9. Accelerating Deductive Inference: Special Methods for Taxonomies, Colours and Times

Lenhart K. Schubert
Mary Angela Papalaskaris
Jay Taugher

Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1

Abstract

Deductive reasoning in a question answering system could in principle be left entirely to uniform inference methods such as resolution or natural deduction. However, more efficient special methods are needed for determining certain kinds of relationships which people seem to grasp 'without thinking'. These include relationships among types (such as person, girl, and computer), among parts (such as Canada, Alberta, and Alaska), among colours (such as brown, tan, and orange), and among times (such as the times of the Apollo 11 mission, the first moonwalk, and the first space shuttle launch). We outline special graphical and geometric methods for determining such relationships efficiently. The times required are often nearly constant and the storage costs linear in the number of relevant facts stored. Such special methods can be combined straightforwardly and uniformly with a general deduction algorithm.

Based on the paper, Determining Type, Part, Color, and Time Relationships, L. Schubert, M. Papalaskaris, and J. Taugher, appearing in COMPUTER, Volume 16, Number 10, October, 1983.

9.1. Introduction

At the University of Alberta, we are trying to construct a system with enough commonsense knowledge, and fluency in English, to be able to answer simple questions about a wide range of mundane subjects.[19] We' would like the system to respond quickly, and to remain structurally comprehensible, no matter how large or varied its knowledge base. Thus our emphasis has been on the generality of both the knowledge representation and of the way in which knowledge is organized for efficient selective access (Schubert, Goebel, and Cercone, 1979), (Covington and Schubert, 1980), (deHaan, 1986).

We will be concerned here with several kinds of inference problems, problems which arise constantly in question-answering processes and, without special handling, can absorb large computational resources. One kind requires determining how two types of things are related, for example, whether "person" subsumes "girl", or whether "girl" is incompatible with "computer"; others require determining similar taxonomic or ordering relationships among parts of objects, colours, or times. These relationships are of fundamental importance in our perception and conception of the world, and it seems likely that we are specially equipped to deal with them efficiently. To match our cognitive skills, AI systems will need analogous special methods.

The special methods we shall describe are designed to supplement a deductive question-answering algorithm which is now operational (deHaan, 1986), (deHaan and Schubert, 1986). The algorithm draws on a base of logical propositions organized as a semantic net. The net permits selective access to the contents of individual 'mental worlds' and narratives, to sets of entities of any specified type, and to propositions involving any specified entity and classified under any specified topic. For

---

[19] Questions are at present posed logically, the English front end being incomplete. Our approach to parsing, interpretation, and generation of English is described in (Schubert and Pelletier, 1982), (Schubert, 1982), (Schubert, 1984), (Bailes, 1986).

example, if the story "Little Red Riding Hood" is inserted into the net (in logical form), the set of all propositions concerned with the wolf's appearance can be separately and efficiently retrieved. More narrowly, just the propositions concerned with the wolf's colour (a subtopic of appearance) can be selected. Using other topical categories, propositions describing feeding episodes, character traits, and so on, of the wolf or of any other particular or generic concept can be efficiently accessed.

The net syntax permits storage of arbitrary first-order or higher-order formulas, but the deductive algorithm requires stored propositions to be in clause form. (The input routines which convert quantified, arbitrarily connected first-order formulas into clauses also permit modal operators such as "necessarily" and "believes", generating a type of modal clause form; however, the deductive algorithm currently does not deal with modal operators.) The choice of clause form was not motivated by any prior commitment to resolution, but rather by the requirements of the topical classification algorithm on the one hand and the objective of minimizing equivalence inferencing' (for example, inferring $A \rightarrow \neg B$ from $B \rightarrow \neg A$) on the other. Nevertheless, this choice of canonical form has made it natural to rely on resolution as the main inference rule.

The deductive algorithm for answering yes-no questions concurrently tries to refute the clauses corresponding to the given question (for a "no" answer) and the clauses corresponding to its negation (for a "yes" answer). The resolution control strategy incorporates set-of-support and unit preference, but unlike standard strategies, trades off resolving against retrieval of additional information, and generally restricts resolving to pairs of clauses lying on a common path in a concept hierarchy and classified under the same topic. Most importantly for present purposes, it provides for the use of special methods to simplify clauses and to resolve and factor them.

We have described our system as a semantic net. In doing so, we are not speaking from a particular camp. We believe that the issues we are addressing are bound to arise in any general knowledge representation sooner or later, whether it is based on semantic nets, frames, scripts, production systems, or anything else. These nominally disparate formalisms have

much in common and are in the process of converging further; for example, all incorporate a predicate-logic-like propositional language, all provide ways of 'clustering' information so that the information brought to bear on a given task at a given time can be sharply limited, and all have (or are to be furnished with) property inheritance mechanisms.

9.2. Recognizing type relationships

We believe that by steering our inference algorithm along 'vertical' paths in type hierarchies and keeping it topically focused, we have done about as much as can be done to ease the computational burden of any general inference algorithm. By radically limiting the set of propositions allowed into the reasoning mill at any time, our strategy helps to prevent the combinatorial explosions which are apt to bring the mill to a halt.

This is not enough for fast question answering, however. For, if all possible derivations of the answer to a question are long, then any general reasoning strategy will probably do a great deal of searching before finding one, even when working with a small set of propositions.

It turns out that standard deductive derivations of the answers to many simple questions are indeed rather long. Consider the question

Whom does Mary love?

and assume that its logical form[20] is

?∃x[[x person] & [Mary loves x]].

The desired answer is the set of persons Mary is known to love. Now suppose that the system finds, by retrieval of information about Mary under the topic "emotional attitudes" that Mary loves John, and also that she loves her prize orchid

[20] We use predicate infix form as an aid to readability, with the predicate symbol following its first argument and followed by the remaining arguments, if any.

plant. It remains to confirm that John is a person (and therefore a suitable answer) while the orchid is not. The former subproblem is not too taxing, assuming that the system has the facts

[John boy],

$\forall x$ [[x boy] $\rightarrow$ [x person]]

at its disposal, and these are selected as relevant. Showing that the orchid is not a person, though, is harder than it ought to be. The following sort of inference chain is required (where o is the beloved orchid):

1. [o orchid]	known
2. $\forall$x[[x orchid] $\rightarrow$ [x soft-stemmed-plant]]	known
3. [o soft-stemmed-plant]	from 1,2
4. $\forall$x[[x soft-stemmed-plant] $\rightarrow$ [x plant]]	known
5. [o plant]	from 3,4
6. $\forall$x[[x person] $\rightarrow$ [x creature]]	known
7. $\forall$x[[x creature] $\rightarrow \neg$[x plant]]	known
8. $\forall$x[[x person] $\rightarrow \neg$[x plant]]	from 6,7
9. $\neg$[o person]	from 5,8

This is not a worst-case example; if, for example, Mary also loves her piano, more steps will be required to rule it out as a candidate answer, assuming that "creature" leads upward to "living-thing" in the taxonomy of types, while "piano" leads to "musical-instrument", hence to "artifact", and hence to "nonliving-thing", known to preclude "living-thing".

Subproblems of this kind arise constantly in question-answering processes and, without special handling, can absorb large computational resources. Yet one feels that subproblems such as establishing the non-personhood of an orchid should not detain the reasoning system significantly.

In essence, we wish to be able to perform type compatibility checks for pairs of type concepts quickly. For example, we should be able to confirm the truth of [o plant] or the falsity of $\neg$[o creature] instantly once [o orchid] has been stored. In other words, we should be able to evaluate certain literals quickly. Similarly, we should be able to 'resolve' the pair of formulas

[o orchid]

```
¬[x person] v [x creature]                    (= 6, in clause form)
```

directly to obtain ¬[o person]; likewise, we should be be able to 'resolve' the pair of formulas

```
[o orchid]
¬[x creature] v ¬[x plant]                    (= 7, in clause form)
```

directly to obtain ¬[o creature]. Note that the first example of generalized resolving is based on the incompatibility of "being an orchid" and "being a creature", while the second is based on the incompatibility of "being an orchid" and "not being a plant", that is, the subordination of "being an orchid" by "being a plant".

With such methods, proofs such as 1-9 above could be 'short-circuited', reducing them to a single step. The potential usefulness of evaluation and generalized resolving is apparent in all of the special domains we consider. This has been a recurrent theme in the knowledge representation and deduction literature (for example, (Papalaskaris and Schubert, 1982), (Brachman, Fikes, and Levesque, 1983), (Stickel, 1983)); much the same idea has motivated the design of sortal logics for AI purposes (for example, (McSkimmin and Minker, 1979), (Walther, 1983), and (Stickel, 1983) offers a unified view of evaluation and generalized resolving, under the rubric "theory resolution" (see section 9.6)).

An obvious method for determining type relations is to pursue upward paths in the type graph -- which we assume to be set apart from other information in the net in any case -- until the paths intersect. If the point of intersection coincides with the point of origin of one of the paths, then the concept at that point of origin is superordinate to the other; (for example, paths from "girl" and "creature" will intersect at "creature", so that "creature" is superordinate to "girl"). In all other cases the concepts are incompatible. (Or are they?...see below). This idea can be implemented as graph algorithms (for example, (Fahlman, 1979)) or as special theorem proving strategies (for example, (Bundy, Byrd, and Mellish, 1982), (Tenenberg, 1985), (Rich, 1985)).

Two comments are in order about such methods. First, it is not always clear in graphical approaches what the intended

interpretation of type graphs is. In particular, it is often unclear whether concepts lying on divergent branches of a graph are to be regarded as incompatible. For example, if there are separate arcs running upward from "plant" and "creature" to "living-thing", are we entitled to conclude that no plant is a creature? If yes, then by the same token, if there are arcs running upward from "novelist" and "poet" to "writer", can we conclude that no novelist is a poet? Moreover, the subcategories of a concept node may or may not be regarded as jointly exhaustive, and this affects what may be inferred. For example, if a microbe is known to be a living thing but not a plant, then the conclusion that a microbe is a creature is warranted just in case the subcategorization of living things into plants and creatures is interpreted as being exhaustive. Such issues can be clarified by relating type graphs to standard logical representations.

The second comment is that step-by-step tracing of superordination relationships may be a rather clumsy solution to our problem, in that constant-time methods may be possible. For example, by precomputing the transitive closure of an acyclic type graph, we could achieve constant-time testing of subtype-supertype relationships (although we would rather avoid the quadratic storage costs this method can entail). Our own method, though closely related to path intersection methods, entirely avoids path traversals and (under certain assumptions) determines type relationships in constant time.

Type graphs specifying arbitrary subordination chains and incompatibilities can be defined quite conveniently in terms of a single partitioning relation § such that

$$[T \S T_1 \ldots T_k]$$

means that type concept T is partitioned into the k mutually incompatible, jointly exhaustive subtypes $T_1,\ldots,T_k$. The possibility that a partitioning is non-exhaustive can be taken care of by introducing a remainder type T_k denoting all things of type T not covered by $T_1,\ldots,T_{k-1}$. T_k may happen to

denote the null concept (= universally false predicate).[21] When subtypes are not considered incompatible, this can be expressed by means of multiple partitioning assertions. For example, the following says that novelists and poets are necessarily writers, without making any commitment about possible incompatibility:

[writer § novelist T], [writer § poet T'],

where T and T' are the remainder categories (T = λx [[x writer] & ¬[x novelist]], and similarly for T'). Sets of such partitioning assertions can be drawn as graphs, as illustrated in Figure 9-1 (discussed more fully below). The named nodes represent type concepts, while each T-shaped connection from a superordinate node to a set of subordinate nodes represents a §-assertion.

The graph in Figure 9-1 is a hierarchy, having a unique root (viz., the most general type of concept covered by the hierarchy) and just one partitioning assertion subdividing each of its nonterminal nodes.

In general, a given set of §-assertions certainly need not form a hierarchy, but may define an arbitrarily complex graph. It would be gratifying indeed to have an efficient algorithm for testing concept compatibility and subordination in this general case (where an efficient method is one which requires linear storage and at worst linear time relative to the number of nodes and §-assertions of a graph). Unfortunately, the prospect of finding such an algorithm is very slim, since doing so would require solving the famous unsolved problem "P=NP?" affirmatively (Schubert, 1979).

Accordingly we have sought methods which, though limited in theory, work well for the kinds of taxonomies we are actually able to construct. In particular, the following is a simple type checking scheme we have implemented. The

---

[21] We would like to interpret types related by § as necessarily incompatible, necessarily subordinate to the head type, but not necessarily jointly exhaustive (only as a matter of fact). Thus [ape § gibbon orangutan chimpanzee gorilla] is true, even though there are conceivable types of apes coinciding with none of the four actual types. However, we will not be concerned with modal logic herein.

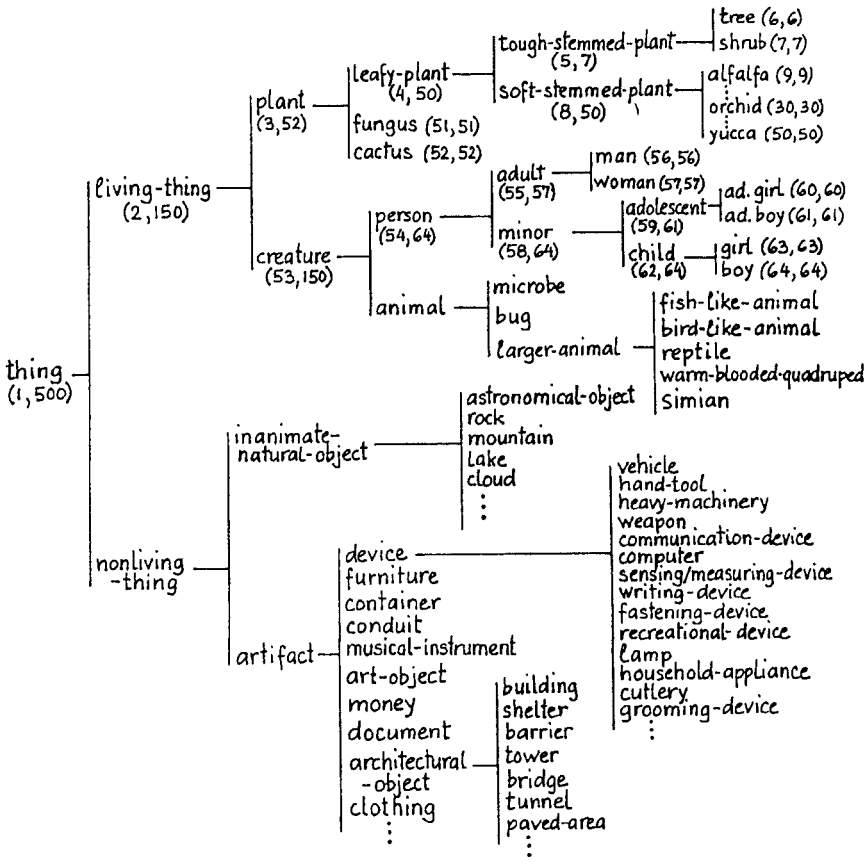


Figure 9-1: A Partitioning Hierarchy for Physical Objects.

Each vertical bar corresponds to a partitioning of a type of object into subtypes; for example, [living-thing § plant creature]. Some of the 'number brackets' based on preorder numbering of the hierarchy are shown. These allow determination of subordination and exclusion relationships in constant time. For example, "orchid" is subordinate to "plant" since (30,30) is included in (3,52), and incompatible with "person" since (30,30) and (54,64) are disjoint.

partitioning graph for types is first decomposed into a set of partitioning hierarchies. The hierarchies are allowed to intersect, but each type concept is assumed to participate in just a few of them. This assumption is based on attempts to sketch

taxonomies of physical objects, virtual objects (shadows, rainbows, reflections, ...), regions (borders, interiors, holes, ...), substances, events, perceptual/conceptual entities (thoughts, fears, pains, ...), symbolic/linguistic entities (words, musical scores, equations, ...), socio-political entities (families, committees, nations, ...), and a few other 'ontological categories'. In the case of physical objects (Figure 9-1), we can see no reason for having more than one major hierarchy, but in the case of substances, for example, we can construct two alternative, equally natural hierarchies, based respectively on naively scientific criteria (for example, plant substance versus animal substance), and on the 'normal states' of substances (for example, solid versus fluid) (Figure 9-2). The hierarchies intersect at the level of specific substances (leaf nodes), although we have drawn them separately and omitted much of their lower-level structure for the sake of clarity. It is perhaps possible to classify substances in still another way, namely according to their normal role or use (for example, foods, solvents, building materials, explosives, etc.), but beyond that, there seem to be few plausible alternatives.

Each concept has attached to it a short list of hierarchy indicators, where each hierarchy indicator consists of an identifier for a hierarchy to which the concept belongs along with the concept's 'number bracket' relative to that hierarchy. The number bracket consists of the preorder number of the node and the maximal preorder number among its descendants in the hierarchy (refer again to Figure 9-1). It is easy to show that if one node is an ancestor of another, its number bracket (regarded as an interval) contains that of the other. If neither is an ancestor of the other, the number brackets are disjoint.[22]

---

[22]See, for example, (Aho, Hopcroft, and Ullman, 1983) for a discussion of preorder traversal of trees. An alternative scheme is to number leaf nodes left-to-right, and to label non-leaf nodes with the minimal and maximal leaf numbers they dominate. The advantage of preorder numbering is that each terminal and non-terminal node is indexable by a single number. For sufficiently small hierarchies with bounded fanout one could also use bit string representations of nodes (for example, x0000, x0001, x0010, ... for the successors of a node x), with the advantage that lowest common ancestors could be easily determined.

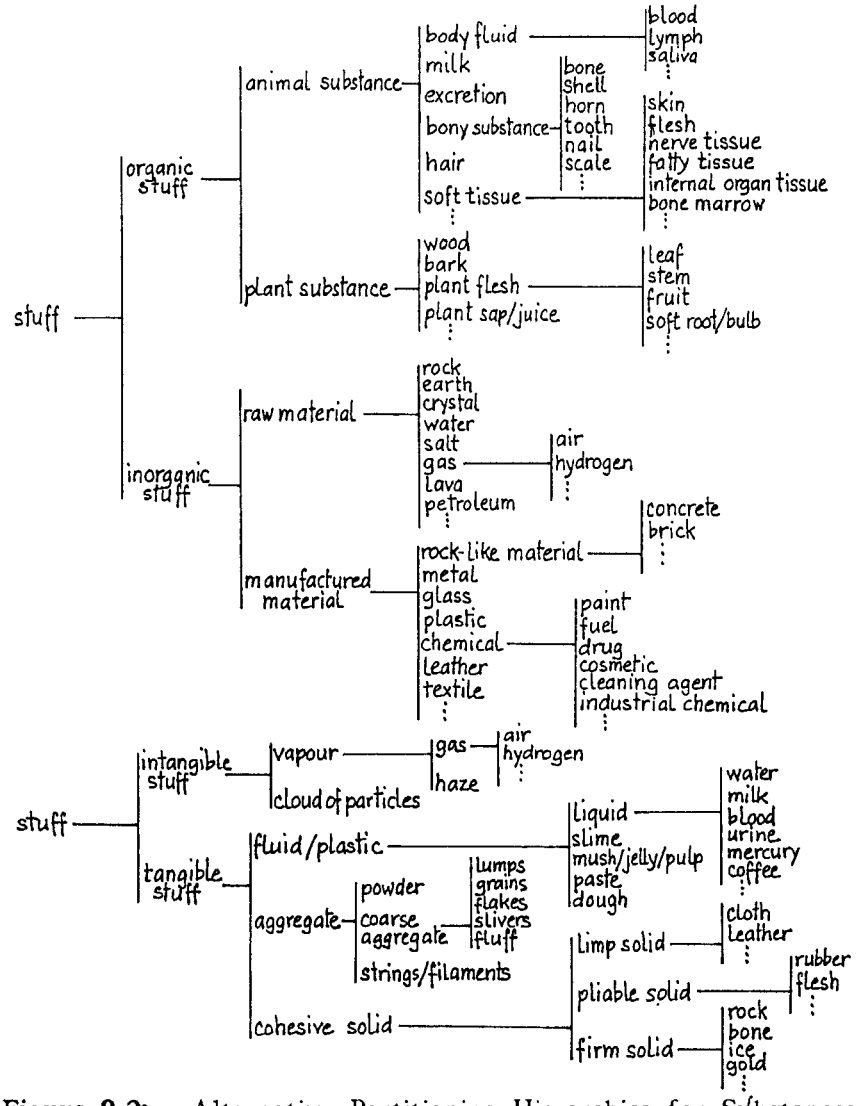


Figure 9-2: Alternative Partitioning Hierarchies for Substances.

The first is based on 'naively scientific' criteria, and the second on 'normal states' of substances. A third hierarchy based on 'uses or roles' of substances could be added.

Thus, given two concepts, their relationship can be checked by scanning their lists of hierarchy indicators for a common hierarchy identifier, and if one is found, checking whether either of the corresponding number brackets includes the other. If yes, then one concept is superordinate to the other and if no, they are incompatible. If no common hierarchy is found,

the result of the check is "unknown relationship". This is a constant-time operation if lists of hierarchy indicators contain just a few elements in most cases, as we have assumed.

This algorithm is very fast, and unlike some of the more ad hoc graphical methods makes provably sound decisions. However, like those methods it is incomplete. Consider for example the "body fluid" node and the "liquid" node in the first and second hierarchies of Figure 9-2 respectively. If we imagine both hierarchies to be completed so that they terminate within a common set of leaf nodes (the lowest-level substance types), then the set of leaves reachable from "body fluid" will presumably be a subset of the set of leaves reachable from "liquid". Thus a 'smart' algorithm could pronounce "body fluid" subordinate to "liquid", whereas ours returns "unknown relationship".

In the next section we sketch such 'smart' algorithms for parts graphs (which are much like type graphs), but for type compatibility checking our present method may be adequate. The reason lies in the fact that the general inference algorithm is not dependent on the §-graph algorithm alone for its type information. For example, the fact that concept A is a special case of concept B will presumably be available as an assertion indexed under the "generalization" topic for A, even though no §-assertion links the two concepts. Such assertions allow the general inference algorithm to make up for the gaps in the graph algorithm. The point is that the special graphical methods are intended only to accelerate the general inference algorithm at the core of the system, not to supplant it. In this respect our design philosophy differs from that in systems like KL-TWO, in which a logically weak (but computationally efficient) core is augmented with specialized extensions, such as a terminological component, to meet the needs of intended applications (Vilain, 1985).

9.3. Recognizing part-of relationships

Suppose that the on-board computer of a manned spacecraft has detected a valve failure. Question: is the valve part of the life support system? In a reasoning system entirely

dependent on general rules of inference, this might be difficult to answer. For a positive answer, an inference chain which progresses from part to superordinate part may have to be constructed, and if the inference mechanism treats part-of assertions like all others, this may involve a good deal of combinatorial searching. Indeed, not only sets of superordinate assertions forming such chains need to be explored in general; for, as we shall see shortly, a part-of relation may be implicit in a collection of assertions about parts structure even though no such chain exists.

The part-of structure of an object can be represented in essentially the same way as a taxonomy of concept types. We introduce an object partitioning relation P, with

$$[x\ P\ x_1\ \ldots\ x_k]$$

expressing that object x is (exhaustively) partitioned into parts x_1, ..., x_k. If we simply want to assert that x has a part y, we can do so by writing

$$[x\ P\ y\ z],$$

where z is possibly the empty part.

Figure 9-3 shows a partial human anatomy, naively conceived, in the form of a P-graph. It subdivides the body into head, neck, trunk and limbs (enumerated separately) and also specifies a skeleton as part of the body, subdivided into skull, spine, ribcage, pelvis and the bones of the four limbs. Also, note that each division of the skeleton is linked to its appropriate body segment.[23]

The algorithms sketched for type graphs could be used here to determine the truth values of such formulas as

```
[cranium-of-John part-of head-of-John] or
[trachea-of-John part-of bowel-of-John],
```

assuming that a sufficiently complete P-graph had previously been established for John. (The assumption would be more

---

[23]The figure glosses over some logical niceties concerning the interpretation of generic nodes such as "pelvis" and "left-leg".

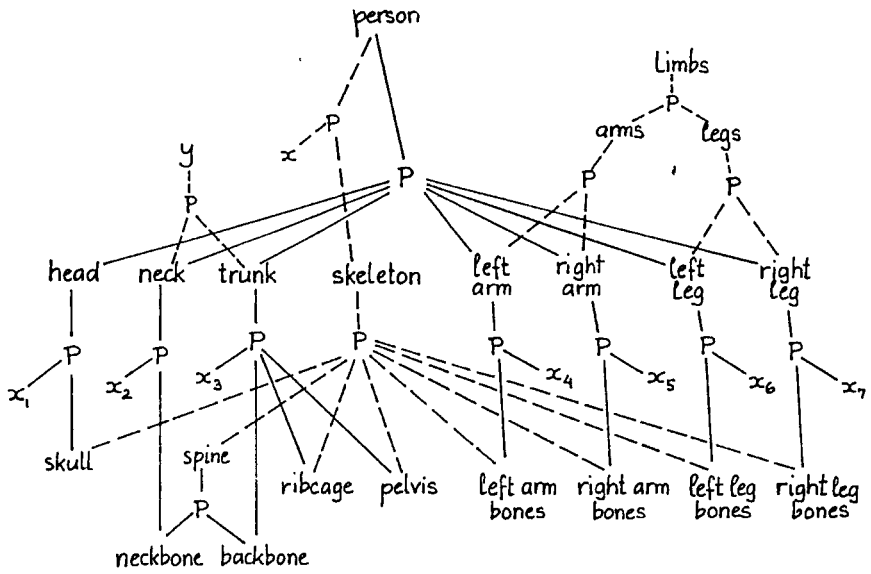


Figure 9-3: The Upper Levels of a Partitioning Graph for the Human Body.

Each P-token represents a partitioning assertion dividing the node to which it is linked above into the nodes to which it is linked below. The solid lines define a partitioning hierarchy, and the broken lines define three additional, superimposed hierarchies.

plausible for an object whose structure is unique, such as a country. For uniformly structured objects like people the algorithms could serve to evaluate more general formulas, such as

```
[(s x) part-of x]
```

where s is a function which, for any person x, picks out x's

skull, that is.

```
∀x[[x person] => [(s x) skull-of x]].)
```

Similarly the algorithms could be used to detect the incompatibility of pairs of formulas such as

```
[x pelvis-of John]  and
[x left-leg-of John].
```

Additional types of generalized resolving facilitated by partitioning graphs are mentioned in section 9.6.

This would short-circuit many proofs, but would also fail to short-circuit many others. Consider, for example, the question "Is the spine part of y?", where y is the combination of trunk and neck, as specified in the graph. This question is similar to the question about the malfunctioning spacecraft valve, and poses just the sort of problem we alluded to. The response would be "unknown", since "spine" and y do not lie in a common hierarchy; yet the graph certainly allows the inference that the spine is part of y, since there is a P-assertion which divides the spine exhaustively into neckbone (cervical vertebrae) and backbone, and these have upward paths to y. Similarly, the question may be asked whether the spine is part of the limbs, and again the response would be "unknown" since "spine" and "limbs" lie in no common hierarchy; yet a negative answer can be deduced from the graph, as the reader can verify.

A realistic human anatomy would, of course, be far more complicated, particularly if it delineated not only structurally or geometrically well-defined parts, but also the functionally cohesive subsystems, such as the digestive system, cardiovascular system, nervous system, and so on. The interplay of structural and functional views would lead to further 'tangling' of subhierarchies of the sort already evident in Figure 9-3.

The complexity of parts graphs in comparison with type graphs has led us to seek more powerful methods for them. In view of the "P=NP?" obstacle to the discovery of efficient methods for unrestricted graphs, we have sought to define classes of graphs allowing greater structural freedom than strict

hierarchies, yet amenable to fast, complete inference. One such class is the class of closed graphs. Roughly, a closed graph consists of at least one main hierarchy, along with any number of additional hierarchies such that all downward paths from nodes of these hierarchies terminate at leaves of the main hierarchy. Nodes are subdivided into those which are known to denote nonempty parts and those which are potentially empty 'remainder' parts. (It is the possibility of empty parts which makes the generalization from hierarchies to closed graphs nontrivial.)

The graph of Figure 9-3 very nearly satisfies the requirements for closed graphs. The solid lines define the main hierarchy, while the broken lines define additional superimposed hierarchies. Note that all downward paths terminate at leaves of the main hierarchy, save one: the (null) path from x ends at a leaf node, namely x, not belonging to the main hierarchy. Since x intuitively represents the soft tissue of the body (that is, body minus skeleton) and x1 represents the soft tissue of the head, x2 the soft tissue of the neck, and so on, the graph could easily be closed by adding the P-assertion

```
[x P x1 x2...x7].
```

Closed P-graphs appear to provide much of the flexibility required for representing part-of structures, yet permit reasonably efficient, complete inference of part-of and disjointness relationships. For nodes which do not lie in the main hierarchy, the inference algorithms work by 'projecting' these nodes into the main hierarchy (or some other common hierarchy). For example, in Figure 9-3 the projection of "spine" into the main hierarchy is the set of nodes S = {neckbone, backbone}, for y it is Y = {neck, trunk}, and for "limbs" it is L = {left arm, right arm, left leg, right leg}. The algorithm for checking "part-of" would conclude that "spine" is part of y since all members of S have ancestors in Y. The algorithm for checking disjointness would conclude that "spine" is disjoint from "limbs", since the members of S have no ancestors in L, and vice versa (for details see (Schubert, 1979)).

While these methods require linear time in the worst case, it is clear that nearly constant expected time is assured if the graph can be decomposed into hierarchies such that no node

belongs to more than a few hierarchies and the nodes being compared usually belong to a common hierarchy. Under these conditions separate preorder numbering of the component hierarchies can be used much as in the case of 'type hierarchies; the main refinement is that projection into the main hierarchy (or into some other common hierarchy) is tried as a last-ditch strategy before an "unknown" response is given.

The restrictions on P-graphs can be relaxed still further without running into the "P=NP?" problem. In particular, we can define a semi-closed P-graph as one which is either a closed P-graph, or a semi-closed P-graph with another semi-closed P-graph attached to it by one of its main roots. Intuitively, such graphs allow for 'entirely unrelated' partitionings of the same entity. Complete and reasonably efficient inference algorithms for such graphs are given in (Papalaskaris and Schubert, 1981), and proved correct in (Papalaskaris, 1982).

9.4. Recognizing colour relationships

Imagine a witness to a bank robbery being questioned about the colour of the get-away car. His impression was that the car was tan, and he is asked "Was the car brown?". Clearly the answer should be affirmative (for example, "Yes, tan"), and this answer could easily be deduced from

10. [c tan],
11. ∀x[[x tan] => [x brown]].

If the question had instead been "Was the car maroon?", a negative answer could have been inferred from 10, 11,

12. ∀x[[x maroon] => [x red]], and
13. ∀x[[x red] => ~[x brown]]

in four proof steps.

These examples follow the pattern of the type and part-of inferences exactly, and suggest that some sort of colour hierarchy or graph should be used to eliminate searching. In fact, the 11 basic colour terms of English could be introduced via the type partitioning

14. [coloured § red orange yellow green blue purple pink white black grey brown],[24]

and 11 could be reformulated as something like

15. [brown § tan rust midbrown chocolate ...]$_t$,

and similarly for 12, allowing either of the above questions to be answered by simple hierarchy methods.

However, a series of complications has led us away from graphical methods towards geometric methods. First, partitionings like 15 are inaccurate since shades like tan, midbrown, and chocolate probably overlap. More accurate characterizations require partitioning these shades into overlap and non-overlap parts. Shades like turquoise and lime, which straddle boundaries between basic colours would also have to be subdivided, adding to the proliferation of partitionings. Second, when we attempted to deal with 'hedged' colour relations, such as the statement that lime is sort of yellow and also sort of green, we realized that the colour partitioning graph would at least have to be augmented with adjacency and/or apart-from relations (see below). But even these additions would leave us totally unequipped to deal with other kinds of colour properties and relationships, such as lightness, purity, saturation, complementarity, and the warm/cool distinction. Geometric representations, on the other hand, offered a handle on all of these problems. If colours could be represented as simple regions in some colour space, all their properties and relationships could be 'read off' their parametric representations.

With this objective in mind, we undertook a search for a structurally simple and theoretically complete colour space. We were at first drawn to representations of colours in terms of

---

[24]We regard it as a reasonable claim that every (uniform) colour is at least a marginal instance of one of these basic colours. The claim that no colour instantiates more than one of the basic colours can be defended as well. Suffice it to say that a question answering system could well hold 14 (and in particular the disjointness of colours 14 entails) to be true, yet avoid applying the basic colour terms to their marginal cases, for pragmatic reasons.

three 'orthogonal' primaries, partly because the human visual system employs three kinds of receptors selectively sensitive to different wavelengths (though their frequency response is rather broadband -- see (Kay, 1981)), and partly because a colour cube can be contrived so that it has some very pleasing regularities. (Our version had the six basic colours of the rainbow, along with black and white, at its eight corners.) However, colour cubes are deficient in two respects. First, they are theoretically incapable of representing all perceptually distinct shades of colour (see (Judd and Wyszecki, 1963); this may seem surprising, in view of the wide commercial use of three-colour schemes). In addition, the regions corresponding to the English colour terms are rather complex, obliquely bounded polyhedrons, making region comparison computationally awkward.

Our ultimate choice was a cylindrical representation, arrived at by imagining any colour to be composed of some amount of a pure, monochromatic colour, plus certain amounts of black and white. Thus one dimension runs through the continuum of rainbow hues, arranged in a circle and arbitrarily scaled from 0 to 12; the second (radial) dimension parameterizes the amount of black present as

$$\text{purity} = \text{pure colour}/(\text{pure colour} + \text{black}),$$

which decreases from 1 to 0 as black is added; and the third (axial) dimension parametrizes the amount of white present as

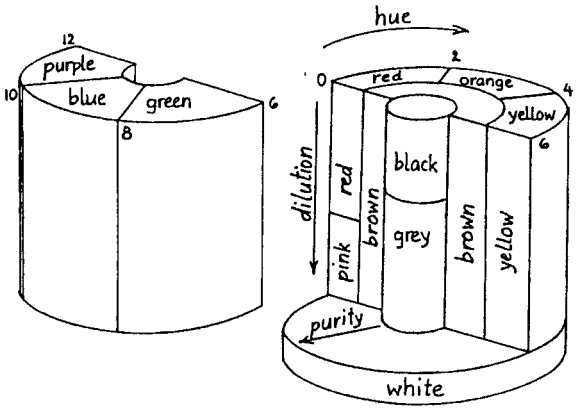
$$\text{dilution} = \text{white}/(\text{pure colour} + \text{black} + \text{white}),$$

which increases from 0 to 1 as white is added (see Figure 9-4).

This model is similar to certain models well-known to colour theorists, (Birren, 1969a), (Birren, 1969b), and like them covers the full range of perceptible shades.[25] It appears to be unique, however, in that it renders each English colour term simply as a region bounded by six coordinate surfaces (defined by three pairs of upper and lower bounds on hue, purity and

---

[25]The 'saturation' and 'lightness' parameters used in these models do not coincide with purity and dilution.



	hue	purity	dilution
white	(0 12)	(0 1)	(9 1)
grey	(0 12)	(0 .2)	(.4 .9)
black	(0 12)	(0 .2)	(0 .4)
brown	(0 6)	(.2 .6)	(0 .9)
pink	(0 2)	(.6 1)	(.5 .9)
red	(0 2)	(.6 1)	(0 .5)
orange	(2 4)	(.6 1)	(0 .9)
yellow	(4 6)	(.6 1)	(0 .9)
green	(6 8)	(.2 1)	(0 .9)
blue	(8 10)	(.2 1)	(0 .9)
purple	(10 12)	(.2 1)	(0 .9)

Figure 9-4: The eleven basic colours in a colour space.

The eleven basic colour in a (hue, purity, dilution) colour space (with the cool shades 'lifted away'). Purity decreases as black is added to a pure colour, and dilution increases as white is added to it.

```
Purity = pure colour/(pure colour + black),
Dilution = white/(pure colour + black + white).
```

The numerical values have been chosen on purely intuitive grounds. They could be quite drastically altered without affecting the results of the algorithms based on the model, as long as the region adjacency relationships are not changed.

dilution). In the other models we are aware of, the boundaries are more irregular.[26]

---

[26]Some adjustments to the simple regions of Figure 9-4 may be required. Perhaps part of the "brown" region separating "red" from "black" should be maroon or purple. We have attempted to investigate this point empirically using the colour graphics of a Jupiter terminal, but the range of available colours, impressive as it seemed at first, was insufficiently subtle to resolve the issue.

With this colour geometry it is possible to check any desired relationship between pairs of colour regions, such as inclusion, overlap, adjacency, and separation (apart-from) in a small, fixed number of comparisons. Moreover, it is easy to define non-basic terms such as turquoise, maroon, beige, scarlet, and so on, as regions bounded (like basic colour regions) by coordinate surfaces. Colour properties such as lightness, purity, etc., and relations such as complementarity are computable in fairly obvious ways.

Up to this point, we have tacitly assumed that the colour cylinder would contain explicit representations of all colours for which a compatibility check would ever be required. We can relax this assumption, allowing for the possibility that certain non-basic colours are understood only in terms of their qualitative relation to the basic colours. For example, turquoise might be said to be greenish-blue, or what amounts to roughly the same thing, both sort of blue and sort of green.

This calls for extension of our special methods, as is evident from the following variant of the 'robbery' example. Suppose the witness recalls the colour of the car as lime, and the questioner asks whether its colour was turquoise. The incompatibility of these descriptions is clear (though of course the accounts of different eye-witnesses can easily differ to this extent). Let us first see how the incompatibility could be detected by unaided deductive inference, the only knowledge about "lime" being that it is sort of yellow and sort of green and about "turquoise" that it is sort of green and sort of blue. The key to the proof of incompatibility is that nothing can be both sort of yellow and sort of blue, because yellow and blue are 'apart-from' each other (being separated by green):

```
16. [c lime]                                          known
17. ∀x[[x lime] => [x (sort-of yellow)]]              known
18. [c (sort-of yellow)]                          from 16,17
19. [c turquoise]                                 hypothesis
20. ∀x[[x turquoise] => [x (sort-of blue)]]           known
21. [c (sort-of blue)]                            from 19,20
22. [yellow apart-from blue]                          known
23. ∀AB[∃x[[x (sort-of A)] & [x (sort-of B)]]
                    => ~[A apart-from B]]             known
24. ∀x[~[x (sort-of yellow)] v
                    ~[x (sort-of blue]]           from 22,23
25. ~[c (sort-of blue)]                           from 18,24
```

26. contradiction from 21,25

The second-order features here are of no importance - they could be suppressed by treating colours as individuals related to objects via a relation "x has colour y". The 'apart-from' relation is stronger than mere incompatibility, since it entails the existence of an intervening colour which separates the two colours. To see that mere incompatibility would be insufficient, consider the problem of checking "lime" and "olive green" (instead of "lime" and "turquoise") for compatibility. "Lime", we may say, entails "sort of yellow" and "sort of green", while "olive green" entails "sort of green" and "sort of brown". But since "yellow", "green" and "brown" are mutually adjacent colours, rather than being 'apart-from' each other, we cannot infer incompatibility on the basis of these entailments. (We may still be able to infer incompatibility in some other way, for example, on the basis of relative lightness or purity.)

We would like to replace inordinately long proofs like the above by methods which directly infer a contradiction from assertions like 18 and 21. In (Papalaskaris and Schubert, 1982) and (Papalaskaris, 1982) we describe a tabular addendum to the cylinder model which allows incompatibilities between pairs of hedged or unhedged, negated or unnegated colour predicates (like those in 18 & 21) to be detected without proof. If the colour predicates are A and B (assumed to be non-equivalent), the method consists of first classifying the relation between A and B as one of "apart", "adjacent", "overlapping" (with neither colour region including the other), "centre-included" (that is, inclusion without boundary contact), or "edge-included", using the colour cylinder. Then this classification, along with the hedging and sign information and the classification of A and B as basic or non-basic is used to judge incompatibility by table look-up. For example, ~A and B are judged incompatible just in case A is basic and B is (edge- or centre-) included in A; (thus ~brown and tan are incompatible); (sort-of A) and (sort-of B) are judged incompatible just in case A and B are apart; (thus sort-of yellow and sort-of blue are incompatible); ~A and (sort-of B) are judged incompatible just in case A is basic and B is centre-included in A; (thus ~red and sort-of scarlet are incompatible);

and ~(sort-of A) and (sort-of B) are judged incompatible just in case A is basic and A and B overlap or B is included in A; (thus ~(sort-of yellow) and lime are incompatible).

These judgements can be understood by interpreting "sort-of" as an operator which expands a colour region with coordinate bounds (x_i,y_i), $i \in \{1,2,3\}$, into one with coordinate bounds (x_i-d,y_i+d), where $d = (y_i-x_i)/4$ (that is, each interval is expanded to one and one-half times its original size). Coordinate intervals for non-basic colours are assumed to be at most half as large as for basic colours. For a one-dimensional visualization of this account see (Papalaskaris, 1982).[27]

9.5. Recognizing time relationships

Did the first moonwalk by an astronaut precede the first space shuttle launch? Most people will be able to answer this question quickly and easily in the affirmative. The answer will perhaps be based on the feeling that the first moonwalk occurred many years ago, while the shuttle program only became operational in recent years. More details than that may be recalled, of course: that the Apollo 11 mission took place in 1969, and the first shuttle launch in 1981, for example, and perhaps even more specific dates. Clearly question answering systems knowledgeable about events will likewise have to be able to store and recall approximate or exact event times.

The ability to retain absolute time information is not enough, however, since people easily recall the time order of connected sequences of events even in the absence of such information. The point can be made by way of any familiar fairy-tale. Did Little Red Riding Hood meet anyone before arriving at her grandmother's cottage? The answer is "the wolf", of course. Now consider how this answer might be arrived at.

---

[27] In (Papalaskaris, 1982) d is in effect assumed to be $(y_i-x_i)/2$, so that interval sizes are doubled by "sort-of". This is probably excessive. For example, the "sort of yellow" region probably should not reach all the way to the centre of the "green" region; similarly, "sort of lime" and "sort of turquoise" probably should be "apart-from" each other, rather than adjacent.

One possibility is that the story is scanned from the beginning forward until both LRRH's encounter with the wolf and her arrival at the cottage have been retrieved, in that order. However, this does not seem very plausible psychologically, and certainly would be a clumsy strategy computationally, especially for long narratives. More likely, events which fit the pattern "LRRH encounters character x" are recalled associatively (and this is something that can be duplicated very nicely with our concept-centred, topic-oriented retrieval mechanism). Presumably, this will include not only the first encounter with the wolf, but also the fateful second encounter, as well as the ultimate encounters with the gamekeeper and with grandmother. Similarly, LRRH's arrival at the cottage would be retrieved associatively. The remaining problem is then to sort out the pre-arrival encounters from the post-arrival encounters.

Much as in the case of type, part-of, and colour inference, special methods are needed here, so that the general reasoning system will not lapse into combinatory search in determining time order. Moreover, as before we would prefer constant-time checks to linear searches of story lines. (In this respect, we would like to improve on heuristic methods such as those of (Kahn and Gorry, 1977)).

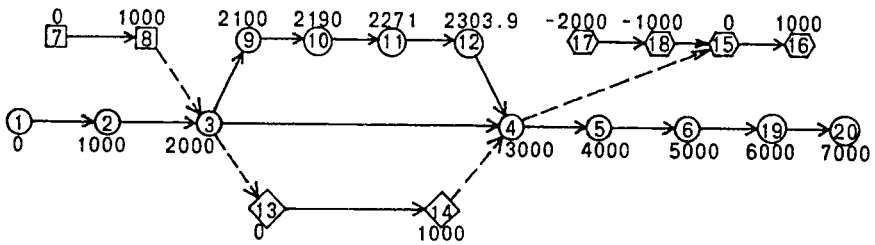
Partitioning graphs which partition time intervals, and in which the left-to-right order of subintervals is interpreted as their time order, are a possibility. We found that this representation makes a mountain out of a molehill, however. We have remarked before that the problem of extracting part-of relationships from an arbitrary § or P-graph is intractable unless P=NP. But the corresponding problem for time intervals of determining whether one interval lies within another (given positive, non-disjunctive time ordering information only) is linearly solvable. All we have to do is to represent all time intervals in terms of their beginning and end points, and insert a directed arc for each pair of time points whose order is known explicitly. The resultant graph is an acyclic digraph (except for re-entrant time travel stories), and any ordering relation implicit in it can be extracted by tracing from one point to the other, a linear operation relative to the number of edges of the graph.

Unfortunately, no methods are known for extracting ordering relationships from arbitrary acyclic digraphs in sublinear time without incurring non-linear storage cost. (However, see (Kameda, 1975) for a constant-time method for certain restricted kinds of planar acyclic digraphs.) Rather than investing effort in this research problem, which would have limited pay-off in any case given that we would also like to introduce absolute times and durations (or bounds thereon), we have proceeded pragmatically.

Roughly, the idea behind our scheme is to try to assign numeric values (pseudo-times, so to speak) to time points in their time order, when this order is known. To the extent that this is possible, the time order of two time points can be checked in constant time by comparing their pseudo-times.

Figure 9-5 (a) illustrates the kind of time graph determined by a narrative. Nodes denote time instants, and are numbered in the order of addition to the graph. Also the pseudo-times (which are incremented in steps of 1000 when not bounded above) are shown alongside the nodes. Typically narrative events correspond to pairs of time nodes, such as 1 & 2, 3 & 4, etc. Figure 9-5 (b) shows one possible sequence of event relationships which would give rise to the time graph of Figure 9-5 (a). The graph consists of a collection of time chains, each with its own pseudotime sequence. A time chain is a linear graph plus, possibly, transitive edges. Note that the link from node 3 to node 4 becomes a transitive edge of the initial time chain when nodes 9-13 are inserted. In the figure different node shapes are used for the different time chains; actually, this distinction is made by associating a metanode with each chain and maintaining a metagraph showing the interconnections between chains (shown as broken links). The metagraph for Figure 9-5 (a) is shown in Figure 9-5 (c).

As an example of the use of such a graph, suppose that the time order of nodes 7 and 16 is to be checked. Upon determination that 7 and 16 belong to different chains, the metagraph would be searched by a depth-first recursive algorithm to determine a valid path from 7 to 16 or 16 to 7. This would yield the cross-chain path which runs from the 'square' metanode to the 'round' metanode via link (8,3) and from there to the 'hexagonal' metanode via link (4,15). Since




```
e(1,2)
e(3,4) after e(1,2)
e(5,6) after e(3,4)
e(7,8) before e(3,4)
e(9,10) during e(3,4)
e(11,12) after e(9,10)
     and during e(3,4)
e(13,14) during e(3,4)
e(15,16) after e(3,4)
e(17,18) before e(15,16)
e(19,20) after e(5,6)
```

(b)

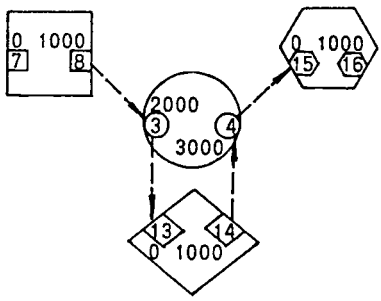


Figure 9-5: Time graph for a narrative.

(a) Time graph for a narrative. The numbers within the nodes record the narrative sequence, that is, the order in which the nodes were added. The numbers beside the nodes are pseudo-times, used for checking time order within a time chain. The four distinct node shapes distinguish the four time chains in the graph. The link from node 3 to node 4 becomes a transitive edge of the original time chain when nodes 9-12 are inserted. The pseudo-times at nodes 9-12 progress in intervals equal to one tenth of the remaining pseudo-time interval (that is, 1/10 the pseudo-time at node 4 less the last-assigned pseudo-time).

(b) Possible sequence of event relationships giving rise to the graph in (a). Note that event e(11,12) is inserted so that it occurs both after e(9,10) and during e(3,4). This is actually the most common case in story understanding, since events are usually reported one after another, but within a pre-established time frame.

(c) Metagraph for the graph in (a). The large metanodes correspond to entire time chains and are connected by the 'cross-chain' links that occur in the time graph.

the pseudotime of node 7 is less than that of node 8, and the pseudotime of node 15 is less than that of node 16, the answer

"7 before 16" can be returned.

Obviously time-checks restricted to one chain require only one comparison, while the worst-case computation time for time checks across chains is proportional to the number of chain-to-chain connections. This number is typically much smaller than the total number of links in the time graph, as far as we can tell from sample time graphs for newspaper stories several paragraphs long, a fairy-tale (Little Red Riding Hood), excerpts from Hemingway's The Old Man and the Sea, and from a book of European history. Moreover, it appears that the temporal inference problems that arise in story understanding and question answering typically involve only nodes belonging to the same chain; this is because the causal connections of interest (which correlate with time order) are usually quite direct.

We have extended the time representation and algorithms to allow for upper and lower bounds on absolute node times and on arc durations, whenever these are available (Taugher, 1983); cf. (Allen and Kautz, 1985). Bounds on absolute node times are specified as 6-tuples of the form (year month day hour minute second). For example,

$$(1984\ 10\ d\ 12\ 0\ 0) \leqslant t \leqslant (1984\ 10\ d\ 13\ 0\ 0)$$

establishes some day *d* in October 1984, 12:00 noon, as a lower bound on *t*, and 1:00 pm of the same day as an upper bound. Note that unspecified constants are permissible in time bounds. Bounds are comparable if they are identical or have identical initial segments followed by distinct numbers. Upper and lower bounds on arc durations are uniformly specified in seconds (possibly fractional).

Optimal bounds on node times and arc durations are maintained by constraint propagation; in essence, upper bounds are propagated backward and lower bounds forward. The constraints used are inequalities relating the bounds associated with pairs of nodes connected by an arc. For example, if (l_1,u_1), (l_2,u_2) are the lower and upper bounds on nodes 1 and 2 respectively, and (l,u) the lower and upper bounds on the duration of the arc from 1 to 2, then the inequalities

$$l_2 - u \leqslant t_1 \leqslant u_2 - l$$

must be satisfied (among others).

The time graphs and associated algorithms provide a basis for fast computation of a wide variety of temporal properties and relationships of events, including time order, overlap, inclusion (during), duration, exact or approximate time of occurrence, and exact or approximate elapsed time between events; all of these are easily expressed in terms of the order of time points marking beginnings and ends of events, actual time bounds on these time points, and bounds on actual time intervals separating them.

We have tested the time-order algorithm (implemented in Pascal) on a set of time relations hand-extracted from "Little Red Riding Hood". The time graph consisted of 290 time points, with 21 metanodes and 33 cross-chain links. Question answering for the ordering of random pairs of time points required 30 milliseconds of CPU time on the average on a VAX 11/780. Randomly generated graphs gave very similar results, and showed the expected linear dependence on the size of the metagraph. (Details are provided in (Taugher and Schubert, 1986)).

It is interesting to compare our approach with that of Allen (Allen, 1983). Allen's interval-based representation is somewhat more flexible -- not because it is interval-based, but rather because it admits certain kinds of disjunctions, such as

e before or after e',

not expressible as conjunctions of time point relations. However, as Vilain & Kautz (Vilain and Kautz, 1986) show, the price paid for this extra flexibility is NP-hardness of temporal inference. Vilain & Kautz also note that when only a conjunction of relations using $<$, $\leqslant$, $=$, $\neq$, $\geqslant$, and $>$ over a set of n time points is allowed, determining the consistency of the conjunction is an $O(n^3)$ time, $O(n^2)$ space operation.

We find that these bounds can be reduced to $O(n^2)$ and $O(n)$ respectively when n is taken to be the number of relationships in the given conjunction, rather than the number of time points. Moreover, our own algorithms allow us to check the consistency of a conjunction of n relations based on $\leqslant$ in $O(mn)$ time and $O(n)$ space, by building up a time graph progressively, with a resultant metagraph of size m. For

each relation, we check whether the current graph returns "yes", "no", or "unknown"; in the first case, the relation is redundant, in the second it is inconsistent with those already present, and in the third it can be consistently added to the graph. As m may be quite small compared to n, our method can be significantly faster.[28]

9.6. Combining general and special methods

In motivating the special methods proposed above, we mentioned literal evaluation and generalized resolution as two ways in which special methods can be used to accelerate a general deduction algorithm. We will now spell out this interaction in a little more detail for a resolution-based system (like ours), and comment on generalized factoring, subsumption, and tautology elimination in such a setting.

As we have remarked, literal evaluation and generalized resolution are special cases of what Stickel (Stickel, 1983), (Stickel, 1985) calls "theory resolution". The idea in theory resolution is the following. Suppose that clauses c_1, c_2, ..., c_n respectively contain non-null subclauses c_1', c_2', ..., c_n' which are (collectively) inconsistent with some separately assumed theory (for example, a taxonomic theory consisting of assertions about the relationships between types). Then we can infer $(c_1-c_1') \vee (c_2-c_2') \vee ... \vee (c_n-c_n')$. Actually, this describes total theory resolution; a more general version still, called partial theory resolution, allows unit clauses to be added to the clauses c_1', ..., c_n' to achieve inconsistency, the disjunction of whose negations must then be added to the resolvent as a residue. When the c_i' are unit clauses, theory resolution is said to be narrow.

Evaluation of a literal with result "false" can thus be viewed as narrow theory resolution with n=1, and generalized

---

[28] The fact that Vilain & Kautz permit additional relations (of which all but $\leq$ and $\neq$ are redundant) turns out to make no difference to the asymptotic upper bounds.

resolving as narrow theory resolution with n=2. It is clear that from this very general perspective, there can in principle be arbitrarily many special methods for any nontrivial domain, geared towards recognition and rapid elimination of arbitrarily complex sets of clauses c_1', ..., c_n'. (For any nontrivial theory there are, after all, arbitrarily complex sets of statements inconsistent with it.) Thus we cannot expect to provide an exhaustive enumeration of deductive shortcuts applicable in taxonomic, colour, or temporal reasoning, even if we confine ourselves to narrow theory resolution with $n \leqslant 2$. What we can do is to list more systematically the kinds of generalized resolving that 'fall out' naturally from the specialized representations we have proposed.

In connection with concept taxonomies, we illustrated two variant forms of generalized resolution. One depended on the incompatibility of two atoms (that is, predicates plus arguments) and the other on subordination of one atom by another (and hence incompatibility of the subordinate atom with the negation of the superordinate one). Incompatibility and subordination of atoms likewise are the key to generalized resolution in the other special domains, as well as to generalized factoring, subsumption testing, and tautology elimination.

In the case of atoms involving 2-place predicates such as "part-of", "skull-of" or "before", there are several ways in which incompatibility or subordination can arise. We can generally classify these ways as 'direct' and 'indirect', but the details depend on the particular predicates involved.

First, atoms can be 'directly' incompatible, or in a relation of subordination, as a result of their predicates being incompatible or in a relation of subordination. Examples are

[x skull-of y], [x spine-of y],

and

[x part-of y], [x skull-of y].

Such cases are analogous to examples involving 1-place predicates, such as "person" and "orchid", or "plant" and "orchid". Generalized resolution in such cases requires that both arguments of one atom be unified with the arguments of the

other.

But in addition, incompatibility or subordination can arise 'implicitly' as a result of relationships between the arguments occurring in atoms. For example,

[x part-of c], [x part-of c']

are incompatible (despite the identity of their predicates) if x is known to be nonempty and c and c' are known to be disjoint parts (that is, if they lie on different branches of a parts hierarchy). Note that only the first arguments need to be unified in this case. Generalized resolving yields the residue [x empty], which reduces to the null clause if x is known to be nonempty. For the same pair of atoms, the first subordinates the second if c' is known to be part of c (that is, if c' is a descendant of c in a parts hierarchy). In this case the negation of the first atom 'resolves' against the second atom to yield the null clause.

Another case of indirect incompatibility is illustrated by the atoms

[x part-of c], [c' part-of x],

where c and c' are known to be disjoint parts. In this case the first argument of the first atom must be unified with the second argument of the second atom, and 'resolving' yields [c' empty].

An additional case of subordination is illustrated by the atoms

[c part-of x], [c' part of x],

where c' is known to be part of c. In this case the second arguments need to be unified, and the negation of the first atom 'resolves' against the second atom to yield the null clause.

For 2-place time relations between moments of time, there are somewhat fewer useful ways of generalizing resolution than in the case of part relations. First, the only cases of 'direct' incompatibility and subordination are trivial ones, not requiring a time graph, such as

[t $\leqslant$ t'], [t $\geqslant$ t'],

or

$$[t \leqslant t'], [t = t'].$$

In the first example, generalized resolving yields residue [t = t'], while in the second example, generalized resolving of the negation of the first atom against the second atom yields the null clause.

Second, there appear to be only 3 useful kinds of 'implicit' incompatibility or subordination. These are illustrated by the following pairs of atoms:

$$[t \leqslant c], [c' \leqslant t]$$

$$[c \leqslant t], [c' \leqslant t]$$

$$[t \leqslant c'], [t \leqslant c]$$

where in all 3 cases [c $\leqslant$ c'] is presumed known (that is, obtainable from the time graph). In the first example, residue t=c=c' can be inferred; in the second and third examples, the null clause is obtained from the negation of the first atom together with the second atom.

This completes our inventory of potentially useful kinds of generalized resolving facilitated by our special methods for parts and times. (About evaluation, and about generalized resolving for types and colours, we provided sufficient detail in earlier sections.) We wish to mention, finally, that just as resolving can be generalized based on recognizing incompatibility or subordination among atoms, so can factoring, subsumption testing, and tautology elimination. The following are examples of clauses and corresponding generalized factors, patterned on the above examples of generalized resolving:

```
[x animal] v [x wolf] ..................        [x animal]
¬[x person] v [x wolf] .................        ¬[x person]
¬[x skull-of y] v [x spine-of y] ...        ¬[x skull-of y]
[x part-of y] v [x skull-of y] .........        [x part-of y]
¬[x part-of c] v [x part-of c'],
  where c, c' are disjoint ..     ¬[x part-of c] v [x empty]
[c part-of x] v [c' part-of x],
  where c' is part of c ..........          [c' part-of x]
```

When c $\leqslant$ c':

$\neg[t \leqslant c] \vee [c' \leqslant t] . . \neg[t \leqslant c] \vee [t=c]$ (i.e., $[c \leqslant t]$)
$[c \leqslant t] \vee [c' \leqslant t] \ldots\ldots\ldots [c \leqslant t]$
$[t \leqslant c'] \vee [t \leqslant c] \ldots\ldots\ldots [t \leqslant c']$

Subsumed literals and tautologous disjunctive pairs of literals follow similar patterns. For example,

[x wolf] subsumes [x animal]

and

¬[x wolf] v [x animal]

is tautologous (in a generalized sense). We leave further details to the reader, but should remark that elimination of a clause one of whose literals evaluates to "true" is tantamount to subsumed clause elimination. For example, eliminating a clause containing [c animal], where c is known to be a wolf, amounts to eliminating a clause subsumed by [c wolf].

In short, then, the special methods we have described provide a basis for constant-time or near constant-time simplification and generalized resolution, factoring, subsumption testing and tautology elimination in the taxonomic, colour, and temporal domains.

9.7. Concluding remarks

We have shown that much combinatory reasoning in a question answering system can be short-circuited by the use of special graphical and geometrical methods.

The domains we have considered -- types, parts, colours and times -- do not quite exhaust those for which special methods seem essential. In particular, part-of relationships are only one aspect of the structure of physical (and other) systems, and more powerful modelling methods are needed for rapid inference of static and dynamic relationships. For example, people intuitively sense the 'faulty physics' in

He put a bunch of roses in the wine glass,

perceiving with their 'mind's eye' that the roses won't stay put (whereas violets might). A good deal has been written on whether image-like representations are psychologically real and

theoretically necessary, but that is not at issue here (see Chapter 15, this volume, for more on this). What is at issue is computational efficacy, and it seems clear that the methods of symbolic logic, though no doubt capable in principle of predicting the behaviour of physical systems, need to be supplemented with special modelling methods in order to reach conclusions within reasonable times. The various expert systems incorporating models of toy blocks, electronic circuits, weight-and-pulley assemblies and so forth will point the way, although the often complex and deformable objects of the real world (like plants, coats, and people) may require methods different from those of the popular microworlds. If sufficiently powerful 'analog' models can be developed for physical objects, these may obviate the need for parts graphs such as our P-graphs, just as the colour cylinder obviated the need for colour §-graphs.

Beyond this, we do not foresee having to devise too many more special representations, as long as we are concerned with question answering of a general nature only, and not with expert consultation (for example, on programming, mathematics, or economic forecasting). In fact, even specialized expertise may often require no more than re-deployment of spatio-temporal modelling skills. For example, expertise in symbol manipulation (as required for symbolic logic, mathematics, and programming) may well rest in part on spatio-temporal visualization, and in part on linguistic skills (parsing, pattern matching) which are of course presupposed in a question answering system.