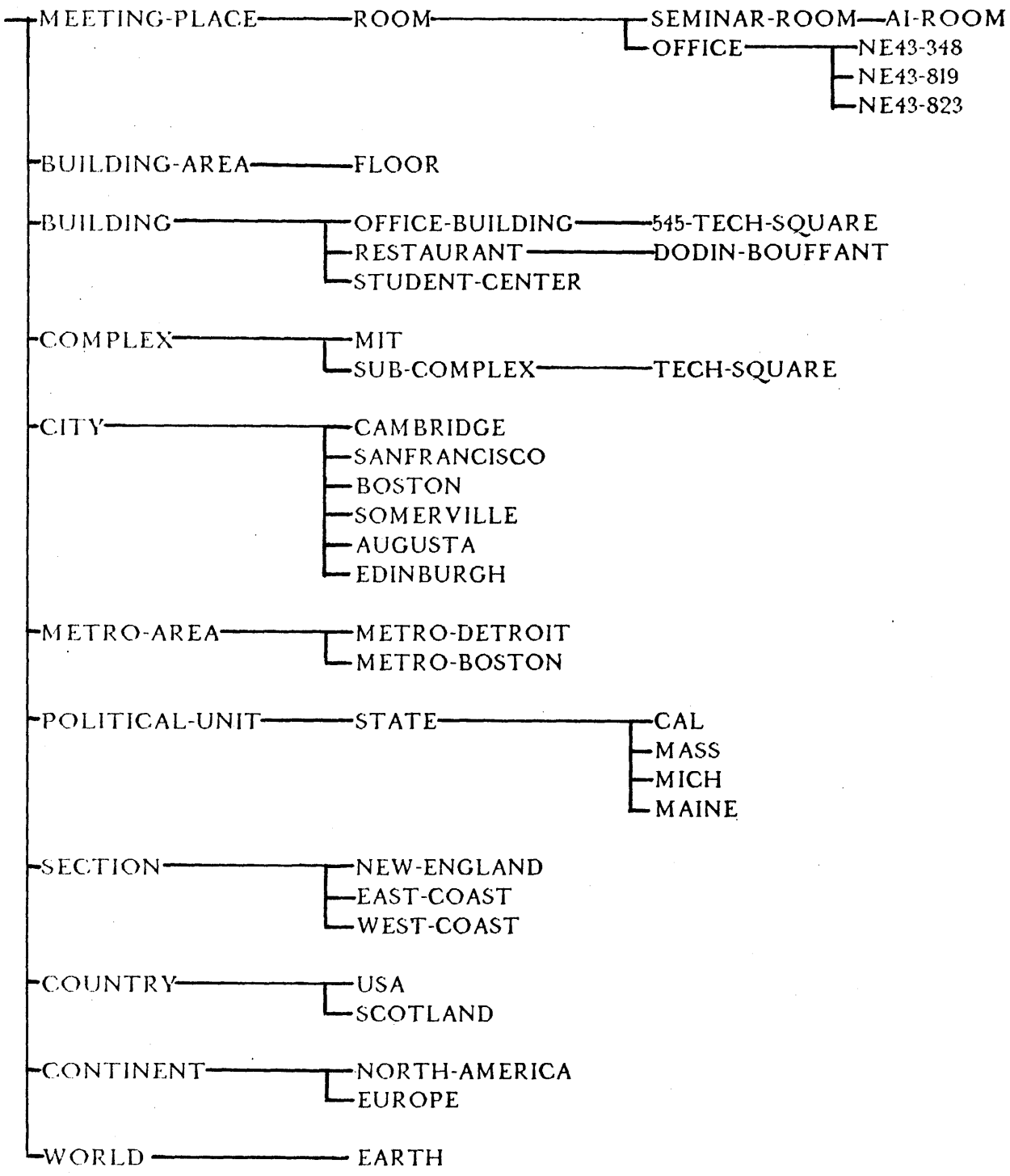


Fig. 2 The "A Kind Of" Hierarchy of the PLACE System

(All the leftmost places are "a kind of" PLACE)

The Inclusion Relation

There are two options as to how to capture inclusion relationships within the frames system. One is to put the information completely internal to each place-frame; the other is to embed the inclusion information in the links between frames. The former method would mean putting the complete inclusion chain for that place inside the frame; for MIT, this would look as follows:

```
(MIT    . . .
        (SUPER     ($VALUE   (CAMBRIDGE METRO-BOSTON MASS EAST-COAST
                             NEW-ENGLAND USA NORTH-AMERICA EARTH)))
        . . . )
```

This approach has the virtue that inclusion does not have to be transitive to be stored this way, making exceptions easier to represent. But it was rejected for several reasons. First, it is highly redundant (it would be identical for every building in Cambridge, even longer for each room in Cambridge). This means that storage space is being used needlessly. Second, the redundancy makes it very hard to integrate changes into the hierarchy. For example, suppose it was desired to add to the place-frames system that Massachusetts was in the section of the USA called the East Coast. This new place information would have to be added to the inclusion list of every single place in Massachusetts. Similarly, deleting that information would involve finding each place-frame in which it occurs and removing it. This difficulty would disappear in a static system, but for an evolving one such as this, it is inappropriate.

Embedding inclusion relationships in the frame links reduces the redundancy: Massachusetts can be made part of the East Coast by adding just the one fact that it is linked to the East Coast via a "SUPER-PLACE" link. All the places that make up

Massachusetts would automatically be a part of the East Coast by the transitivity of inclusion. There is an increase in computational complexity, however, in determining SUPER-PLACEs, since rather than having a ready-made list, the chain of transitive inclusion links must be traced. It was decided that this increase in complexity was more than compensated for by the increased modularity of information in the linked frames approach and by the ease of modification. This approach also results in increased storage efficiency: since there are many more places of lower ranks than higher ranks (many more rooms than countries than continents) and about ten rank levels, most place-frames would require 80-90% less storage for the inclusion relation. These advantages of modularity and storage efficiency are typical of hierarchical systems, and are similarly reflected in the AKO hierarchy. The representation of a non-transitive relationship, which the first approach allowed, will be reconsidered in chapter IV for handling exceptions to transitivity.

Sub-super Links

The PLACE system embeds inclusion relationships as a transitive link called the SUPER-PLACE link, as discussed above. It occurs in pairs with its inverse, the SUB-PLACE link, making it easy to trace inclusion chains both up and down (these links will henceforth be abbreviated as SUPER and SUB). Using the AKO information we examined earlier, we see that MASS is AKO STATE (which is AKO POLITICAL-UNIT), and has SUPERs NEW-ENGLAND and EAST-COAST, and SUB METRO-BOSTON:

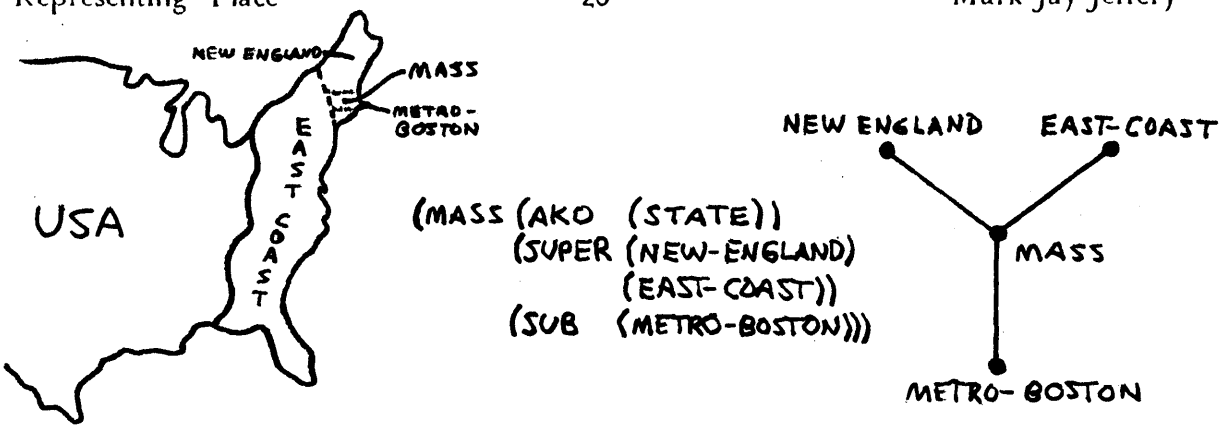


Fig. 3 Using SUB-SUPER Links to Capture Inclusion

A requirement on the general PLACE frame (which every place-frame has, by inheritance) enforces the property that the rank of a place-frame must be less than or equal to that of its SUPERs, and greater than or equal to that of its SUBs (see the PLACE frame at the beginning of the chapter).

Frames can also have preferences on their SUB and SUPER links which indicate what ranks it is desirable to have fill those slots. For example, one would generally want to narrow down the location of a city as much as possible, and therefore one would rather know what metro-area or state its in than what country or continent. Hence metro-area and political-unit occur under the preference facet of the SUPER slot of CITY. These are only preferences, however, not requirements. The including country or continent is permissible, if that's all that's known. Here is how the preference facet of CITY would appear:

```
(CITY   (SUPER  ($PREFER  ((RESTRICT-AKO
                            '(METRO-AREA POLITICAL-UNIT))))
                ($REQUIRE ((>= (RANK :VALUE)
                               (RANK 'CITY))))))
```

The exact ordering for the rank levels is implicitly stored in the preference facets of the SUPER slots. However, a function called RANK has access to an ordered

list called PLACE-TYPES which explicitly contains that ordering. This results in greater efficiency through less searching and less computation in establishing rank ordering, although it is not in keeping with a *pure* frame representation of knowledge.

The current SUB-SUPER structure of the place-frames system is mapped out on the next page (Fig. 4).

Procedural Attachment

There are certain disciplines maintained in the place-frame hierarchy to facilitate data integrity and efficient computation. First, IF-ADDED and IF-REMOVED methods are used to insure inclusion information is properly available -- that an upward SUPER pointer is paired with a corresponding downward SUB pointer. Every place inherits these procedures from the top-level place-frame PLACE, thereby exploiting the hierarchical structure of the PLACE system:

```
(PLACE      (AKO        ($VALUE         (THING))
                        ($IF-ADDED      ((ADD-INSTANCE)))
                        ($IF-REMOVED    ((REMOVE-INSTANCE))))
            (SUPER      ($IF-ADDED      ((ADD-SUBS)))
                        ($IF-REMOVED    ((REMOVE-SUBS))))
            (SUB        ($IF-ADDED      ((ADD-SUPERS)))
                        ($IF-REMOVED    ((REMOVE-SUPERS)))))
```

For example, if WARREN, a city, is added to the system with SUPER METRO-DETROIT, a SUB pointer is automatically added (by the IF-ADDED method on the SUPER slot of WARREN) to METRO-DETROIT, pointing back to WARREN:

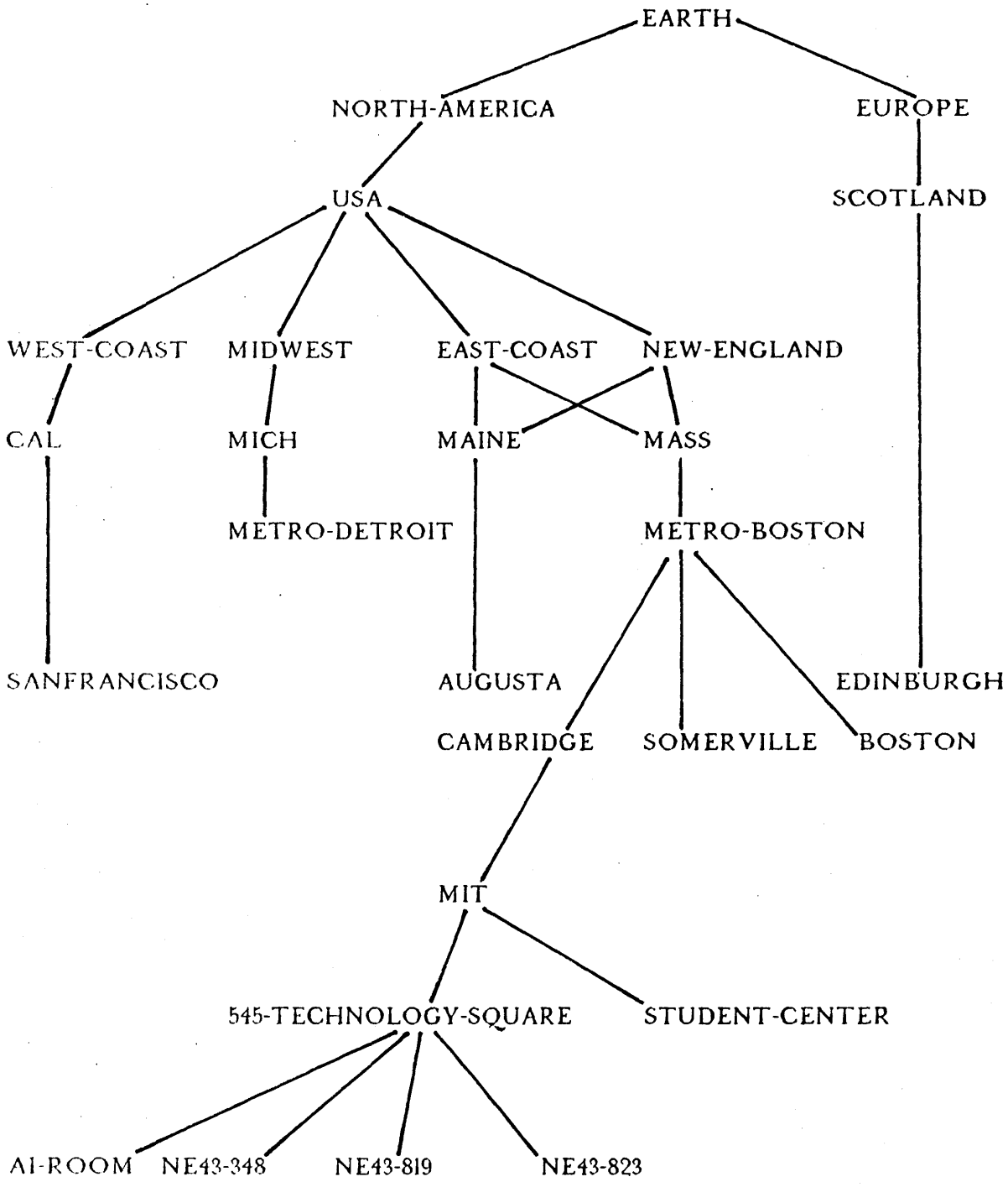


Fig. 4 The Place-frames Inclusion Hierarchy

This is the hierarchy represented by the SUB-SUPER slots of the place-frames.

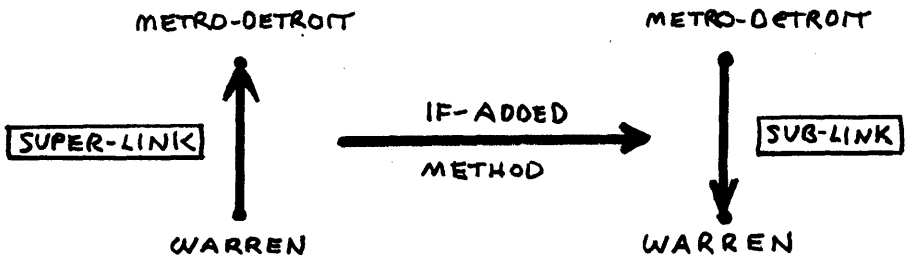


Fig. 5 SUB and SUPER Pointers Are Added as Pairs

Similarly, SUB-SUPER pointers are deleted in pairs. This guarantees that any search which goes up the inclusion chain will get the same results as any that goes down the chain. A similar discipline is at work for AKO-INSTANCE links: if HARVARD, which is AKO SCHOOL, is added to the system, then SCHOOL automatically gains an INSTANCE pointer to HARVARD. Hence we see that attached procedures provide a convenient way of maintaining links in a hierarchical structure.

The rank-inclusion property is also a part of the place-frame discipline, since it can be used to properly locate new places in the inclusion hierarchy. Recall that this property says that if any part of P is IN Q, and the rank of P is less than the rank of Q, then all of P is IN Q. Hence if P is IN several different places which all have different ranks, it is possible to determine the inclusion relationships between these places. For example, if CAMBRIDGE is IN METRO-BOSTON, MASS, and NORTH-AMERICA, one can deduce that METRO-BOSTON must be IN MASS, and that MASS must be IN NORTH-AMERICA:

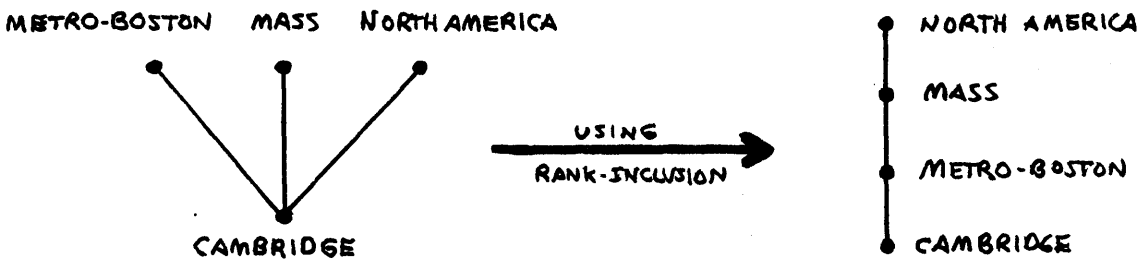


Fig. 6 Restructuring Inclusion Relationships

In the place-frames system, each time a new SUPER is added to a place, a check is made to see if such a hidden inclusion relation holds. If it does, inclusion links are manipulated to reflect the new relationship.

This restructuring is accomplished by maintaining a discipline which guarantees that no hidden inclusion relationships exist. This discipline consists of enforcing the constraint that the SUPER slot of a place-frame hold only its most immediate SUPERs. These will all necessarily be of the same rank, and hence can not contain any hidden inclusion relationships deducible from the rank-inclusion property. (There is no similar discipline at work regarding SUBs -- it *is* possible for SUBs to have varied rank.) More formally, its says that for each place P with SUPER Q (P IN Q holds), there is no known P' such that P is IN P' which is also IN Q. For example, if CAMBRIDGE has SUPER MASS, then there should be no place IN MASS which contains CAMBRIDGE.

This constraint is maintained by comparing the rank of each SUPER that is added (via IF-ADDED and IF-REMOVED methods) to see if its rank is greater than or less than the old SUPER. If the the rank of a new SUPER is less than the rank of the old SUPER, then the new place is inserted in the inclusion hierarchy between the current

frame and the old SUPER. In Fig. 7, we that if CAMBRIDGE was IN MASS, and it was asserted that CAMBRIDGE was also IN a new place METRO-BOSTON, which has a lower rank than MASS, then CAMBRIDGE would have MASS removed as its SUPER, and have it replaced by METRO-BOSTON. MASS would then gain the SUB METRO- BOSTON. This leaves CAMBRIDGE with just its most immediate SUPER, as desired. This process generalizes when a place has several SUPERs in a straight forward manner by comparing the new SUPER with the entire set of old SUPERs.

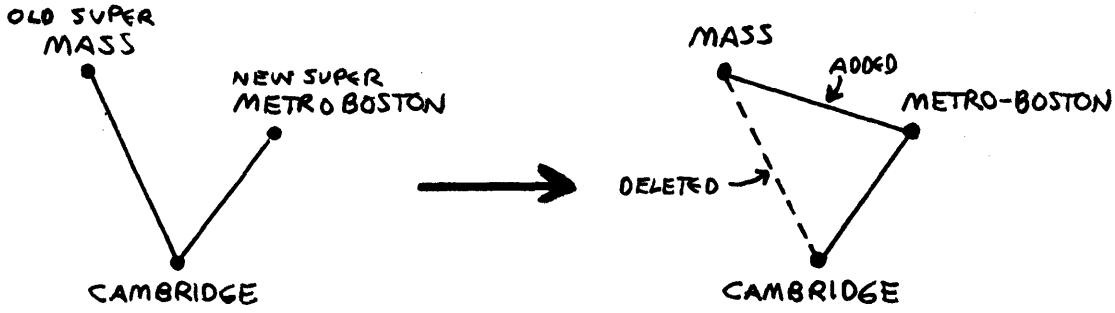


Fig. 7 Eliminating Hidden Inclusion Relationships

More formally, when a new SUPER Q is added to place P, an IF-ADDED method (CHECK-IMPROPER-SUPERS) compares the rank of the new SUPER with that of one of the old SUPERs P(i):

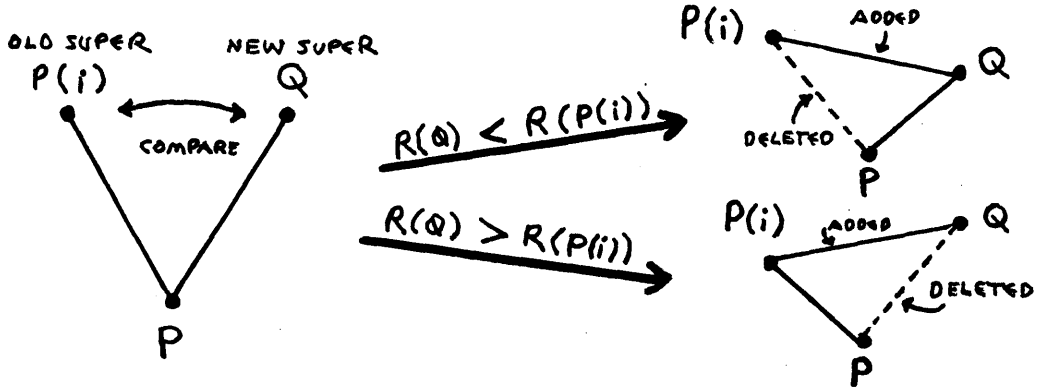


Fig. 8 The Formal Algorithm

If the rank of Q is less than the rank of some P(i) (which are all necessarily of the same rank), then Q remains a new SUPER of P, and the old P(i) are removed as P's SUPERs, and become Q's SUPERs. If Q is of greater rank than the P(i), then it is deleted as a SUPER of P, and added as a SUPER to each of the P(i). This last step can cause another use of CHECK-IMPROPER-SUPERS, since Q behaves as a new SUPER being added to each P(i). This process continues until Q can no longer rise in the hierarchy and has "percolated" to its proper position. There are a few subtleties to this process when places have the same rank (like NEW ENGLAND and the EAST-COAST), since overlap does not imply inclusion, but we will not pursue them.

The following is an example of how the percolation process described above takes place. Suppose that it is known that CAMBRIDGE is IN METRO-BOSTON which is IN MASS which is IN NORTH-AMERICA (see Fig. 9). Then suppose the new information is asserted that CAMBRIDGE is IN the UNITED-STATES. The rank of the UNITED-STATES is compared with the rank of CAMBRIDGE's old SUPER, METRO-BOSTON. Since it is higher, the UNITED-STATES is deleted as a SUPER of CAMBRIDGE, and becomes a SUPER of METRO-BOSTON. In becoming a SUPER of METRO-BOSTON, the UNITED-STATES triggers the process again. The UNITED-STATES, as a *new* SUPER of METRO-BOSTON, is then compared with MASS, which is an *old* SUPER of METRO-BOSTON. These comparisons continue until the UNITED-STATES is compared with NORTH-AMERICA. Since the rank of the UNITED-STATES is less than that of NORTH-AMERICA, NORTH-AMERICA is removed as MASS's SUPER, and becomes the UNITED-STATES' SUPER. This leaves each place with only its most immediate SUPER, guaranteeing there are no hidden

inclusion relationships. This process is summarized below:

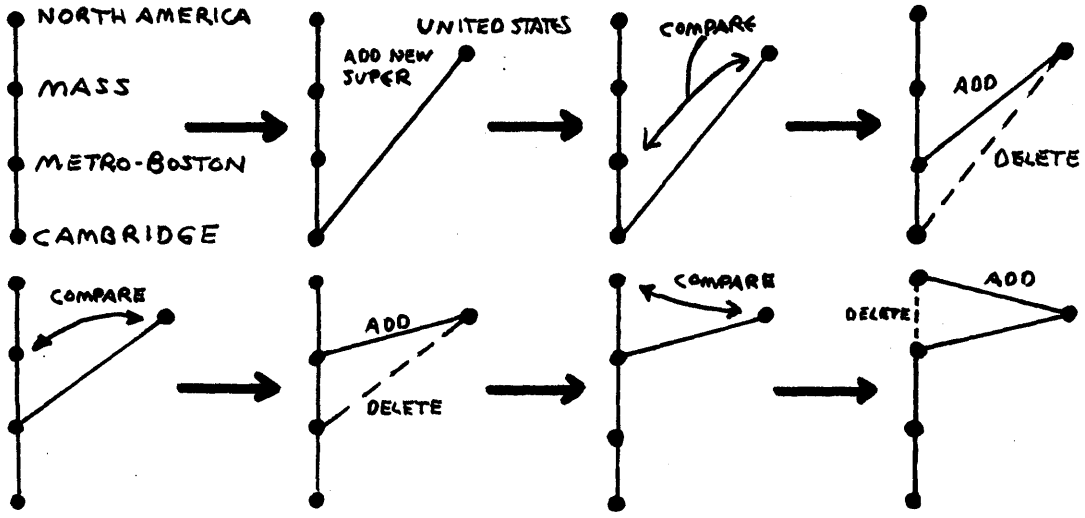


Fig. 9 Recursive Application of the Rank-inclusion Principle

While the use of attached procedures does allow new places to be automatically integrated into the place hierarchy, we see that it results in a fairly complex, recursive computation. In chapters IV and V, we'll find that this tendency becomes more pronounced as the system tries to cope with the imperfect information base probable in useable systems.

Place Functions

There are several types of procedures that operate on the place-frames data base. First, there are predicates which *test the validity* of statements about the data base. Second, there are functions which *return information* from the data base. Finally, there are commands which *alter information* in the place-frame data base. We'll briefly examine functions of each type.

Predicates:

(CONTAINS? 'USA 'MICH) Does the USA contain Michigan?

CONTAINS? is computed using inclusion relationships by checking to see if the first argument is a member of the transitive set of SUPERs of the second.

(NEAR? 'MASS '(BOSTON CAMBRIDGE)) Are Boston and Cambridge near to one another relative to Mass.?

NEAR? is computed by intersecting the transitive set of SUBs of the first argument with the smallest place(s) which contains all the places of the second argument (this is the PLACE-SPAN of the second argument -- see below). The result is then a place (or places) which is a SUB of the first argument, and which contains all the second arguments; this corresponds to our definition of NEAR. If this result is non-empty, the function returns true.

(PLACE-TYPE? 'BOSTON 'CITY) Is Boston an example of a city?

PLACE-TYPE? is computed by seeing if its second argument is either the AKO of the first argument, or is a member of the set of all the AKOs of all the SUPERs of the first argument.

Functions that interrogate the data base:

(TYPE-SPAN 'BOSTON) ==> CITY

TYPE-SPAN is computed by intersecting the rank levels with the transitive set of AKOs of the argument.

```
(PLACE-SPAN '(MAINE MIT)) ==> (SECTION NEW-ENGLAND)
```

PLACE-SPAN is computed by finding the place of lowest rank which contains all the places in the argument. This is done by intersecting the list of their SUPERs, and then finding the one that is of lowest rank.

```
(TRAVEL-COST '(BOSTON AUGUSTA))      = ((CAR 15) (PLANE 35))
(TRAVEL-TIME '(BOSTON AUGUSTA))      = ((CAR (HOUR 5))(PLANE (HOUR 2)))
(TRAVEL-MEANS '(BOSTON AUGUSTA))     = (CAR PLANE)
(MINIMIZE 'COST: '(BOSTON AUGUSTA)) = (CAR 15)
(MAXIMIZE 'TIME: '(BOSTON AUGUSTA)) = (CAR (HOUR 5))
```

These functions are all computed by looking at the INTRA-TRAVEL slot of the frames in the PLACE-SPAN of the places given. If, for example, the PLACE-SPAN is NEW-ENGLAND, then NEW-ENGLAND, via inheritance, will have the INTRA-TRAVEL slot value for a SECTION. This slot contains a list for each vehicle with order of magnitude estimates for typical travel times and costs within an average area of that type:

```
(SECTION        (AKO            ($VALUE    (PLACE)))
                (INTRA-TRAVEL   ($DEFAULT (CAR    (COST:  15 DOLLARS)
                                                  (TIME:   5 HOURS))
                                          (PLANE (COST:  35 DOLLARS)
                                                  (TIME:   2 HOURS))
                                          (BUS    (COST:  20 DOLLARS)
                                                  (TIME:   7 HOURS))))
                . . . )
```

A set of estimates for CITY occurs at the beginning of this chapter. More sophisticated estimates of travel time and cost are possible if exact sizes and/or routes are known. Inheritance also provides an effective way to supply information specific to a particular place, since inheritance will reach more specific information before general information. Hence special information about NEW-ENGLAND, such as the effects of a rail strike, would be inherited before general information about SECTIONs.

```
(DESCRIBE-PLACE 'MIT)    ==>  SCHOOL:        MIT
                              CITY:          CAMBRIDGE
                              METRO-AREA:    METRO-BOSTON
                              STATE:         MASS
                              COUNTRY:       USA
                              CONTINENT:     NORTH-AMERICA
                              WORLD:         EARTH
```

DESCRIBE-PLACE looks thru the transitive set of SUPERs of its argument, and prints them in order of increasing rank.

Commands that alter the place-frame data base:

SPECIFY-PLACE is a function which engages the user in a "menu" style dialog to define a new place. It uses the preference and exemplar facets and SUB-SUPER and AKO-INSTANCE information to generate reasonable choices and examples for the user (see chapter VI). Here's a sample dialog generated by invoking SPECIFY-PLACE:

```
What is the name of the place?

>Center-line

Center-line is AKO:

        (A) room        (D) state
        (B) building    (E) country
        (C) city        (F) other
>C

Center-line has super type:

        (A) metro-area          (such as Metro-Boston)
        (B) political-unit      (such as state Cal Mass)
>A

Can you be more specific?

>yes
```

```
Choose one or supply your own:

        (A) Metro-Boston
        (B) Metro-Detroit
>B

Is this correct for Center-line?

CITY:           CENTER-LINE
METRO-AREA:     METRO-DETROIT
STATE:          MICH
COUNTRY:        USA
CONTINENT:      NORTH-AMERICA
WORLD:          EARTH

>yes

(place Center-line located)
```

This information is computed by the DESCRIBE-PLACE function, above.

Everything in Its Place

We have seen that all the elements of our basic theory of place can be effectively expressed in FRL, a hierarchy-based knowledge representation language, thereby illustrating the usefulness of inheritance for both data and procedures. Inclusion relationships are expressed as SUB-SUPER pointers between place-frames. The restrictions of rank ordering are embedded as inherited requirements and preferences on SUB-SUPER links. Procedural attachment through IF-ADDED and IF-REMOVED methods on the SUB and SUPER slots maintains the logical integrity of the place hierarchy, bringing about the changes that must be made in the SUB-SUPER relations when a place is added or removed. Finally, SUB, SUPER, and AKO chains are used to implement the concept NEAR, and to establish basic travel information. The next chapters explore some of the difficulties encountered when the hierarchical structure forced to cope with less rigorous information.

IV. Refining the Theory of Place

The theory of place presented so far depends on a hierarchy of inclusion relations and place types. We have seen that there are exceptions to these relations, such as Kansas City, and that a comprehensive theory of near is needed for situations like Detroit and Windsor. This chapter develops extensions to handle the limitations of a strictly hierarchical representation. We will find that these extensions often depend on information that goes beyond the general, widely known information of name, rank, and inclusion that our simple theory uses. Hence there is a trade-off between the accuracy provided by the extensions, and the availability of the facts they depend upon. We shall also find that these extensions raise general issues whose solutions require new processing capabilities, such as greater control of inheritance mechanisms, reasoning with *sets* of values, converting between multiple representations, and reasoning with uncertain values.

Exceptions to Inclusion

A fundamental property of the inclusion principle is its transitivity -- if A is IN B, and B is IN C, then A is IN C. This property hinges on the fact that IN means completely in. What about those places which are only partly IN other places? Kansas City is in two different states; Berlin is divided between East Germany and West Germany; the Rocky Mountains cut through portions of many states. One of the most important implications of a PARTLY-IN relationship is that the rank-inclusion principle no longer holds. Hence, if I am IN the Rocky Mountains, and they are only PARTLY-IN Wyoming, then there's no way to tell if I'm IN Wyoming or not.

It is clear that it is best to use the "completely in" relationship when it *is*

applicable, because of the reasoning power it provides. When it doesn't hold, it seems that that some provision must be made for expressing and reasoning with PARTLY-IN, since it can't be avoided. What constraints are there for the use of PARTLY-IN?

First, a PARTLY-IN occurring anywhere in a chain of inclusions breaks the chain from that point downward. Hence if A is IN B, and B is PARTLY-IN C, and C is IN D, nothing can be deduced about the relationship between place A and the places C and D; Fig. 10 demonstrates several possibilities. However, a PARTLY-IN relationship holds for B and D in every case, since the IN relation does preserve the PARTLY-IN relation.

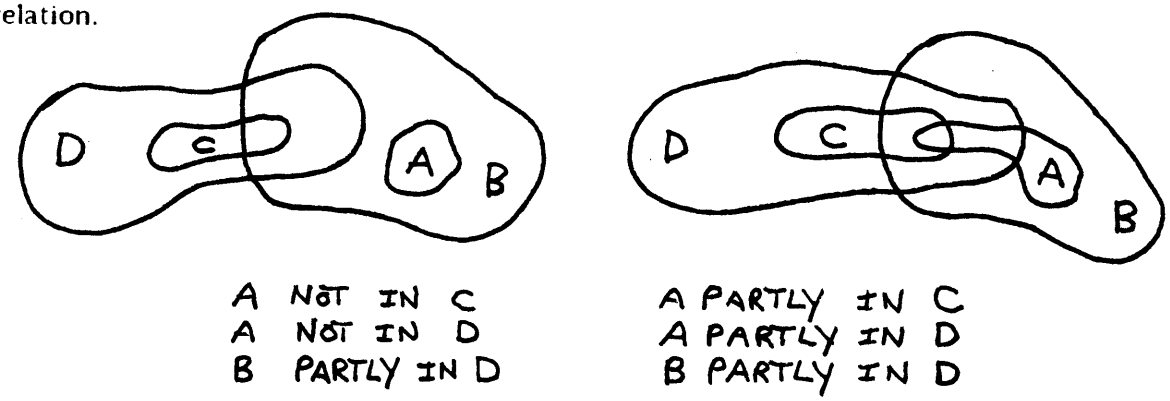


Fig. 10 Possible Inclusion Relations with PARTLY-IN

For example, if my boat is IN the Rio Grande, and the Rio Grande is PARTLY-IN Texas, and Texas is IN the United States, then I don't know whether my boat is IN the United States, PARTLY-IN in the United States, or completely outside it. I do know, however, that the Rio Grande is PARTLY-IN the United States, since the IN relation preserves the PARTLY-IN relation.

The PARTLY-IN relation can reduce to a "completely in" of the union of the places involved. For example, if Joe's Place is IN Kansas City, then it can be deduced that it is IN the United States, although not directly from inclusion. Kansas City is

PARTLY-IN Kansas, and PARTLY-IN Missouri, but it is *completely in* the two taken together. The use of exhaustive sets of PARTLY-IN'S allows the same deductions that can be made about individual places to apply to the entire set. Hence since Joe's is IN {Kansas, Missouri}, and {Kansas, Missouri} is IN the United States, we know that Joe's is IN the United States.

Reasoning with PARTLY-IN can become quite subtle. For example, this same deduction concerning Joe's Place could have been done another way, utilizing adjacency and size information. Since Kansas is far from the United States border, and Joe's Place is much smaller than Kansas, the fact that it is PARTLY-IN Kansas guarantees that it is IN the United States. We will not consider this degree of subtlety in detail.

We have seen that a PARTLY-IN relationship can be used to represent some of the exceptional cases mentioned previously, as well as the more irregular inclusion relations of natural entities such as rivers and mountains. We also have explored some of the machinery needed to utilize this extension to the representation, such as reasoning with *sets* of place. Finally, we note that the increased scope of the representation has a concommitant cost in terms of the complexity of the reasoning which must be done to process the extended representation.

Exceptions of Rank

Rank is our term for the categorization of places according to various levels of generalization (such as room, building, city, country, and continent). One desirable property is that each kind of place, such as university, restaurant, laboratory, stadium, etc., should fit into exactly one of these categories, thereby inheriting useful, unambiguous

information, such as approximate size, means of travel, kind of activity, etc. This uniqueness property seems to break down as far as man-made structures are concerned -- a restaurant, for example, does not fall into exactly one category; it may be a room, a collection of rooms, or an entire building. This can result in conflicting inheritance chains: a building may have a furnace or stairwells, while a room typically has neither. Leaving the situation ambiguous also has drawbacks; reasoning is difficult when even these broad characteristics are in doubt. For example, "I will meet you at Joe's Place" is not sufficient to specify a place to meet, if Joe's is a collection of buildings; it *is* sufficient if Joe's is a single room. This failure of places to fall into a single category also makes monitoring for violations of rank inclusion more difficult: a collections of rooms can be IN a building, but not vice versa, but there is no way to test for a violation if one of the places involved could be *either* a collection of rooms *or* a building. Finally, nothing has been said about how natural places like mountains and lakes fit into this ranking scheme. How can these difficulties be dealt with?

Conflicting Properties

One approach to conflicting inheritance is to find a common abstraction, i.e. to set up internal categories which contain only the common properties of a given set of categories. One such category might be that of STRUCTURES, which would have all the properties held in common by a room, collections of rooms, a building, and a collection of buildings. Then there would be no difficulty with not knowing which of the four types of places a place actually is -- only properties common to all four types would be inherited.

This approach has weaknesses. Besides the fact that the number of such

properties is small, simple heuristics in a normal situation often narrow down the possibilities to some subset of these four categories. Being able to eliminate a category is often very helpful, since it allows stronger constraints to be put on the properties. For example, if something was known to be smaller than a collection of buildings, maximum travel time is significantly reduced, and certain factors, like weather, are no longer very important. But if eliminating categories is useful, 15 possible groupings of categories would be needed in this case to allow for all possible eliminations, including ROOM-OR-ROOMS, ROOM-OR-BUILDING, ROOM-OR-ROOMS-OR-BUILDING, etc. Hence we find this approach rapidly becomes inelegant and ponderous. This large number of artificial categories would be confusing to someone trying to classify a place. Finally, there is no way to compare ranks of these sub-categories -- the rank of a ROOM-OR-ROOMS can not be compared with the rank of a ROOMS-OR-BUILDING.

Another approach is to group categories in a more natural way -- as a unit and the corresponding collection of units -- such as room and rooms, or building and buildings. While these groupings combine categories of similar properties, and can be nicely described as singleton-collection relationships, they do not provide sufficient flexibility to handle the property conflict in the ROOMS-OR-BUILDING case. This singleton-collection relationship does seem to be a very common one, though, since city-metropolis, state-section, and country-federation all reflect it. A more refined theory of place should make it possible to define and use such a relationship.

Rank ambiguities could also be handled by a general facility for dealing with logical operations on inheritance. For example, a restaurant would be of rank (OR ROOM ROOMS BUILDING BUILDINGS). However, schemes as simple as ORing

don't seem sufficient by themselves. For example, reasoning with rank disjunctions becomes an exponentially explosive problem, since at each decision point, each option would has to be taken into consideration. A similar difficulty exists with the simple use of ANDs. If a restaurant were the AND of the four categories, conflicting inheritance chains can result. There is also a problem of consistency -- a place shouldn't inherit a property from ROOM one time, and BUILDING the next. A more sophisticated logic scheme which takes into account ambiguity, conflict, and exceptions in the inheritance chain is needed.

Throughout the above discussion, we have seen a clash between the uncertainty of available knowledge, and the desire for classification into exclusive categories, which makes powerful reasoning possible. It may be the case that these limitations are intrinsic to powerful classifications which must deal with uncertainty, and hence are not the result of bad classification. Fahlman (1977) takes such a perspective in that he develops a theory of hierarchical representation which directly addresses itself to issues of exclusiveness, conflict, exceptions, etc. The theory that he presents may in fact be adequate for the PLACE system. Unfortunately, his implementation depends on a parallel processing scheme which is not generally available. Rather than duplicate Fahlman's work, the approach taken in extending the PLACE system has been to delay direct confrontation of this issue until Fahlman's scheme or a similar one is available. When it is, the PLACE system would be a ready candidate for testing it.

In the mean time, a far simpler solution to these difficulties will be used. Where there is uncertainty, a choice will first be made as to which rank is most likely, and then the regularities of the place theory will be applied to that *single* rank. Since the

theory is being applied as before, little modification to the current implementation is required. This approach should prove adequate until more powerful hierarchically-based schemes are available, and it deserves study as a significant style of reasoning in its own right.

Places Outside the Ranking Scheme

Another short-coming of rank as discussed so far is that it doesn't include certain large classes of places -- such as playgrounds, ball fields, lakes, rivers, mountains, mountain ranges, harbors, valleys, etc. Furthermore, these kinds of places don't exhibit the same kinds of regularities as the types of places already mentioned, such as buildings, cities, and countries. For example, size variations are immense; travel means can vary a great deal for items of the same type; and the rank-inclusion property doesn't often hold, since these areas *are* split by boundaries. This last issue has the greatest impact on the current theory, since it means that a PARTLY-IN relationship is required if these places are to be merged into the existing hierarchy.

Some of these types could be accommodated into the current theory quite simply, such as playgrounds and ball fields. This can be done by creating a new category, say OUTSIDE-PLAY-AREAS, and giving it a rank intermediate to those of building and complex. The other areas, besides requiring a PARTLY-IN relationship, don't have inclusion or relative size relationships to the other established categories, and hence are not constrained by their relationships to established ranks. This substantially limits the amount of reasoning that can be done. Perhaps it is inevitable that for any large knowledge base, there will be significant portions where generally useful reasoning heuristics don't apply.

Another significant issue is that while places outside the current ranking scheme don't follow strict inclusion very well, they are sometimes spoken of in terms of *dominant* inclusion. For example, one might say "The Rocky Mountains are in the western United States," even if they do extend into Canada slightly. Hence "in" doesn't means "completely in", or "partly-in", but "mostly-in". When the system's normal reasoning heuristics are applied to these situations where dominant inclusion holds, the conclusions arrived at are *often* correct, but not *always* correct. Hence some care must be taken in their use. No modifications are required for the system to perform this reasoning as long as the dominant including area is a single entity. If the including area is a set of entities, then an extension to the IN relationship to handle sets is needed, just as with the PARTLY-IN relationship. Again we see that the added complexity of handling these exceptions gives the system completeness, but not much more reasoning power.

Another way in which these types of places are spoken of is as borders. This is the case that the present theory least addresses. We'll consider it more under the concept of adjacency.

Refining Near: Directionality, Adjacency, and Size

The hierarchical concept of near that has been examined so far is too narrow to accommodate the finer distinctions that are often made in everyday circumstances. We will examine some of the those concepts which can be used to provide a more refined near. The following examples serve to illustrate the various concepts and their use.

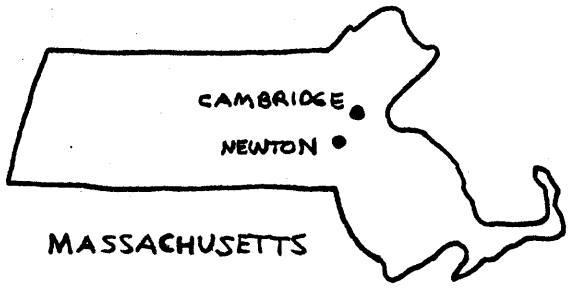


Fig. 11 Directional Information

Newton and Cambridge are considered near to each other relative to Massachusetts because it is known that each is in Eastern Massachusetts.

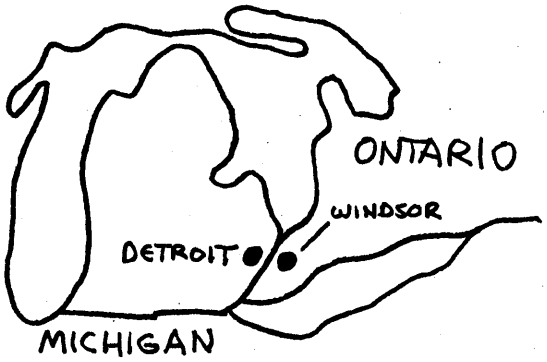


Fig. 12 Adjacency and Directional Information

Detroit and Windsor are considered near because Southeast Michigan is known to be adjacent to Southwest Ontario, and because Detroit and Windsor are known to be in those sub-parts.

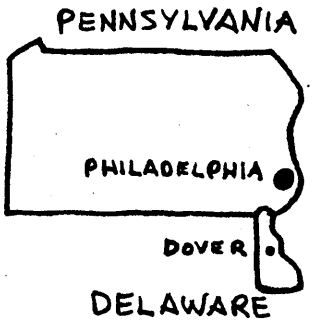
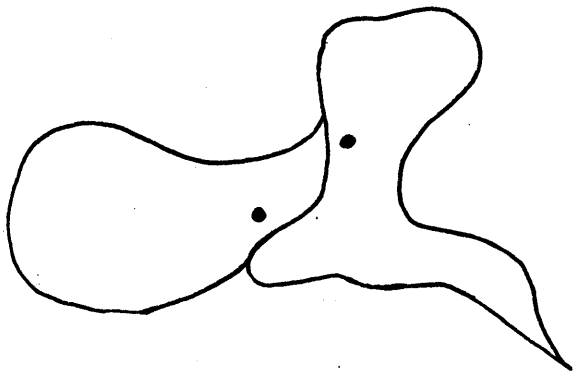


Fig. 13 Adjacency, Size, and Directional Information

Philadelphia is near Dover relative to Pennsylvania because Delaware is known to be adjacent to Southeastern Pennsylvania, and is very small; hence any place in Delaware is near any place known to be in Southeastern Pennsylvania, such as Philadelphia.



Various other factors can be used in determining nearness. Here, knowledge about the detailed geometry of the places is needed to determine nearness.

Fig. 14 Adjacency, Direction, Shape, Orientation, and Position

Directionality

Directional modification is probably the most common means of modifying a place specification (no matter what the rank): western Massachusetts, northside of Detroit, etc. The incorporation of directional information is quite simple -- a directionally modified place can be thought of as defining a pseudo-place, with its own name, rank, and inclusion relationship. More specifically, the name would be <direction>-<unmodified place name>; the rank would be just below that of the unmodified rank; and the inclusion relationship IN would hold with the unmodified place. For example, Newton and Cambridge would be near each other relative to Massachusetts, because they are both in the pseudo-place Eastern-Massachusetts, which is IN Massachusetts, and thus fulfills the definition of near. (The use of directional information *across place boundaries* is a more difficult question which we'll examine later.) There is some caution required in dealing with directionally modified places, though, since a place can be in two pseudo-places at the same time -- inclusion is not exclusive with directionally modified places. For example, a city in Northwestern Massachusetts would be in both Western

Massachusetts and Northern Massachusetts. Though troublesome, this overlap does not require any modification to our current theory, since the rank-inclusion principle only says that places of *differing* rank can't overlap.

A weakness of the use of directional information is that potentially misleading inferences are possible: while normally the rank of a place is known, the appropriate directional modifier may not be. For example, places P and P' may be known to be IN western Massachusetts, and place Q may be known only to be IN Massachusetts. Therefore, relative to Massachusetts, place P will be near to place P', since they are both in the sub-region Western Massachusetts. However, Q will not be near, even though it may actually be much closer to P' than P, since it is not known to be IN any sub-region of Massachusetts. Hence deducing from the fact that P is not known to be near Q that P is, in fact, not near Q is invalid, unless both have the relevant directional modifiers present. The ability to deal with uncertainty is required, since the correct answer should be: P and possibly Q are near P'.

Another difficulty with directional information is that it follows an approximate, but not exact, relation to inclusion. If city P is south of city Q, there is a good chance that the country of city P is south of the country of city Q. But this is not necessarily so, since countries can have irregular borders:

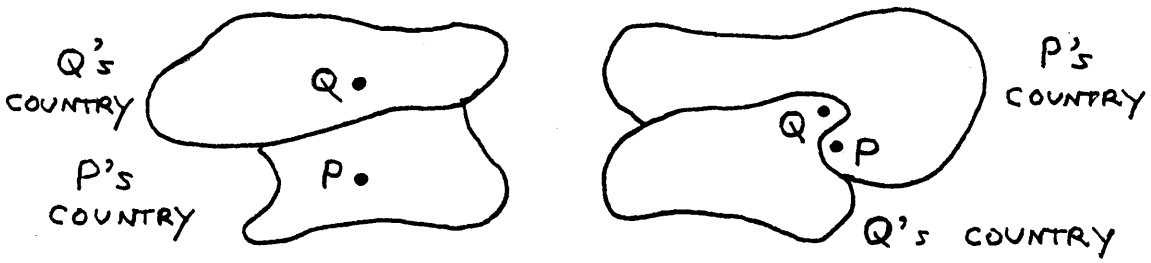


Fig. 15 Directional Information and Inclusion

Sometimes special information, like the regular square shape of most rooms or the regular arrangement of buildings along a street can be used to strengthen this type of reasoning.

Also, seeming contradictions can result from the use of "dominant" direction. For example, country P shares its eastern border with country Q, but Q is basically southeast of P:

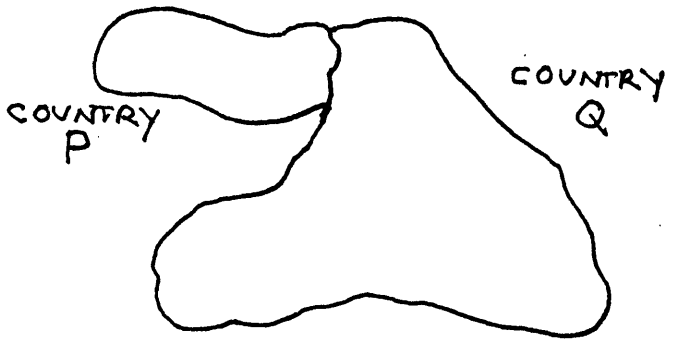


Fig. 16 An Exception To Dominant Directionality

No attempt will be made to accommodate these peculiarities, but only to recognize that directionality deduced through inclusion and dominant directionality are weaker regularities than the others we have been exploring, and that there are many subtleties involved in their use. Perhaps this is where a geometric representation becomes useful -- the force of a picture being worth a thousand words begins to be felt more strongly as detailed knowledge increases.

Finally, there is a danger of not using the "appropriate" directional modifiers -- the ones that give useful distinctions. For example, the indicators southern P and northern P tell very little in regard to place P; the "appropriate" modifiers are eastern, central, and western (Fig. 17). For other places such as place Q, southwest and northeast would be better (Fig. 18). This idea of appropriate modifiers is really a combination of orientation, shape, and position information.

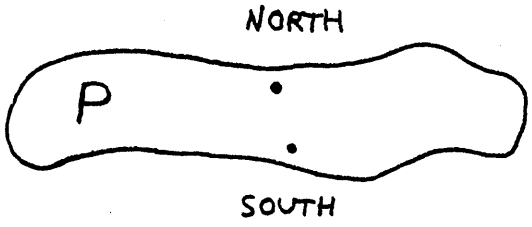


Fig. 17 Inappropriate Directional Modifiers

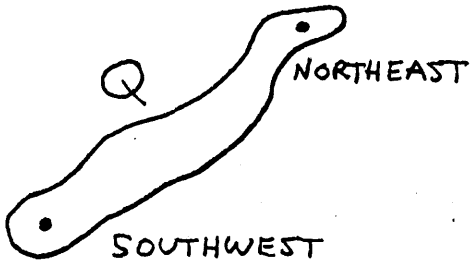


Fig. 18 Appropriate Directional Modifiers

Adjacency

The greatest weakness of the theory of place presented so far is that it cannot handle the concept of adjacency. As we have seen, this is particularly important in relationships like near when the crossing of place boundaries is involved. For example, as was mentioned, Windsor, Canada is near Detroit, Michigan, but there is no direct way to express that without additional refinements of near. The refinement suggested here is that whenever adjacent areas exist, checks utilizing directionality and size should be made. As we have seen, modifiers of Southeast for Michigan and Southwest for Ontario would indicate the nearness of Detroit and Windsor (Fig. 12); Eastern Pennsylvania and the size of Delaware would indicate the nearness of Philadelphia and Dover, relative to Pennsylvania (Fig. 13).

This relationship cannot be expressed by rank and inclusion relationships. However, rank and inclusion are both useful in *discovering* adjacency. For example, suppose the cities of Detroit and Windsor are known to be adjacent. A question as to whether Michigan and Ontario are adjacent could be answered by checking to see if any place IN Michigan is known to be adjacent to any place IN Canada; eventually Detroit

and Windsor would be found to be adjacent, and hence so would Michigan and Ontario. As the ranks of the places known to be adjacent and those being asked about grow further apart, the computational effort can go up significantly. If asked to see if the United States was adjacent to Canada, a naive approach could involve checking every place IN the United States to see if it was adjacent to any place IN Canada. Note, too, that we have been using adjacency of two places to show the adjacency of their superiors; using nearness to suggest adjacency of superiors is possible, but not always valid. For example, if two houses in different countries are near each other relative to the size of, say a city, then one might guess that their respective countries are adjacent. However, it is conceivable that another country could lie between them:

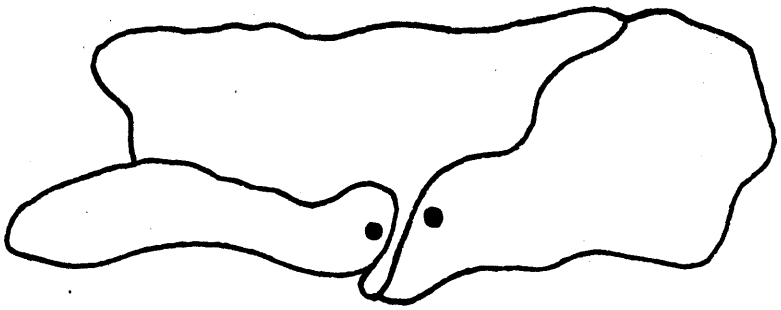


Fig. 19 Using Nearness to Suggest Adjacency

Rank information can be used to simplify adjacency calculations if each place has associated with it all the places that are adjacent to it *of the same rank.* Questions like "Is the United States adjacent to Canada" are easy then. This can be done by percolating adjacency information from places of low rank to places of higher rank. For example, if Detroit is known to be adjacent to Windsor, then each successive rank generalization also holds (Michigan adjacent to Ontario, United States adjacent to Canada), and Detroit can pass its adjacency information to its SUPERs who will in turn pass it on to their SUPERs. There are still many other cases that could be explored, but

we'll move on.

This added power provided by adjacency information carries with it the same difficulties that extension to directionality did -- there is dependence on increasingly less available information (did *you* know Delaware and Pennsylvania touched?). The absence of such information is not necessarily evidence for its being false, so care is needed in interpreting it. When adjacency information is not available for a detailed consideration of near, the looser rank-inclusion definition serves as a less exact, but more often applicable, default.

Relative Size

As we have seen, size information can be useful in determining nearness; it can also be used to guess travel times, seating capacity, etc. While rank provides a rough approximation of size, actual size can vary considerably. For example, consider Texas vs. Rhode Island, or the Soviet Union vs. Japan. Size, as well as height and depth, become even more important when dealing with rivers, mountains, mountain ranges, etc. Relative size is also important when dealing with *types* of places; for example, it may be known that a state is only a medium-sized political unit, while shire is a very small one. Massachusetts is a small state, which is in turn a medium-sized political unit, so a relative size weighting is needed when applying information about political units to Massachusetts. While absolute size information (such as number of square feet or square miles) would be the ideal canonical measure, again it must be remembered that the more detailed a piece of information is, the less likely it is to be available.

How does this size information relate to rank and inclusion? First, size is

generally *relative*, and hence a context is necessary in order to understand what is meant. The rank of a place provides some of that context -- any reference to the size of a place can be taken in the context of its rank. Hence, when one says Japan is small, it is relative to its being a country. Specialization of rank, such as state as a specialization of political unit, could provide a further refinement, with the immediate specialization serving as the context. Then a small state could be compared somewhat with, say, a large shire.

A further refinement in the use of size could involve relative comparisons between places of *different* rank. This is more difficult, since the weighting of sizes down through the various specializations of a rank is not very exact. While it might be easy to believe that an auditorium (which is a very large room, which is a kind of meeting place) is smaller than even the smallest continent, the comparison of a very large city with a small country is much more difficult. It's close enough so that one typically compares the suspected sizes of known *examples* of large cities, say New York or Chicago, with the suspected sizes of known *examples* of small countries, say Switzerland or Japan. Hence we say that size relationships between ranks can be examined via examples.

Other Refinements

We have explored some of the major factors that one might need to refine the theory of place to handle more information about nearness. Other factors exist which reflect geometric information about a place more explicitly, including its shape (oblong, round, pear), the orientation of that shape (north-south, big end southwest), and the actual position of sub-places (3 miles from the southern border, and 20 from the eastern border). The more geometric this information is, the less it is amenable to the symbolic approach being employed here. It may be true that any theory which is to handle all

possible place information will have to be able to convert symbolic information to numeric information, and numeric information to symbolic. One could ask why can't all these symbolic expressions be turned into ranges in numeric terms. One reason is that purely relative comparisons can't be adequately expressed -- there is no way to indicate country P is 10 square miles bigger than the country Q, if the uncertainty of the absolute sizes of P and Q is large relative to 10 square miles. Secondly, the use of numerical ranges tends to obscure the effects of inexact reasoning -- the uncertainty in the amount of uncertainty is often large or unknown, and numerical interpretations of it seem presumptuous.

There are many other types of place information that a more sophisticated theory might include, such as: applicable laws (passport, customs), currency exchange (rubles, dollars, pesos), accessibility (club membership, age limits), population statistics, and dominant weather conditions. We won't consider any of these.

New Issues Raised

This exploration of extensions to the basic theory of place indicates several new issues. First, the issue has been raised whether the difficulties in handling *exceptions* in the place classification is a general problem of classification schemes, or a result of a poor choice of classes. It has been argued that it is a general problem of dealing with uncertainty in exclusive classification schemes.

Second, it has been suggested that being able to deal with *sets* of places as easily as with individual places would be very useful -- particularly when properties such as exhaustiveness and exclusiveness are known. This raises a more general question:

perhaps this is a capability that should be built into any general scheme which deals with part-whole relationships.

Third, we have seen that symbolic representation becomes strained when more detailed geometric knowledge becomes available. The converse also appears to be true -- geometric representation becomes difficult when information is sparse and uncertain. Does this point to an intertwined dual representation which can handle both extremes? Further work is needed.

Fourth, it is evident that refinements of place concepts such as near depend on more detailed information. The more detailed a piece of information is, the less likely it is to be available. Also, added detail usually means added computational complexity. Together, these factors suggest that a reasoning system should have provision for making trade-offs among the specificity of the desired answer, the availability of the required information, and the cost involved in making the computation. When a quick general answer is needed, or when little is known with much certainty, a broad theory can be used, such as the inclusion-based definition of near. More effort to acquire information and more computational effort can be used as more exact answers are required. This use of progressively more detailed theories as a way of limiting cost and dealing with uncertainty seems promising.

Finally, we have seen that secondary regularities (which are often, but not always valid) and the uncertainty of information necessitate a reasoning system which can accommodate varying states of certainty. The following table represents a coarse-grained approach in that direction:

T = Valid reasoning, proving truth.
T? = Positive, though not conclusive, evidence
? = No evidence found.
F? = Negative, but inconclusive, evidence.
F = Proven false.
A = Ambiguous inconclusive evidence (i.e., both T? and F?)
C = Contradictory evidence found (i.e., both T and F)

Of course, the interaction of certainty of information with certainty of reasoning is quite complex; the suggestion here is that it appears necessary that such inexactness must be handled by any theory that hopes to be generally applicable.

We have seen in this chapter that extending the theory of place to handle exceptional cases and more detailed descriptions brings up many general issues of representation. Hence we find that while a hierarchical organization of knowledge proves effective as a first order theory of representation, it is ineffective by itself for handling refinements. It seems that future work should focus on integrating the organizational power of hierarchy with more powerful processing and reasoning capabilities, such as controlled inheritance, set manipulation, dual representation schemes, cost-validity considerations, and multi-valued logic.

V. Refining the Frame Implementation of Place

We have already examined some of the more theoretical limitations of our hierarchical scheme. In this chapter, we explore the limitations of the implementation of that scheme, focusing on difficulties encountered while trying to maintain the hierarchy in the face of incomplete and changing information. We shall find that incomplete information complicates the use of local attached procedures to add information, resulting in recursive computations which make it difficult to understand and verify the resulting reconfiguration of the hierarchy. This suggests that a cleaner, clearer approach is needed which has a somewhat broader perspective than just the immediately connected frames. We shall also see that deleting information raises difficult questions about data restoration that the PLACE system can not handle, indicating that perhaps more processing power is needed.

Ignorance Is Not Bliss

The first issue we will consider is that of incomplete information. Data bases with hierarchical structure depend on having available certain information to create that hierarchy. When that information is not available or is only partially available, provision must be made for keeping track of data until it can be properly integrated into the data base, and for correctly propagating the side-effects of the addition.

In the context of the place-frames system, this problem arises in maintaining the rank and inclusion disciplines of the data base. For example, suppose the SUPER of a place is known, but its rank is not. How is such a place to be included in the data base? How can rank information be used to properly locate that place in the place hierarchy

when that information does become available? How should the functions which depend on knowing rank handle this case? Exploring these questions will help us understand the difficulties that incomplete information presents for hierarchical data bases.

Whenever a place is being specified, the user can choose not to respond to the question being asked, whether it relates to determining rank, inclusion information, or individuality. One approach to handling this lack of information would be to keep incompletely specified frames in a separate data base, and generalize the functions which access and manipulate the frames system to utilize this separate data base. This approach seems very difficult in that the separate data base would have to be searched often, and the general system functions would have to become quite complex to handle the lack of uniformity. Furthermore, in a dynamic knowledge-based system, all frames may be to some extent incomplete. Hence I have pursued an integrated approach where a frame that is missing information is treated the same as the other place-frames, and is incorporated into the hierarchy as well as the missing knowledge permits. We will now look at how such a frame can be better incorporated when that missing knowledge becomes known.

A typical situation would be where the user asks "Is MIT in MASS?", and receives an answer of "Don't know". Then the user supplies the additional information that MASS is a state, and that is sufficient for the system to determine that MIT *is* in MASS. How could such a circumstance come about?

Consider the place-frame MASS which has a SUB link to NE43-348 (my office), and a SUPER link to USA, but no AKO value, so that its rank of state is unknown (Fig. 20). (We adopt the convention that inclusion links are represented by lines, with SUPER

chains generally heading up the page, and that places are represented by small circles. An *unfilled* circle means a place's AKO is unknown, and a *filled* circle means the AKO is known.)

Fig. 20

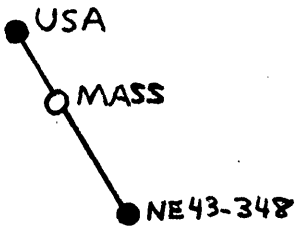
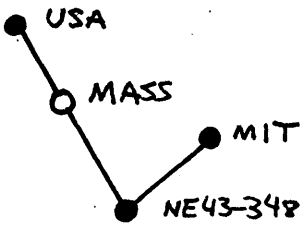
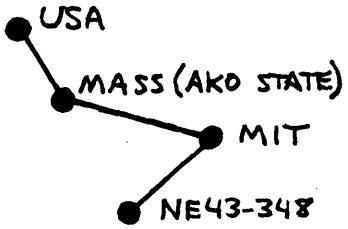


Fig. 21



Now suppose another known frame MIT is a SUPER of MASS's SUB (Fig. 21). Now, as suggested in the example above, suppose that MASS is AKO state becomes known. It is now possible to deduce via the rank-inclusion principle that the two branches of the "V" should be merged:

Fig. 22



There are two general courses of action possible as to how the merge could be discovered and carried out. One involves a global view of the system which looks over the entire data base and decides what changes to make; the other involves a local view of just the frame being changed. In order to investigate the power of if-added and if-removed methods, which are activated by the specific data being added, the "local view" approach was chosen. This puts the power of procedural attachment of frame-based systems up against the complexities of maintaining hierarchical discipline. A key advantage of the local approach is that the complexity of the computation remains independent of the total size of the data base. I shall present one such local algorithm, and then discuss its limitations and extensions (the basic algorithm is implemented in the

PLACE system, but its extensions are not). In the discussion which follows, we will assume that the initial system contains no hidden inclusion relationships; of course it may contain other places with unknown AKOs.

The insight behind the algorithm which we will consider is this: when a place's AKO becomes known, it will be treated as though it were just being added to the system for the first time. Recall from chapter III that when a new SUPER Q is added to a place P, the rank of that new SUPER is compared with the ranks of P's other SUPERs. The end result is that Q "percolates" to its proper place in the hierarchy. If we change our perspective so that we view the world from the *new* place Q, the algorithm for adding new places becomes "Check the SUPERs of the SUBs of the new frame". The algorithm does exactly the same thing as before, since the SUPERs of the SUBs of Q are precisely the SUPERs of P (which include Q). Hence when a frame's AKO becomes known, we apply the algorithm "Compare the SUPERs of the SUBs of this frame". The added AKO knowledge will then be used to percolate that frame to its proper place in the hierarchy:

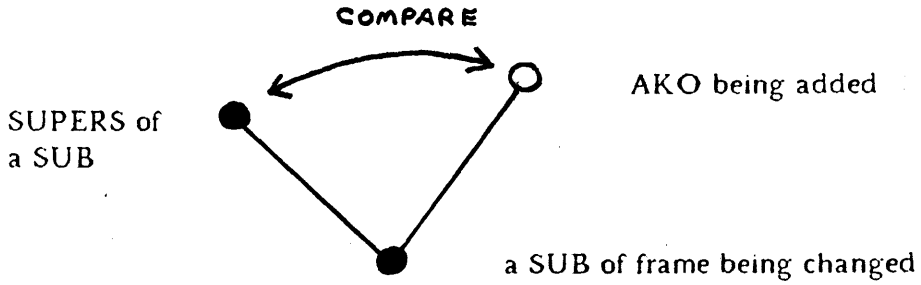


Fig. 23 The Initial Merging Algorithm

Let us look at this alogrithm in operation. The most common situation is where a place missing an AKO straddles several other places of known AKO (Fig. 24).

There's no way to tell whether METRO-BOSTON should be merged under MASS, CAMBRIDGE, or MIT. METRO-BOSTON's SUB is just NE43-348, an office, which has SUPERs MIT and METRO-BOSTON. Hence when METRO-BOSTON's AKO becomes known, applying our "Check the SUPERs of the SUBs" algorithm involves comparing MIT's rank with the newly learned rank of METRO-BOSTON. METRO-BOSTON will percolate to its proper place in the hierarchy (Fig. 25) just as a new place would.

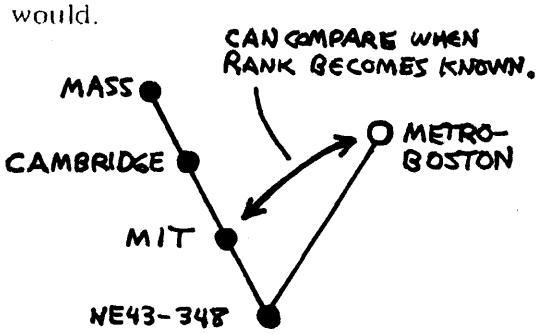


Fig. 24 A Missing AKO

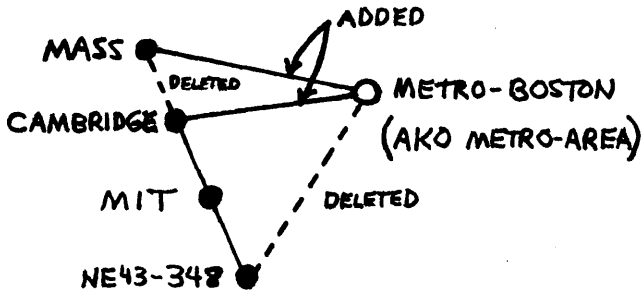


Fig. 25 The Merge is Completed

The difficulty with this algorithm is that there may be other places in the hierarchy with unknown AKOs. If this is the case, it may be that in the course of comparing ranks, a dead end will be reached, because there will be no ranks to compare. For example, if we apply our "Check the SUPERs of the SUBs" algorithm to Fig. 26, we will reach such a dead end very quicky, since MASS's rank cannot be compared with the unknown rank of CAMBRIDGE. Hence the proper merge (Fig. 27) will not be made:

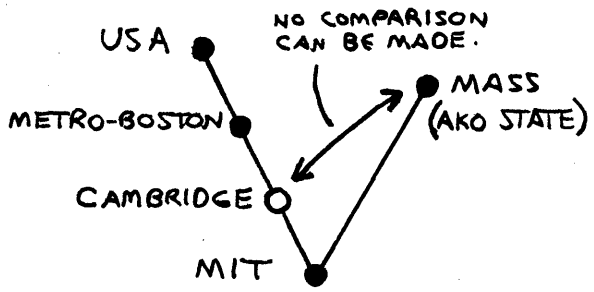


Fig. 26 Multiple Missing AKOs

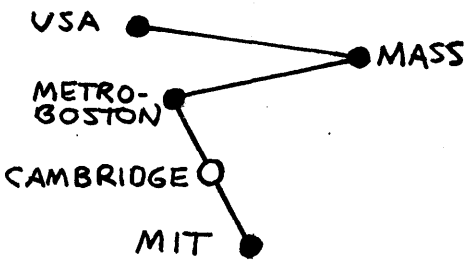


Fig. 27 The Desired Merge

One approach to handling the dead ends caused by these unknown AKOs is to pass through them when following SUB-SUPER links until a place which does have a rank is reached. A place which is a SUPER of this ranked place will be a proper SUPER of all its unranked SUBs as well. We can see how this would work in the last example -- if we "pass through" CAMBRIDGE, we can compare MASS with METRO-BOSTON and USA, and the proper merge results.

A slightly different example (Fig. 28) demonstrates some subtlety with "passing through"; when we pass through CAMBRIDGE and METRO-BOSTON, we reach USA, which is already a part of MASS's SUPER chain. So even though it has a rank, no merge can be made. What should happen next? In this same example, a merge *is* possible (Fig. 29), since MASS should be a SUPER of BOSTON. Hence we can see that the algorithm must not stop when already-merged SUPERs are encountered. In fact, we can see from this example that the proper comparison results if we "pass through" MIT, and check the SUPERs of the next SUB down, NE43-348 (my office):

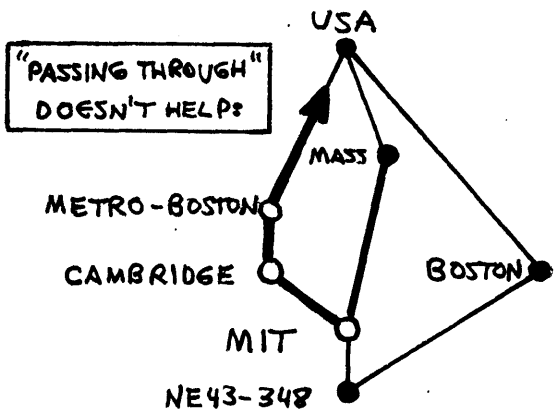


Fig. 28 Subtleties of "Passing Through"

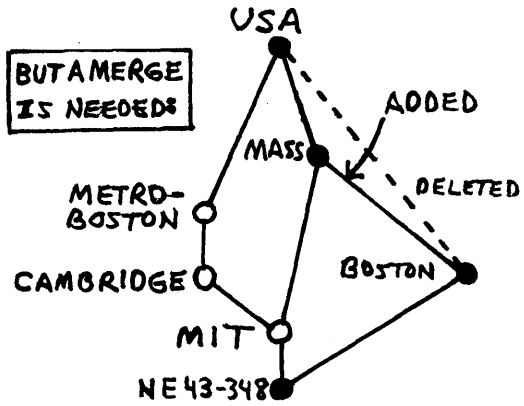
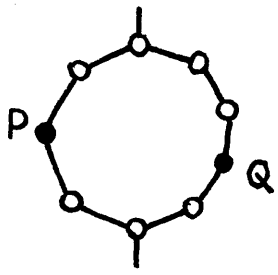
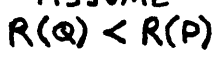


Fig. 29 The Desired Merge

When is it safe, then, to terminate the algorithm? Is everything properly merged when one merge has been made? Could there be any more possible merges with SUPERs of SUBs further down? An additional insight is required to answer these questions. We already know that the rank-inclusion principle is only applicable where places have a common SUB. We also know that comparisons require both places involved to have a rank. Figure 30 is drawn according to these two criteria -- a common SUB, and two unmerged SUPER chains with a least one ranked place in each chain. In Figure 31, we see that a merge is always possible if the two ranked places have different ranks:





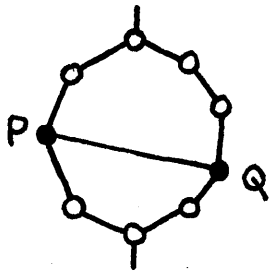


Fig. 30 Conditions Necessary for a Merge

Fig. 31 Also Sufficient for a Merge

Hence we have what appear to be necessary and sufficient conditions for when a merge is required: a common SUB with disjoint SUPER chains with a least one ranked place in each chain.

These conditions help us tell when to terminate our algorithm, since they indicate only one SUPER chain with unmerged ranked frames is possible. We can see this as follows: When a new AKO is added to the data base, it has added a ranked SUPER to all of its SUBs. Hence any SUB could now potentially be a common SUB of several rank-containing SUPER chains. Our algorithm begins, then, by checking the SUPER chains of each SUB, looking for one with a ranked, unmerged SUPER; then a merge with the newly-ranked frame will be possible, as we have seen. There can be no other SUPER chains with unmerged ranked frames, because finding another such frame on a different chain (Fig. 32) would mean that a merge was possible *before* the new AKO was added (Fig. 33), and we assumed the initial system had no hidden inclusion relationships. Having determined which SUPER chain must be merged with, it remains to determine exactly where in the SUPER chain the merge should take place. This can be done by first locating the lowest SUB in the hierarchy which this SUPER chain and the newly-ranked frame have in common, and then comparing the newly-ranked frame with each ranked SUPER of this SUB. While this last modification nearly completes the algorithm, we won't pursue any further changes; it is sufficient for our purposes to note that the algorithm is quite complex for even this simple hierarchy.

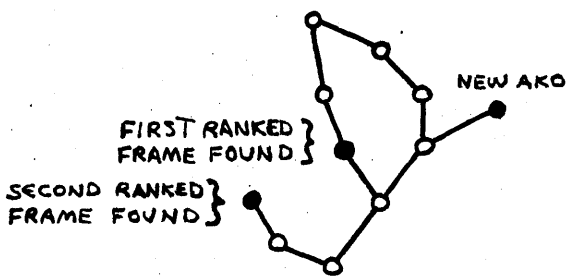


Fig. 32 Previous Unmerged Chains

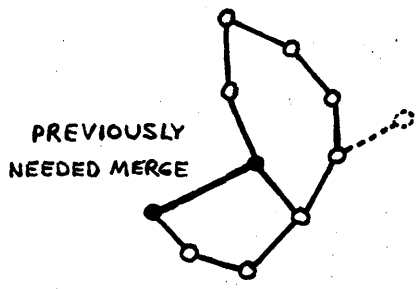


Fig. 33 A Merge Was Possible

We now have an algorithm such that if the system is properly merged before an

AKO is added, it will be properly merged afterward. This inductive approach is useful for building up the initial data base -- places are added one at a time, just as we have seen here. In practice, the full power of the complete algorithm is seldom needed, since almost always a place's AKO is known, and multiple SUPERs occur only with places of the same or unknown rank. We note, too, that a similar algorithm is needed for handling the creation of SUB-SUPER links.

We have seen that attached procedures can be used to maintain hierarchical structure in spite of incomplete information. We have also seen that ascertaining the correctness and completeness of a computation is difficult when it uses only local information to accomplish results which depend on global properties and conditions. Since this is only a simple system which ignores exceptional cases, it seems probable that if-added procedures will need more than a local perspective if they are going to be used in a well-understood, controlled fashion for more complex systems. Perhaps a formal analysis of hierarchical structures is required which focuses on the general properties of inclusion relations with incomplete and partial orderings. This analysis could be used to develop more comprehensible and general purpose algorithms for manipulating the hierarchy than those embedded in local attached procedures.

Modification Is Worse than Addition

The complications which result from deleting data in a hierarchically organized data base are more difficult to deal with than those of adding data. Sometimes these deletions are straightforward, and other times they are disguised as additions which conflict with known information. In either case, it must be decided how the hierarchy

should be restructured around the deletion. Questions are raised which can not necessarily be solved with backtracking mechanisms; there are also problems of protecting certain data, deciding what conflict-generating changes should be allowed, and resolving those conflicts which are allowed. We'll explore how some of these difficulties are manifested in the PLACE system, and then focus on treating deletions as a problem in restoring previous values. We shall see that general solutions don't seem possible within the current system, but that simplistic approaches provide some useful capabilities.

Consider the following situation: two hitherto independent inclusion chains exist (Fig. 34); the user adds the information that a place P on one of the chains also has a SUPER from one of the other chains. Assuming all the AKOs are known, the final result will be a merge of some of the SUPERs (Fig. 35):

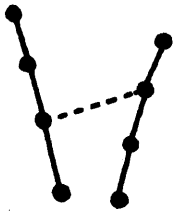


Fig. 34

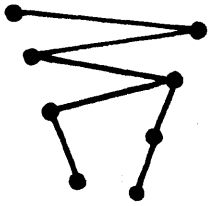


Fig. 35

Now, suppose the user decides he or she was mistaken, and removes that SUPER link which was added earlier. Does the removal of that link imply that all the changes that it originally caused should go away as well? What information must be kept around to make those corrections? Or, suppose the user decides that the USA is really a city; this will create inconsistencies among SUB-SUPER relationships. Should the USA "percolate" down the inclusion chain? Should any place ever be allowed to have its rank changed to one above that of its SUPERs or below that of its SUBs? Perhaps it depends

on how the SUBs and SUPERs got there -- whether they were explicitly specificed, or implicitly determined based on the rank of the frame in question. If allowed, how would the change be made to the data base so as to maintain the required discipline? Should the user be prevented (or at least warned) from trying to change basic inclusion and rank information of the initial world, for his or her own protection? These questions serve to illustrate the complexity of manipulating the information of the place-frames system. We'll briefly explore some of the issues they raise.

There are three particularly important characteristics of a change involving deletion. The first is the scope of the change -- how much does it affect the structure of the hierarchy? It could be an intra-frame change, such as a change in travel time, which can have affects by inheritance. It could be a change to links between frames (SUB, SUPER, and AKO links), which changes the local structure of the hierarchy. Finally, it could be a change which results in global changes, such as the rank-inclusion principle can produce.

Another characteristic of a change is how the original value was determined, and a final characteristic is how the new value is being determined. In some cases the values involved are changed explicitly by the user through answering a question the system has posed; other times values are changed because of the propagation of the direct implications of another change; sometimes the changes are the result of an educated guess or careful deduction made by the computer. One approach to handling deletions is to keep track of which of these three sources of change is involved, and how it came about. That information can be used to restore the system to its state before the data being deleted was added, thus taking alot of guess work out of how to patch the

deletion.

Each of these sources of change requires different information: the name of the user in the first case, the basic change that triggered the propagation in the second, and the information source (both data and program) of the last. Full use of all this information would require forward and backward pointers linking the place-frames involved together, leaving a trace of causal associations. A difficulty with this approach is that it is cumulative; changes interact, and soon a whole web of pointers exists linking frames in tangled ways. A CONNIVER-like approach (McDermott and Sussman, 1973) which saves each state of the system would automatically keep track of these changes; however it would not be able to "undo" a *particular* change without undoing *all* changes back to that point, and recomputing everything. A smarter system which knew more about how individual changes affected the system might be able to *selectively* restore the system without this huge cost in space and time. Finally, since the PLACE system is an interactive system, re-computation would involve re-asking the user any questions asked previously. Again, a smarter system might be able to limit the number of questions re-asked by knowing which might actually have different answers.

A less complete, but computationally less demanding approach is possible. First, an attempt is made to selectively restore the system, as mentioned above. Then a local consistency check is made to see if everything appears to be in order. Finally, a watch is kept for situations which suggest that the patch may be in error. If such evidence is observed, either the above process can be repeated, utilizing the additional knowledge of how the patch has failed, or a "back-up and recompute" phase could be entered. The level of detail at which these recomputations are made would depend on the system's

evaluation of how complex the error appears. This recompute stage can also be repeated, at increasing levels of detail.

While the CONNIVER approach appears too expensive and inappropriate for an interactive system, this compromise approach is too informal and depends on a wide range of domain knowledge and debugging knowledge. Hence the problem of recovery seems very difficult for the simple frame representation being explored here. In order to see just how much recovery support a much simpler approach could in fact provide, detailed causal records were distilled down to just the name of the user at the time a change came about. The record of any previous user was erased. For a single given change, this turned out to be an efficient means of tracing the extent that a change had propagated, making it useful for debugging the system. It can also be used to detect collisions between user and system information (indicated by a NIL source). For example, suppose a user tries to change the SUPER of MASS from USA to USSR. Since countries form a mutually exclusive set, this indicates a direct collision between the user and the initial system value. Changes which do not result in such discrepancies, such as supplying the state for a city for which the state had been previously unknown, would be allowed, even though they result in changing system values (the SUPER of the city would change). Certain classes of information, such as the travel data on the general rank frames, could also be protected from the user by the source information.

Recording only the user is not sufficient information to undo "mergers"; however this doesn't happen very often. It would be difficult to cause a merger inadvertently which would not also generate collisions with initial system information. This corresponds somewhat to the "local consistency check" of our compromise approach.

A more likely case than incorrect merging is the supplying of an incorrect AKO which causes a place to percolate to the wrong place in the hierarchy. However, this percolating will leave behind a trace of change by that user, which can be enough to restore the system if it is the only change. Change which is isolated from the frame which caused it can not be traced in this way.

One can use this approach to identify the effects of a change by indexing the source annotation of each change according to a unique identifier for each top-level interaction. Hence one can find the effects of any top-level interaction if one can find all instances of the appropriate unique identifiers.

Finally, the possibility of trading off look-ahead for backtracking exists -- a "before-adding" method could be used to forecast the impact that a proposed change might produce. If-added methods can not be used for this purpose, since they gain control only after a change has taken place, and until *all* if-added's have been run, the data base may be inconsistent. Predicates may have different or meaningless values than they had before the addition. Of course, one could immediately remove the value just added, hence eliminating the inconsistencies it might have caused, but then the if-removed methods would be dealing with an inconsistent state of the world. Before-adding methods would in essence serve as requirements (preconditions) for making changes to the frames-system. They are different from "pre-plans" in that they do not represent preparation for a change, but evaluation of a proposed change. Perhaps some of these inconsistency difficulties of if-added and if-removed methods could be lessened by considering them an *ordered* part of the data base; then it would be known which of them had been run and which had not.

Source information could also be used by the system to propagate its own internal "guesses" in order to determine their global effects. If sufficiently sophisticated, hypothetical reasoning would be possible. Straight low-level back-tracking with a CONNIVER-like context approach is probably insufficient, since knowing a value is only a guess could easily affect depth and breadth trade-offs in significant ways. This topic will be examined again in the next section from the perspective of top-level control.

In this section, we have investigated the changing of information in the structured data base of a hierarchically organized system. Some of the complexities of the problem have been pointed out, such as types of changes, sources of changes, and the cumulativeness of source information. These difficulties seemed too much for the current system, and so a simplified approach of recording the current user was suggested, and some of its uses and short-comings were looked at. The possibility of a "before-adding" method, to replace some back-tracking with look-ahead and to serve as a mechanism for imagining, was also discussed. To close on a hopeful note, we take heart that "real" systems which deal with place will probably change *less* than toy ones, since place information is basically static.

In summary, we see that the current implementation lacks sufficient power to handle the problem of restructuring incomplete hierarchies when information is added or deleted. It appears that a stronger formal analysis is needed for dealing with the manipulations required for additions, and that the handling of deletions will require more processing power and more exploration of recovery heuristics.

VI. User Interaction with a Hierarchical System

As a knowledge base grows larger, it becomes increasingly important that a user be able to add more knowledge to it without having to know the details of its structure and content. For a hierarchical system, this becomes a problem of classification; this is explored in the first section of this chapter. It is also important that the system be flexible enough to accommodate the requirements of a given user without requiring special detailed knowledge. Hence the second section explores dimensions of control that a user should be able to exercise, and discusses some features of hierarchical systems which can interfere with that control.

Supplying New Information

One difficult problem faced by hierarchical representations is that of dynamically integrating new information into its data base. Getting the user and the machine to cooperate in classifying new facts for such a disciplined data base is crucial. That problem is approached here by having the computer provide the user with appropriate choices generated from information provided by values, preferences, defaults, requirements, and exemplars.

The issue we will examine is that of accurate classification -- whenever a new place is being added to the data base, it is desirable to narrow down the type (AKO) of the place as far as possible, thereby allowing maximum reasoning accuracy. The problem is that the user may not know explicitly what types the machine knows about, and so an attempt must be made to supply the user with likely choices sufficiently fine-grained that the most concise answer is obtained. Where there are many possibilities available (as

there are for types of places), listing all the choices seems cumbersome. Several alternatives are possible. One is to pick a smaller number of larger categories, and then expand sub-categories based on the response. This raises the difficulty, however, of getting the machine to come up with "larger" or "more general" categories on its own, unless the categories are already a part of the hierarchy. A variation on this approach is to provide a choice just among the most common known categories at a given abstraction level, reserving the additional category "other" for the more uncommon categories. The PLACE system has all the facilities required for this kind of interaction, as well as additional ones. Here is a possible system-directed dialog where the machine guides the user through several levels of hierarchy -- place to building to store to supermarket. The computer asks the question, provides the choices, and prompts the user for his or her response with the ">" character.

What kind of place is A&P?

(A) room	(D) state
(B) building	(E) country
(C) city	(F) other

>B

What kind of building is A&P?

(A) store	(D) house
(B) school	(E) office
(C) factory	(F) other

>A

What kind of store is A&P?

(A) restaurant	(D) department
(B) fast-food	(E) other
(C) grocery	

>E

What kind of store is A&P?

(F) clothing (I) drugstore
(G) hardware (J) other
(H) supermarket

>H

Difficulties arise with this system-directed interaction. One difficulty is that the user may try to force a category match if the common categories aren't covered very well by the given choices. An example of this would be the categorization of MIT under the choices room, building, city, political-unit, and country; the answer the system would prefer is "complex", as in building complex. However the user may prematurely match the type of MIT to building before examining "other". This difficulty is lessened when there is a distinct break between the common categories and the not so common.

Another difficulty is that the suggested categories may not be sufficiently disjoint from categories not yet listed to prevent generally correct, but less than completely accurate classification. For example, "school" might be chosen for the type of MIT, when the more accurate choice "university" would have appeared as a choice only if "other" had been selected. Similarly, "grocery" could have been prematurely chosen instead of "other" (which lead to super-market) in the A&P example, above. It is easy to see that these classification problems would be even more difficult if unfamiliar categories, chosen for the machine's purposes, were used, such as the subsets of a STRUCTURES category, like ROOM-OR-ROOMS and BUILDING-OR-ROOMS.

An alternative that addresses these difficulties is to reverse the roles of the computer and user by allowing the user to take the initiative in supplying a category. The computer could then see if it recognizes the category. For example, instead of

pursuing a menu style dialog as above in classifying MICH, the following format could be used:

What kind of place is MICH?

>state

Or: What kind of place is MICH?

>region

I don't know about regions. Should I:

(A) wait for another response?
(B) give possible choices?

> etc.

In the cases where the machine is aware of more specific categories, it could still pursue them:

What kind of place is MICH?

>political-unit

Can you be more specific?

>yes

What kind of political-unit is MICH?

>state

This kind of interaction where the user has the initiative is desirable, but puts great demands on the breadth and depth of the system's knowledge. Hence the PLACE system explores an intermediate position: the system still controls the interaction via a menu-style dialog, but it generates choices using a large number of refinements over just offering a list of possible alternatives. These refinements include the ones we have

already discussed, such as limiting the number of alternatives, the use of an "other" category for the more uncommon choices, and the asking for greater specificity. Additional features discussed below include the use of examples to clarify categories, attention to type-token distinctions in choice and example generation, and the use of domain heuristics in deciding which choices to suggest.

Sometimes a category may not make much sense to a user, because it reflects a generalization which is not very common. For example, state, province, and shire are grouped under the category political-unit. The PLACE system generates "such as" examples as an explanation by example:

(A) metro-area	(such as Metro-Boston, Metro-Detroit)
(B) political-unit	(such as state, province)

Just as it is best to suggest most likely categories first, thereby eliminating excessive confusing detail, it is best to supply "such as" examples only where most needed. Having lists of examples for every type, including even the common ones such as "city", becomes cumbersome to the user, and obscures those cases where examples may really be needed. A question of the semantics of examples also arises. It must be clear that the "such as" examples are meant to clarify the meaning of the category, and not to serve as possible options for the response.

Some issues also arise in choosing what kind of examples are required. Choice "A" above uses specific places (Metro-Boston, Metro-Detroit), but choice "B" uses category types (state, province). Determining which, if either, is better, and why, is important to optimally aiding the user. For example, MICH is of type state; it would make no sense to offer USA as a possible type for MICH. Conversely, it would make no sense to offer

"country" as a possible SUPER of MICH; only specific places should be offered. If this kind of specific-general distinction is important to supplying intelligent choices, the system has to know whether a place-frame is a specific place ("individual") or general category ("non-individual"). Such information is kept in the SELF slot of each frame; the question of specificity is dealt with as each new place is added to the system.

Another way of offering intelligent choices to the user is to make logical guesses based on either deductions or heuristics. For example, if MICH is a SUB of the USA, and a SUPER of DETROIT, then the best choices for its type are SECTION, STATE, or METRO-AREA. Hence only these three should be offered as choices. The advantage of suggesting a correct answer right away is balanced against having to start over if the guess is not right, or close.

All these added refinements of choice generation come at a cost of increased information requirements: somehow the machine must know or deduce how to sub-categorize the various categories; it must know which categories are common in order to assign types to the "other" category and to know when to provide "such as" examples; it must have type-token information; and it must have some domain knowledge to determine additional restrictions. Some of this information is already available in the system, such as the rank categories and "individuality" information. The rest of the needed information is stored in specific place-frames and general place-frames under five different facets which can occur under any slot: $DEFAULT, $PREFER, $REQUIRE, $EXEMPLAR, and $VALUE. The PLACE system tries to exploit the information provided in these facets to generate choices which lead to acceptable user-machine interaction.

The semantics for the various classes are still evolving, but here are some of the general distinctions being made. Defaults are used to supply the "usual" value one might expect in the absence of more definite information. For example, a state is assumed to be of medium size if no other information is available. In general, it seems appropriate to verify the value of a default, particularly when the value is clearly related to the user's inquiry or statement. Preferences are used to create expectations of type that guide giving choices. For example, usually the SUPER of a city is either a state or a country; that information is stored in the system as a preference for state or country on the SUPER slot of city. Requirements on the other hand represent restrictions on values which must hold if the discipline of the data base is to be maintained. The rank and inclusion relations are enforced as requirements on SUBs, SUPERs, and AKOs. Requirements would also used to suggest possible types if no preference information is available, or if such information proves to be inadequate. Exemplars are used to provide "canonical" examples or classes of examples for a particular slot value. This is helpful in deriving examples to explain a category through "such as" examples (values from the "instance" slot can serve the same purpose if no exemplars are listed), or in picking out the major sub-categories of a type from all the possible sub-categories. The major sub-categories can be the major options offered the user, and the minor sub-categories supply the options to be used under "other". Also, being an exemplar indicates that a "such as" expansion is probably not needed. Finally, the value facet is used to hold actual known values. The type of a value can sometimes be used for deducing possible AKO categories for other values in similar places in the place hierarchy. For example, if the SUPER of a particular place has a value which is AKO building, the system might

expect that similar places might also have SUPERs of AKO building.

An alternative to this approach of using separate facets would be to have an "association index" facet whose value increases with the strength of the association. This would reflect the continuum aspect of the classes: a strong preference is little different from a weak requirement; a strong default is like a value with a few exceptions; etc. However, while the FRL system inherits differentially for different facets, there is currently no elegant way to direct it to inherit differentially for different values on a continuous scale. Finally, we are presently concerned only with gross differences along this scale; the other place functions can not handle answers along a continuum, only discrete categories. This continuum approach will have to wait for a more sophisticated system to be adequately tested.

There are some difficulties with the multi-facet approach as well. For example, how far up the tree should these classes be inherited? Certainly inheriting makes a lot of sense if no other values are available, but when several or even a great many exist, priorities have to be developed to choose which to use. Currently, inheritance is used very sparingly (chaining stops when a value is found). If, for example, all the requirements of a particular slot were used in generating choices, many of the suggestings would be much too general, coming from even the top level place frame. The inheritance primitive alone lacks sufficient high-level perspective to perform appropriately; there should be some provision for higher-level input.

Using the Slot Facet Categories

In the current system, a function called SAMPLE looks at a specified facet in a place-frame, and returns the specific places and general types that it finds there, translating requirements into frame names where needed. For example, suppose the prefer facet of the SUPER slot of CITY has the following form:

```
(CITY   . . .
        (SUPER  ($PREFER     ((RESTRICT-AKO :VALUES
                                            '(POLITICAL-UNIT
                                              METRO-AREA)))))
        . . . )
```

Then (SAMPLE 'CITY 'SUPER '$PREFER) yields (POLITICAL-UNIT METRO-AREA). SAMPLE is used by a function called EXAMPLES, which maps SAMPLE over the various facets in a specified order to generate a specified number of suggestions, usually around five to ten. The facet ordering ($PREFER, $EXEMPLAR, $REQUIRE, $VALUE, $DEFAULT) reflects the relative strength of the associations, and serves to establish a priority among choices, the less likely ones going into the "other" category. For example, suppose the PLACE, CITY, and CAMBRIDGE frames have facets as follows under their SUPER slots:

```
(PLACE  . . .
        (SUPER  ($REQUIRE    ((>= (RANK :VALUE)  (RANK :FRAME)))))
        . . .  )

(CITY   . . .
        (SUPER  ($PREFER     ((RESTRICT-AKO :VALUES
                                            '(POLITICAL-UNIT
                                              METRO-AREA))))
                ($EXEMPLAR   (METRO-DETROIT) (USA))
                ($DEFAULT    (STATE))))
        . . .  )
```

```
(CAMBRIDGE  . . .
        (SUPER  ($VALUE      (METRO-BOSTON)))
        . . .  )
```

Then (EXAMPLES 'CAMBRIDGE 'SUPER) would first compute, in order, the results for each facet:

$PREFER	POLITICAL-UNIT METRO-AREA
$EXEMPLAR	METRO-DETROIT USA
$REQUIRE	CITY METRO-AREA POLITICAL-UNIT COUNTRY CONTINENT WORLD
$VALUE	METRO-BOSTON
$DEFAULT	STATE

Then it would combine them, in order, to get:

(POLITICAL-UNIT METRO-AREA METRO-DETROIT USA CITY METRO-AREA POLITICAL-UNIT COUNTRY CONTINENT WORLD METRO-BOSTON STATE)

Many different kinds of post-processing are available to refine this list so it can be used effectively. First, duplicates are eliminated, but the order, which reflects the priority of the facets, is preserved. If the choices required must be general types, then each individual is replaced by its AKO value. This way, a type which is associated with the given frame and slot by specific value only still contributes to the choice list. Duplicates are again removed, while still preserving order. The first four choices are given, with the next set being accessed if "other" is chosen:

First four:	POLITICAL-UNIT
	METRO-AREA
	COUNTRY
	CITY
Second set:	CONTINENT
	WORLD
	STATE

A further refinement is the generation of "such as" examples for those choices which may not be well-known. For example, POLITICAL-UNIT would have the examples "state"

and "province". Types which have no sub-types would have examples of individuals, found by using the EXAMPLES function. How can the system tell if a type is uncommon and hence requires examples? Currently, it checks to see if that type appears as an exemplar under its own AKO. POLITICAL-UNIT has AKO PLACE, and the exemplar facet of the INSTANCE slot of PLACE does not contain POLITICAL-UNIT. Hence it is given examples.

One further refinement is to replace a general type which is uncommon by an exemplar of its sub-types, and put the more general category under "other". Hence POLITICAL-UNIT would be replaced by STATE in the first set of options, and added to the second set. The actual choice list would appear as follows:

What is the super of Cambridge a kind of?
(A) state
(B) metro-area (such as Metro-Detroit)
(C) country
(D) city

The first three choices are reasonable; the fourth, CITY, is not very reasonable, since cities do not usually contain other cities. This choice was generated by the require facet of the general PLACE frame, and reflects a common difficulty with inheritance: the requirement is from a very general level, and hence is very apt to be necessary, but not sufficient, to guarantee a reasonable candidate for a slot far down in the hierarchy. In this example, it suffices to re-order the mapping of facets done by the EXAMPLES functions so that the require facet is done, say, last. A more general solution should distinguish necessary requirements from necessary *and sufficient* requirements.

As has been mentioned, the manipulations which can be done with general types can also be done using individuals -- all the types can be replaced by exemplars

and/or instances of individuals. These manipulations can be integrated to produce special combinations, such as a specific individual for each general type (state, metro-area, country ==> MICH, METRO-BOSTON, USA). While still in the early stages of development, these heuristics serve to illustrate how the general knowledge in the various facets can be effectively used to generate intelligent choices. It is hoped that continued refinement of these heuristics will result in a broad-based, robust strategy which can be used not only for choice generation, but also for guessing at missing information.

Some issues of dynamically integrating data into a hierarchically organized data base have been examined. Problems of providing appropriate choices for the user's response have been approached using information provided by values, preferences, defaults, requirements, and exemplars. It is suggested that developing a language for examples and their manipulation is important for communicating with any interactive, knowledge-rich system.

Top-level Control

We have seen that accurate classification is particularly important when dealing interactively with hierarchical systems. The are several more issues that an interactive system must address which significantly affect user interaction: dialog control, context control, reason control, resource control, and validity control. While not peculiar to hierarchical systems, these issues rapidly become important as a knowledge base expands, and hierarchy allows for that kind of rapid expansion. Also, hierarchy lends itself to natural abuse in these areas. I will briefly discuss each area.

In course of asking the user questions, intelligence if often displayed by dialog

control -- that is, by not asking certain questions, such as ones which would not make sense to the user, ones which have already been dealt with recently, ones which try the user's patience, etc. For example, perhaps the system needs the knowledge of whether a frame is an "individual" or not. The user may have been asked the question, and responded that he or she didn't know, or ignored the question. An intelligent system should somehow record this response so that the user isn't continually bothered with a question he or she has chosen not to answer. Or, perhaps the user is working in a certain problem domain where all the places mentioned are AKO restaurant; the user should be able to tell the system this fact. The user may be in a hurry, and wishes only essential comments, not long-winded embellishments. All of these examples relate to a knowledgeable user-machine interface: intelligent dialog.

Hierarchical systems can subvert the intent of intelligent dialog by using local attached procedures to ask the user questions. The local nature of these procedures prevents them from considering the issues mentioned above, such as recent answers to the same question in a similar context. Hence an uncontrolled, naive dialog results. The intelligent use of hierarchy requires that dialog be under structured control. It seems quite probable that this control could be embedded in a hierarchy of dialog frames which used inheritance to relate previous subjects, questions, and answers.

Context control is the ability of the system to set up its own "context" for its operation. This involves being able to "keep notes" for itself. For example, it may want to make an assumption, such as "Detroit is an individual", or "Detroit is AKO city". It may want to suppose that states are mutually exclusive, or that its knowledge of the members of the Midwest is collectively exhaustive. Such guesses or assumptions can be

used to forecast the global effect of local changes, supply more refined guesses, supply a piece of missing information, anticipate a user's response, or keep track of general facts the user has supplied. One can also imagine wanting to ignore particular pieces of data, or temporarily replacing them with others.

Again there is a need to structure this kind of information, so that the system can provide the flexibility the user expects without loosing sight of what it's doing. The inheritance mechanisms of hierarchical schemes tempt one to embed this information throughout the system as individual values; this is sufficient for simple problems, but when dealing interactively, it will be necessary for the system to know what its current assumptions are, and why. This information should be a structured part of the frame hierarchy, which can be accessed by the dialog frames.

Reason control is the flow of task-oriented control that is followed in the system. For example, when the user is classifying a new place, normal control flows from one attached procedure to the next, until each slot has been covered. This is a simple technique which suffices for simple systems. But as the knowledge base grow in complexity, there will be slots on each place for many more kinds of information, such as time, people, activities, and facilities. It should be possible for the system to control the flow of operation so that only certain kinds of information is requested, or only a certain level of detail is pursued. Again, this involves taking control away from the local attached procedures, and giving it to a complex of frames which contain knowledge about level of detail and appropriateness of information; these should be accessible to the dialog and context frames.

Finally, we consider resource control and validity control. Whenever a large

system is involved, it is likely that there are several ways that it can proceed to determine a given piece of information. We explored this idea in chapter IV when we looked at refinements of NEAR. Some methods are more costly than others, but they provide more accurate results. In an interactive system, it should be possible to direct the operation of the system so that concerns for computation time and logical validity can be taken into account. This is difficult to do if the system relies too heavily on fixed inheritance and local attached procedures. A more general scheme is required which accommodates these cost-validity considerations. Such a scheme would include the use of a multi-valued logic for expressing varying degress of certainty.

Together, these different kinds of control capabilities should make it possible for the user to direct the system's operation towards the end he or she has in mind without becoming frustrated by the system's mode of interaction. The development of hierarchical systems should proceed with a sensitivity to the importance of top-level context.

Appendix A

Actual "Near" Scenario

This is the actual LISP version of the "Near" scenario in chapter I. The mechanism for handling specific exceptions (the function NEAR:) was not implemented, due to its lack of generality; it would be easy to do so, however, as the explanations below demonstrate.

```
(NEAR?  'MASS  '(BOSTON CAMBRIDGE)) ==> T

(NEAR?  'NORTH-AMERICA  '(BOSTON NEW-YORK)) ==> T

(NEAR?  'NORTH-AMERICA  '(BOSTON MONTREAL)) ==> NIL

(NEAR?  'NORTH-AMERICA  '(DETROIT WINDSOR)) ==> NIL

(NEAR:  '(STATE PROVINCE)  '(DETROIT WINDSOR)) ==> DONE.
```

NEAR: would add a NEAR slot to Detroit and Windsor which would compute all their respective SUPERs which were of rank greater than or equal to the specified ranks.

```
(NEAR?  'MICHIGAN  '(DETROIT WINDSOR)) ==> T
```

NEAR? would catch the exception by checking Detroit and Windsor's SUPERs for a NEAR slot, and intersecting the results with the given place, Michigan.

```
(NEAR?  'ONTARIO  '(COBO-HALL WINDSOR-MUTUAL)) ==> T
```

Detroit and Windsor would be found when the SUPERs were computed, and hence so would their NEAR slots.

```
(NEAR?  'MICHIGAN  '(CENTERLINE FAIRFIELD)) ==> NIL
```

Detroit and Windsor do not occur as the SUPERs of Centerline or or Fairfield, and hence their NEAR slot would not be found.

Appendix B

Sample Place-frames

```
(fassert PLACE
        (ako            ($val (thing))
                        ($if-added ((progn
                                    (forall v (fget-values
                                              :frame 'ako)
                                             (fadd-value+source
                                                 v
                                                 'instance
                                                 (fname :frame)))
                                    (forall v (fget-values :frame 'sub)
                                           (check-improper-supers v
                                            (fname :frame)))
                                    (suggest*super)
                                    (forall v (fget-values :frame 'sub)
                                           (check-improper-supers v
                                            (fname :frame))))))
                        ($if-removed ((fremove-value (fget-value
                                                     :frame
                                                     'ako)
                                                     'instance
                                                     (fname :frame)))))
        (instance       ($val (city) (school) (building) (room))
                        ($exe (room) (building) (city)
                              (state) (country))
                        ($req ((restrict-ako :vs '(place)))))
        (super          ($if-added ((progn (add-sub)
                                           (check-improper-supers
                                            (fname :frame)
                                            (fname :value))
                                            (suggest*ako)
                                           (check-improper-supers
                                            (fname :frame)
                                            (fname :value)))))
                              ($if-removed ((remove-sub)))
                              ($req ((restrict-ako :vs '(place))))
                        ($req ((forall v :values (greaterp
                                                 (rank (fname :frame))
                                                 (rank v))))))
        (sub            ($if-added ((and (add-super) (suggest*ako))))
```

```
                        ($if-removed ((remove-super)))
                        ($req ((restrict-ako :vs '(place))))
                    ($req ((forall v :values (lessp
                                    (rank (fname :frame))
                                    (rank v))))))
        (self       ($type (non-individual)))
)

(fassert ROOM
        (ako        ($val (meeting-place)))
        (super      ($pre ((restrict-ako :vs '(building-area
                                    building)))))
        (capacity   ($req ((integer-range? :v)))
                    ($def ( (integer-range 5 10))))
                          ; The number of people it can hold.
        (phone-number)     ; Unfortunately, places, and not
                           ; people, have phones.
        (facilities ($def (chairs) (light-fixture)))
        (users       ($def (guests-of 'primary-person)))
        (primary-person ($def ( (assignee :frame))))
        (care-takers ($def ( (owner :frame))))
        (user-focus  ($exe (facilities) (convenience) (privacy))
                    ($def (assignment)))
        (available  ($def ( (interval (am 8) (pm 11))))
                    ($req ((null (overlap? :v
                                    (schedule-of :frame))))
                          ((interval? :v))))
        (self       ($type (non-individual)))
)

(fassert OFFICE
        (ako        ($val (room)))
        (instance   ($exe (ne43-819))
                    ($val (ne43-819)(ne43-823)))
        (facilities ($val (desk) (waste-basket)))
        (primary-activity ($def (paper-work)))
        (self       ($type (non-individual)))
)

(fassert NE43-819
        (ako        ($val (office)))
        (super      ($val (545-technology-square (floor: 8))))
        (primary-person($val (ira)))
        (self        ($type (individual)))
)
```

```
(fassert CITY
        (ako            ($val (place)))
        (instance       ($val (cambridge)(san-francisco)))
        (intra-travel    ($val (car (time: (minute 20))
                                    (cost: .35))
                               (taxi (time: (minute 20))
                                     (cost: 3))
                               (bus (time: (minute 25))
                                    (cost: .25))
                               (by-foot (time: (minute 40))
                                        (cost: 0))))
        (super          ($pre ((restrict-ako :vs '(state metro-area)))))
        (sub            ($pre ((restrict-ako :vs '(building
                                                   complex)))))
         (self          ($type (non-individual)))
)

(fassert BOSTON
         (ako           ($val (city)))
         (super         ($val (mass)))
         (self           ($type (individual)))
)

(fassert CAMBRIDGE
         (ako           ($val (city)))
         (super         ($val (mass)))
         (self           ($type (individual)))
)

(fassert STATE
         (ako           ($val (political-unit)))
         (super         ($val (usa)))
         (self          ($type (non-individual)))
)

(fassert MASS
         (ako           ($val (state)))
         (super         ($val (usa)))
         (self           ($type (individual)))
)

(fassert CAL
         (ako           ($val (state)))
         (self           ($type (individual)))
```

```
)

(fassert COUNTRY
        (ako           ($val (place)))
        (super         ($pre ((restrict-ako :vs '(continent)))))
        (intra-travel  ($val (plane (time: (hour 6))
                                    (cost: 100))
                             (car (time: (hour 10))
                                  (cost: 35))
                             (train (time: (hour 8))
                                    (cost: 50))))
        (self          ($type (non-individual)))
)

(fassert SCOTLAND
        (ako           ($val (country)))
        (super         ($val (europe)))
        (self          ($type (individual)))
)

(fassert USA
        (ako           ($val (country)))
        (super         ($val (north-america)))
        (sub           ($val (cal) (mich) (east-coast)))
        (self          ($type (individual)))
)

(fassert WORLD
        (ako           ($val (place)))
        (intra-travel  ($val (plane (time: (hour 12))
                                    (cost: 500))))
        (self          ($type (non-individual)))
)

(fassert EARTH
        (ako           ($val (world)))
        (sub           ($val (north-america) (europe)))
        (self          ($type (individual)))
)

(fassert RESTAURANT
        (ako           ($val (building)))
        (instance      ($val (dodin-bouffant)))
        (users         ($val (public)))
        (user-focus    ($def (social-event) (food-quality)))
```

```
        (activity       ($val (eating)))
        (self           ($type (non-individual)))
)
```

Appendix C

Sample Place System Functions

```
(setq PLACE-FUNCTIONS
      '(near? contains? place-span minimize maximize
             continent travel-cost travel-means
             travel-time specify-place describe-place))

(defprop PLACE-SPAN
         ((inputs:      (<place-name> <place-name> etc ) )
          (return:      (<place-type> <place-name> <place-name> etc.))
          (function-type: generative)
          (action:      returns the type of the smallest
                        common abstraction level and the
                        places of that type (usually one))
          (example:     (place-span /'(mit maine)) =>
                               (section new-england east-coast)))

         comments)

(defun PLACE-SPAN (place-names)
       (prog (common ans place-type)
             (ifnot (forall p place-names (ako? p 'place))
                    (print (list 'not
                                 'all
                                 'places:
                                 place-names)))
             (setq common
                   (apply 'intersectq
                          (foreach p
                                   place-names
                                   (cons (fname p)
                                         (fdescendents p
                                                       'super)))))
             (setq place-type (rank))
             a
             (cond ((setq ans
                          (mapappend '(lambda (place-name)
                                              (if
                                               (memq
                                                (car
                                                  place-type)
```

```
                                                (cons place-name
                                                      (fdescendents
                                                 place-name
                                                 'ako)))
                                                (list
                                                 place-name
                                                 )))
                                        common))
                         (return (cons (car place-type) ans)))
                        ((null (pop place-type)) (return nil))
                        (t (go a)))))

(setq place-types '(meeting-place building-area building
                                  sub-complex complex
                                  city metro-area
                                  political-unit section
                                  country continent
                                  world))

(defun PLACE-TYPE? (place type)
       (mapappend '(lambda (place)
                           (cond ((null (memq type
                                              (fdescendents place 'ako)))
                                  nil)
                                 (t (list place))))
                  (cons (fname place) (fdescendents place 'super))))
)

(defun ROOM (place)
       (place-type? place 'room))

(defun STATE (place)
       (place-type? place 'state))

(defun CITY (place)
       (place-type? place 'city))

(defun TRAVEL-MEANS (place-list)
       (setify (mapappend '(lambda (place)
                                   (fproperties place
                                                'intra-travel
                                                '$val))
                          (place-span place-list))))

(defun TRAVEL-COST (place-list)
```

```
       (travel-news place-list 'cost:))

(defun TRAVEL-TIME (place-list)
       (travel-news place-list 'time:))

(defun MINIMIZE (topic place-list)
       (assoc-pred 'min* (travel-news place-list topic)))

(defun NEAR? (region place-list)
       (prog (span descendents ans)
             (setq span (place-span place-list))
             (and (null span) (return nil))
             (setq descendents (fdescendents region 'sub))
             (setq ans (intersectq (cdr span) descendents))
             (and (null ans) (return nil))
             (return (cons (car span) ans))))

(defun CONTAINS? (outer-place inner-place)
       (and (memq inner-place (fdescendents outer-place 'sub))
            t))

(defun RANK args
  (prog (ans)
        (cond ((equal args 0) (return place-types))
              ((null (arg 1)) (return nil))
              ((numberp (arg 1))
               (return (nth (difference (add1 (length place-types))
                                        (arg 1))
                            place-types)))
              ((equal (typep (arg 1)) 'symbol)
               (ifnot (fget-values (arg 1) 'ako) (return nil))
               (if (setq ans (memq (arg 1) place-types))
                   (return (length ans)))
               (return (length (memq (car (place-span (list (arg 1))))
                                     place-types)))))))

(defun RANK-NAME args
  (prog (ans)
        (cond ((equal args 0) (return (rank)))
              ((null (arg 1)) (return nil))
              ((numberp (arg 1)) (return (rank (arg 1))))
              ((equal (typep (arg 1)) 'symbol)
               (return (rank (rank (arg 1))))))))

(defun CHECK-IMPROPER-SUPERS (frame new-super)
```

```
  (prog (rank-new rank-old)
        (setq rank-new (rank new-super))
        (return (mapappend '(lambda (old-super)
           (progn (setq rank-old (rank old-super))
                  (return (cond ((greaterp rank-new rank-old)
                                 (fremove-value old-super 'sub frame)
                                 (fadd-value+source old-super
                                                    'sub
                                                    new-super)
                                 (print (list 'removed
                                              frame
                                              'from
                                              old-super
                                              'added
                                              new-super
                                              'to
                                              old-super)))
                                ((lessp rank-new rank-old)
                                 (fremove-value frame
                                                'super
                                                new-super)
                                 (fadd-value+source old-super
                                                    'super
                                                    new-super)
                                 (print (list 'percolating
                                              new-super
                                              'thru
                                              frame
                                              'to
                                              old-super)))
                                (t nil)))))
                           (setminus (fget-values frame 'super)
                                     (list new-super)))))))

(defun DESCRIBE-PLACE (frame)
       (prog (ans)
             (terpri)(terpri)
             (return (mapappend '(lambda (type)
                                   (cond ((setq ans
                                                (apply
                                                      'place-type?
                                                      (list
                                                          (fname
                                                           frame)
                                                          type)))
```

```
                                        (princ (car
                                                 (fget-values
                                                   (car ans)
                                                   'ako)))
                                        (princ '|:    |)
                                        (tyo 9)
                                       (princ (car ans))
                                        (foreach x (cdr ans)
                                                (terpri)
                                               (tyo 9)
                                               (tyo 9)
                                               (princ x))
                                        (terpri)
                                        ans)
                                       (t nil)))
                              (rank)))))

(defun SPECIFY-PLACE args
       (prog (ans ako name des temp temp1)
             (if (greaterp args 1)
                 (setq ako (arg 2)))
             (if (or (equal args 0) (null (arg 1)))
                 (setq name (request2 (list '|what is the name of the|
                                            (or ako '|place|) '?) nil))
                 (go c))
             (if (greaterp args 0)
                 (setq name (arg 1)))
         c   (if (null (frame? name))
                 (fcreate name))
             (if (greaterp args 2)
                 (cond ((eq (arg 3) '?)(request-individual name))
                       ((arg 3) (make-individual name))
                       (t (make-non-individual name))))
             (and ako (fadd-value+source name 'ako ako))
             (or ako (setq ako (car (fget-values name 'ako))))
             (or ako (suggest* 'place 'instance (list name) 'ako))
          a  (cond ((and (setq des (describe-place name))
                         (request2 (append '(is this correct for)
                                           (list name)
                                           '(:))
                                   nil))
                    (print (list 'place name 'located))
                    (return (ascii 0)))
                   ((setq ans
                          (request2 '(what would you like to do?)
```

```
                    '(|replace a place name|
                      |delete a place name|
                      |add to an existing category|
                      |add a new category|)))
          (cond ((equal ans '|add a new category|)
                 (and (setq temp (type-range des))
                      (setq ans (request-choice
                                  (expand-individuals
                                    temp 3)))
                      (setq temp1 (next-down ans des))
                      (specify-place nil ans t)
                      (foreach p temp1
                              (for (:frame (fname ans)
                                    :slot 'sub
                                    :value name)
                               (fadd-value+source (fname ans)
                                                  'sub name)))
                      (go a))
                 (print '|addition failed|)
                 (go a))
                ((print '|can not do that yet|)
                 (go a)))
                (t (print 'ok)))
(return (ascii 0))))
```

Bibliography

Denofsky, M. E. "How near is near?"
AI Memo #344, MIT AI LAB, February, 1975.

Goldstein, I. P. & R. B. Roberts. "NUDGE, a knowledge-based scheduling program".
AI Memo #349, MIT AI LAB, February, 1977.

Kuipers, B. "Spatial knowledge".
AI Memo #359, MIT AI LAB, June, 1976.

Kuipers, B. "Representing knowledge of large-scale space".
Doctoral dissertation, MIT Mathematics
Department, MIT, May, 1977.

McDermott, K. and G. J. Sussman. "The Conniver Reference Manual".
AI Memo #259a, MIT AI LAB, 1973.

Moon, D. MACLISP Reference Manual.
MIT, April, 1974.

Roberts, R. B., and I. P. Goldstein. "The FRL Manual".
AI Memo #409, MIT AI LAB, September, 1977.

Rumelhart, D. E. "The room theory".
Unpublished computer printout, February, 1974.

Winograd, T. "Frame representation and the declarative/procedural controversy". In D. Bobrow & A. Collins (Eds.), Representation and Knowledge, Academic Press, Inc., New York, 1975.