Cancellation in a Parallel Semantic Network

Scott E. Fahlman, David S. Touretzky*, Walter van Roggen

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

ABSTRACT

Handling exceptions to general rules is a persistent and difficult problem in systems for representing knowledge. In semantic networks, this often takes the form of cancelling some item of information, such as membership in a class, that would otherwise be inherited from a higher level description in the type hierarchy. Some conceptually clean approaches to cancellation lead to great inefficiency in accessing the information in the network, and can negate much of the speed advantage that would otherwise be possible in a parallel network system such as NETL. In this paper we explore some of the interactions between cancellation and parallelism in semantic networks, and we propose a cancellation scheme that appears to be workable for NETL-like systems.

I. Introduction

Every practical scheme for representing knowledge must provide some way to represent general statements about classes of objects ("all elephants are gray") and to perform the inferences necessary for this information to be inherited by the individual members of the class. Such inheritance machinery makes it possible to represent a body of knowledge much more compactly than if every property of every individual were represented explicitly. As the size of the knowledge base grows, however, inheritance can become costly to implement; in a complex hierarchy it may require a great deal of computation and search to answer apparently simple questions. The NETL system [1] performs these searches by propagating markers in parallel through the nodes and links of a hardware semantic network.

In much of mathematics and in many games and puzzles, general statements about the members of a class can be treated as inviolable, but in most real-world domains it is necessary to make occasional exceptions to general statements: "Elephants are gray, except for royal elephants, which are white." If there were no way to cancel or override the application of a general statement that would otherwise be inherited, we would be faced with two very unattractive choices: either remove the general statement about elephants being gray and state the color individually for each elephant, or remove royal elephants from the class of elephants and describe their elephant-like properties from scratch. As we accumulate large amounts of information in our knowledge base, this dilemma is encountered more frequently; if we know hundreds or thousands of facts about the typical elephant, the probability that a given elephant will be typical in *all* of these respects becomes vanishingly small. We just happen to live in a universe in which there are regularities too important to ignore, but in which every general statement has some exceptions (including this one).

The current interest in non-monotonic logics [2] is in large part due to a realization that traditional (monotonic) logics cannot handle generalizations and default reasoning in the presence of exceptions, and that such reasoning is critically important in real-world systems. However, the majority of work in non-monotonic logic has been aimed at developing a useful, consistent, and mathematically tractable notation and the associated rules of inference. Little work has been done on how to make non-monotonic inference systems efficient enough for practical use, and even less on the possibility of performing the necessary inferences in parallel.

Most existing AI knowledge-base systems either decline to handle exceptions at all [3,4] or handle them by a system of local masking, in which more local information ("royal elephants are white") masks and supersedes any conflicting information ("elephants are gray") found at more abstract levels of the hierarchy [5,6]. This system of masking has several serious problems. First, it is often hard to determine whether the local information conflicts with the inherited information, and therefore supersedes it, or whether it is intended as additional information. Second, in a system whose type hierarchy allows multiple upward branches from a node, it may not be possible, or even meaningful, to determine which of two statements inherited along different paths is the more local. Third, the masking system works only for determining the properties of a given subtype or individual; it does not lend itself to the complementary problem of finding those individuals with a certain property. Finally, the masking system works only for properties; it does not extend naturally to the situation where inherited membership in a class is to be cancelled. These problems can be greatly reduced if the information to be superseded is explicitly cancelled.

The NETL system [1] originally included a scheme for handling explicit cancellations in parallel during the normal marker-propagation operations of the network. The scheme, as originally proposed, was able to handle the trivial cases that were considered at the time, but it began to break down as more complex situations were considered. After a considerable period

---

*Fannie and John Hertz Foundation fellow.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

of wrestling with the problem, we developed some new insights into the nature of cancellation and a better system for handling cancellation in NETL. We believe that these insights and methods will be of value to other workers in the area of knowledge representation, even those working on inference systems that are not designed for parallel implementation.

In the discussion that follows, we confine our remarks to the case where the information being cancelled is the inherited membership of some individual in a class or the inclusion of some class in a more abstract and general class. This is the hardest sort of cancellation to handle: it exhibits all of the problems that we have found in cancelling inherited properties, plus some more. Also, it subsumes the other cases, since in a system like NETL one can always transform a predicate like GRAY or LIKES-PEANUTS into a class like GRAY-OBJECTS or PEANUT-LIKERS. The real advantage of limiting the discussion to class membership cancellation, however, is that we can describe our approach purely in terms of IS-A links and the various sorts of CANCEL links and avoid a discussion of roles, mapping, and quantification in semantic networks -- a complex set of topics beyond the scope of this paper.

II. Some Design Considerations

Before we look at the ways in which the obvious cancellation schemes fail, we must describe what we want a successful parallel scheme to do.

In designing a system like NETL, we cannot separate the design of the elements (nodes, links, flag bits) from the design of the parallel marker-passing algorithms (called *scans*) that locate information in the network; the representation and the algorithms must work together. Two scans will concern us here. The *upscan* starts with a mark on a single node and attempts to mark all of that node's superiors in the tangled type hierarchy -- that is, all of the superior type-nodes from which the starting node is to inherit properties and statements. In the absence of cancellation, this scan consists simply of propagating markers across IS-A links (these are called *VC links in [1]), in the upward direction (following the arrow) only. Because of the parallelism in the network, this operation can be done in time proportional to the length of the longest IS-A chain to be followed, regardless of how many nodes are marked due to branching in the network. Upscans are used very frequently for all sorts of access in the knowledge base, so any proposed cancellation scheme must preserve their parallel nature or pay a very heavy price in performance.

The *downscan* is used to mark all of the subtypes and individuals that lie below the starting type-node in the IS-A hierarchy; these are the nodes that inherit properties from the starting node. A downscan is like an upscan, except that the markers propagate downwards across IS-A links (against the arrows), not upwards. This scan, too, is used very frequently, as it plays an essential role in all recognition operations. Again, the time taken is independent of the number of nodes marked, and this parallelism must be preserved when cancellation is added to the system. For the system to function properly, the upscan and downscan operations must be complementary: if an upscan from node X marks node Y, a downscan from Y should mark node X, and vice versa.

In addition to working smoothly with the above scans, a cancellation scheme must allow for cancellations to be revoked. Consider the following set of assertions, which we would like our system to handle in a natural way:

1. A mollusc is a shell-bearer.
2. A cephalopod (octopus, squid, etc.) is a mollusc, but *is not* a shell-bearer.
3. A nautilus is a cephalopod, but *is* a shell-bearer.

Note also that we need the ability to add information to the knowledge base in any order. As we will see, the addition of a seemingly innocuous assertion can sometimes clash with assertions that are far away in the network. This must be detected and dealt with, perhaps by cancelling some of the conflicting information.

III. A Simple Cancellation Scheme and How It Fails

Let us now consider the membership-cancellation scheme proposed in [1] and see how it fails. This scheme uses a special CANCEL link (called *CANVC in the original work) to represent the cancellation of an inherited class membership. In Fig. 1, we see a network (actually a fragment of the complete network knowledge base) in which this CANCEL link is used. According to this network, a D is a C, but is not a B or an A because of the CANCEL link. E inherits the class memberships of D. To perform an upscan from node E in this network, proceed as follows:

1. First select an *activation marker* for marking the superiors of node E; select a second marker for cancellations. Suppose we select markers M1 and M2 for these roles. Begin by putting marker M1 on the starting node, E.
2. If any CANCEL link has M1 on the node at its tail, put an M2 on the node at its head.
3. If any IS-A link has M1 on its tail, and has neither M1 nor M2 on its head, put an M1 on the node at its head. (The effect of this is that the activation markers propagate upward, but will not enter or pass through a node with a cancellation marker.)
4. Repeat steps 2 and 3 in alternation until step 3 no longer generates any activity. If there are no nodes with both M1 and M2 markers present, stop. M1, the activation marker, is now present on all of the nodes from which the starting node, E, is supposed to inherit properties.
5. If in step 4 a node is found with both M1 and M2, the network contains a lost race: a cancellation marker has arrived at a node too late to prevent an activation marker from passing into and through that node. In such cases (rare in practice, but possible) the scan must be run a second time. Remove all M1 markers, leaving the M2 markers in place. Then put M1 on E, and run the entire scan again. (Note: when they are detected, race conditions can be eliminated by adding a redundant, more direct CANCEL link to the network. This will get the cancellation marker to the disputed node sooner, in time to stop the activation marker from getting by it.)

Fig. 2 shows the result of this scan. E inherits from D and C, but not from B and A. If we had started the upscan at node G, no cancellation would have occurred; G inherits from F, C, B, and A.

The algorithm for downscan is identical, but the direction is reversed: cancellation markers flow down CANCEL links, and activation markers flow down IS-A links. (They move in the

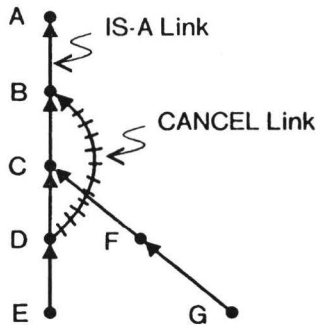

Figure 1: A Simple Cancellation

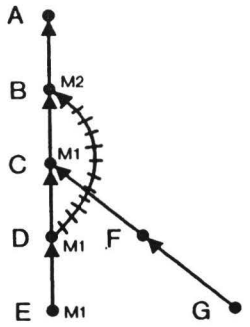

Figure 2: The Result of an Upscan from E

direction opposite to the arrows drawn on the links.) Fig. 3 shows the result of a downscan from A in the same network: B, C, F, and G inherit from A; D and E do not inherit because the CANCEL link places M2 on node D and this blocks any M1 from entering or passing through that node.

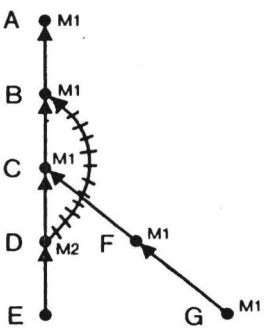

Figure 3: The Result of a Downscan from A

The inadequacy of this cancellation scheme can be seen in the example of Fig. 4. If we perform an upscan from G, we mark F, E, C, B, and A, but not node D, which is cancelled. Since the upscan says that A is a superior of G, a downscan from A should mark G. This is not the case, however; a downscan from A will mark nodes B, C, D, and E, but will not mark F and G because of the cancellation marker on F. We cannot fix this by arbitrarily

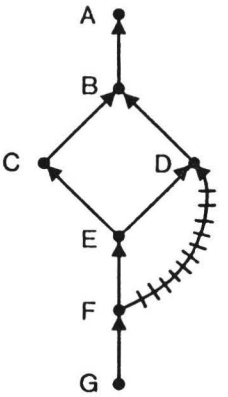

Figure 4: A Case Where Downscan Fails

deciding that downscans should not send cancellation markers downward; in many cases, such as the one in Fig. 3, these cancellation markers are essential.

A similar problem can occur in upscans. Consider the network of Fig. 5. It is clear that a downscan from G will mark F, E, C, B, and A, but not D. Using the algorithm above, an upscan from A will reach B, C, D, and E, but not F and G. In a case like this, it is not clear what *should* happen. Members of class C inherit membership in F, while members of class D explicitly do not. What about members of class A, which are members of both C and D? Should the inheritance or the cancellation take precedence? If the upscan is to be complementary to the downscan in this example, inheritance along a non-cancelled path must take precedence over the cancellation, but it is not clear how to implement that policy in a parallel scan.

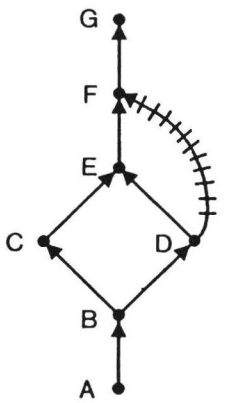

Figure 5: A Case Where Upscan Fails

IV. The Nature of the Problem

At first we viewed these problems as minor bugs in our scanning algorithms. We tried a number of fixes, such as cancelling particular IS-A links rather than cancelling class membership directly at the node, but the same kinds of problems kept appearing no matter what we tried. Gradually, it became clear to us that the problem was a fundamental one, owing more to the nature of marker-passing parallelism and non-monotonicity than to the details of our various notations and scans. Consider

again the case in Fig. 5. What we really want here is for the cancellation marker at node F to stop those activation markers that arrive by way of node D, and not to stop any activation markers that arrive via the uncancelled path through node C. In other words, we need to know not only the result that A is an F, but also how we established that fact. Is proving that A is a D an essential part of the proof that A is an F, or is there some other path available? Obviously, our single-bit markers do not carry such information.

One could imagine a parallel system in which such information is recorded and used. Each CANCEL link would "paint" a marker that passes one of its ends, and would block any marker so painted from entering the node at its other end. To prevent errors due to interactions between different CANCEL links, each CANCEL link would require its own color of paint, and a marker might have to carry many colors at once. This would solve the cancellation problem, but at a heavy cost: we are no longer passing around single-bit markers, but arbitrarily complex information structures; each parallel element of the network must now be a complex processor, capable of storing, comparing, and modifying an unbounded number of these "marker" structures. (That is, the number of colors could be as large as the number of CANCEL links in the network, which is theoretically unbounded.) In the NETL system, we have chosen to provide only a small, fixed number of single-bit markers; this results in a system which we know how to build relatively cheaply [7].

Our problems with cancellation, then, are a direct consequence of our decision to use a very simple kind of parallelism in NETL. A more complex parallel system would not exhibit these problems, but it would be orders of magnitude more costly and difficult to build.

We should note that this difficulty with cancellation is not the only problem arising from our choice of simple marker-passing parallelism. For example, to locate in parallel every known instance of a son who is hated by his own father* we need the same kind of painted markers. We can start from the father nodes and mark their sons. A second mark can be placed on all the people that fathers hate. We can then locate in parallel the set of all sons who are hated by someone's father. But without the paint, we cannot be sure that the SON-OF marker and the HATES marker came from the same father. We have produced a list of suspects, but we must now look *serially* at each suspect to see if he is, in fact, a member of the class in question. Painted markers could also be used to eliminate the copy confusion problem discussed in [1], section 3.7.

Despite these arguments, there are ways to cope with cancellation in NETL without going to more complex hardware. These involve introducing some amount of serial behavior into the operation of the network -- in essence, achieving the effect of many-colored markers by using simple markers in a number of separate scans. For example, we could send "suspicion marks" rather than cancel marks down the CANCEL links during a downscan. Every node that receives both a suspicion and an activation marker *might* be cancelled. We check this by doing an upscan from that node. Once all of these possible cancellations have been checked, and cancellation markers have been placed on the appropriate nodes, we can then do a final downscan to locate the true, uncancelled descendants of the starting node.

*This problem was put to us by Brian Smith.

Note that what was a single parallel downscan has now become a group of scans, one for each CANCEL link that the original downscan touched. In most real-world systems, a single downscan will only initiate a few CANCEL-testing scans, but in principle there is no limit, just as there was no limit on the number of colors we needed earlier. Many variations on this theme are possible, but they all result in additional scans, one for each CANCEL link that might alter the results of the scan being performed.

Another approach we can take is to perform these additional cancel-checking scans every time the network is modified, and to record the results in the network in the form of additional links. This allows the upscan and downscan algorithms to be simple parallel scans, as they were initially, but requires that a potentially large number of cancel-checking scans be run whenever any link is added to the network or removed from it. This move is advantageous, since we anticipate that in the normal operation of the knowledge base the ratio of accessing to modifying operations will be large.

A second advantage of this pre-processing scheme is that in many situations, as in the example of Fig. 5 above, it is not clear whether the user intends that an inherited cancellation should apply or should be overridden. If we locate such ambiguous situations at the time the network is modified, we can ask about how to resolve them while the knowledge-base builder is still present; if we wait until access-time to discover the problem, we might be dealing with a different user who could shed no light on the intent of the knowledge-base builder. Note that the "builder" and the "user" of this knowledge base might be programs instead of humans, but it is still the builder's job to resolve ambiguities.

This idea of checking all possibly-affected CANCEL links with individual scans at the time of any modification to the network, and of keeping the network in this pre-tested, unambiguous, "consistent" state is the basis of the scheme described below. Preserving such consistency across all possible changes can be a tricky business, especially when complications such as multiple contexts are brought into the knowledge base. We have not, at the time of this writing, dealt with many of these possible complicating factors. That is why we refer to this as a "partial" solution to the problem. However, we are reasonably confident that some combination of pre-testing and access-time testing of CANCEL links can be made to do the job.

V. A Partial Solution

We begin with the notation and the algorithm from [1] described in section 3 above. CANCEL links run from node to node and place cancellation markers during upscans and downscans. These cancellation markers prevent any activation markers from entering or passing through the nodes that they occupy. In this scheme CANCEL links are stronger than IS-A links.

In this sort of scheme, we need some way to explicitly override a CANCEL link in order to handle the mollusc-cephalopod-nautilus problem described earlier. We cannot just add an IS-A link from NAUTILUS to SHELL-BEARER, since the CANCEL link from CEPHALOPOD to SHELL-BEARER takes precedence. Instead we must use both an IS-A link and a new link whose effect is to turn off a CANCEL link for a particular subclass that would otherwise inherit the cancellation. We call this new link an UNCANCEL link. The resulting network is shown in Fig. 6.

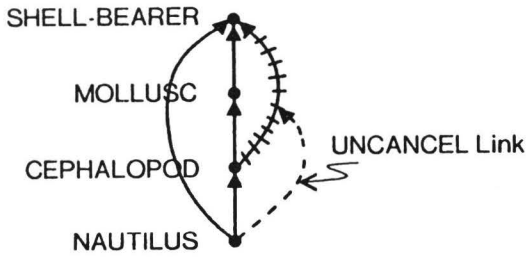

Figure 6: The Use of an UNCANCEL Link

The algorithm for upscan is unchanged, except that we select an extra marker (M3) for use by the UNCANCEL links, and step 2 is replaced by the sequence of step 2a and step 2b as shown below.

2a. If any UNCANCEL link has M1 on the node at its tail and does not have M3 on the CANCEL link at its head, put an M3 on this CANCEL link. If the CANCEL link being marked has an M2 on the node at its head, remove the M2 marker from that node.

2b. If any CANCEL link has M1 on the node at its tail *and does not have M3 on its body*, put M2 on the node at its head.

Fig. 7 shows the result of this new upscan algorithm when used to mark upward from NAUTILUS in Fig. 6. No cancellation marker is placed because the UNCANCEL link places an M3 on the CANCEL link, turning it off. A NAUTILUS is therefore a CEPHALOPOD, a MOLLUSC, and a SHELL-BEARER, according to this network.

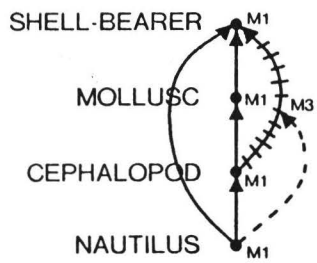

Figure 7: The Result of an Upscan from NAUTILUS

Note that the running time of this scan still depends only on the length of the longest IS-A chain, not on the branching factor or the number of nodes in the network. The addition of UNCANCEL links increases the running time of the basic upscan by a small constant factor.

Note too that CANCEL and UNCANCEL are sufficient; there is no need for UN-UNCANCEL links. If we want to create a subtype of NAUTILUS, the NAKED-NAUTILUS, that is *not* a SHELL-BEARER, we simply add this node to the network below NAUTILUS and run a new CANCEL link from it to SHELL-BEARER, as shown in Fig. 8. The old CANCEL link and its UNCANCEL are left alone.

Recall from the example of Fig. 5 that the difficulties for upscans arise in those cases in which some node lies below the tail node of a CANCEL link (node D in this case) and also lies below the head node of the cancel link (node F) by an independent path (one that does not pass through D). Node B in

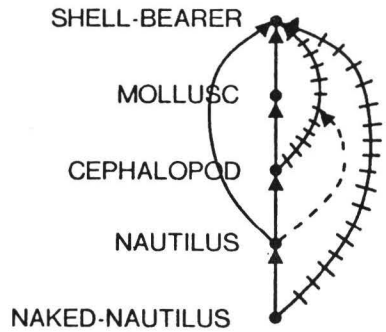

Figure 8: Re-cancelling SHELL-BEARER

this example is such a node, which we call a *merge point* below this particular CANCEL link. In this case it is unclear whether B is to be a member of class F or whether the cancellation of F is to prevail.

We propose to deal with such ambiguous cases by excluding them from the network. The structure in Fig. 5 is declared to be "inconsistent" or ill-formed. Every merge point below a given CANCEL must have an UNCANCEL link from it to the CANCEL. Therefore, the network of Fig. 6 is legal, since the merge point at NAUTILUS has an UNCANCEL link. This network will function properly for both upscans and downscans. Fig. 8 is legal as well.

Another way to make Fig. 5 legal would be to run a new CANCEL link directly from B to F. In this case, B is not a merge point, since it is not below F by the normal rules of inheritance. Thus, if we see an illegal merge point like B about to be created, we can give the user a choice: is B an F, or is it not? If it is, we insert an UNCANCEL; if it is not, we insert a new, direct CANCEL link from B to F.

Note too that by fixing the problem at node B, we also fix it for node A. In general, therefore, we need only to locate and fix the *uppermost* merge points below a given CANCEL link, not every node that lies anywhere below both the head and tail. Since the IS-A network is not a total ordering, there may be a number of uppermost merge points below a CANCEL, but we expect this number to be small.

It is not difficult to find all of the uppermost merge points below a single cancel. Call the node at the head of the CANCEL link A and the node at the tail of the CANCEL link B. The procedure is as follows:

1. With one triplet of markers, M1 for activation, M2 for cancellation, and M3 for inactivation, perform a normal downscan from node B, observing all CANCEL links as the downscan proceeds.

2. With a second triplet of markers, M4, M5, and M6, perform a downscan from node A. Because of the CANCEL link, M5 will be placed on node B, so M4 markers will not flow down through that node.

3. We must now find the *uppermost* nodes with both M1 and M4. Have every IS-A link in the network examine the node at its head. If this node contains both M1 and M4, place an M7 mark on the node at the tail of the IS-A. Every node that ends up with M1 and M4 but not M7 is an uppermost merge point: it has no merge point above it.

4. To handle some difficult cases properly, we also must find and fix any merge points that lie below the uppermost merge points found in step 3, but that have independent upward paths to both A and B. By "independent" we mean that neither of these paths go through the uppermost merge points found earlier. To find these nodes, we re-run steps 1 - 3 above, but first we put cancellation markers, M2 and M5, on all of the uppermost merge points we have found already. Repeat these scans until no new merge points are found.

So far, this section has only dealt with the difficulties encountered in upscans and what to do about them; as we saw earlier, downscans have problems as well. These problems occur in situations like those in Fig. 4 (which is just Fig. 5 turned upside-down): somewhere *above* the CANCEL link from F to D, there is a merge point, B. It is unclear whether F should be a B or not. This symmetry suggests a solution: we need a form of UNCANCEL link (called an UNEXCLUDE link) from the CANCEL to the merge point at node B. Fig. 9 shows the network of Fig. 4 with this UNEXCLUDE link added. During a downscan from node B or above, the UNEXCLUDE link places marker M3 on the CANCEL link, turning it off, just as the UNCANCEL link turned off the CANCEL link during upscans. Because of this, the downscan from B does reach nodes F and G; upscans and downscans in this network are exactly complementary to one another.

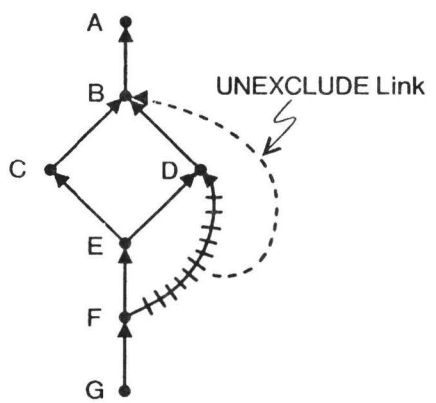

Figure 9: The Use of an UNEXCLUDE Link

In order for a network to be considered legal under this scheme, there must be an UNEXCLUDE link to every lowermost merge point above a given CANCEL link. The procedure for finding these points is exactly like the procedure described above for finding UNCANCEL points, except that the markers all go upward rather than downward. Once the upward merge points are found, we have the option of adding an UNEXCLUDE link or of adding a new, direct CANCEL link, depending on what we want the network to say. The operation of upscans and downscans, of UNCANCEL and UNEXCLUDE links, is now quite symmetrical. However, since every node must have some uncancelled path upward to the THING node, there is guaranteed to be at least one merge point above any CANCEL link, and therefore at least one UNEXCLUDE; since there is no single lowermost node in the IS-A hierarchy, many CANCEL links will have no merge points and no UNCANCEL links below them.

It seems to be easy for people to understand intuitively what the role of the UNCANCEL link is, but somewhat harder to understand what the UNEXCLUDE links are doing. The best way to visualize this is to think of the downscan as a query about which nodes are members of the class represented by the starting node of the scan. In Fig. 4, the CANCEL link excludes F and its descendents from membership in class D, and from any class superior to D. The UNEXCLUDE in Fig. 9 lifts this exclusion for class B, so that we can state that F is a B without having the CANCEL link take precedence.

We have now placed a new set of constraints on the set of legal networks: all CANCEL links must have the appropriate merge points covered by UNCANCEL or UNEXCLUDE links. If the network is legal in this sense, upscans and downscans will work properly. (In fact, there will never be a race condition, so step 5 of the scan can be eliminated.) If the network is not in this state, there are no guarantees. Therefore, we must endeavor to keep the network in this legal state. This requires that we do a certain amount of work every time a CANCEL or IS-A link is added to or removed from the network. (This is similar to the work that must be done to detect other kinds of inconsistency in NETL, such as the violation of SPLIT conditions.)

We have already described how to find the illegal merge points above and below a single CANCEL link. The act of adding an IS-A link to a network can cause some new merge points to appear for CANCEL links that are already present. In order to be sure that the network is returned to a consistent state, we must examine every CANCEL link that might be affected by this addition. This is done by performing an upscan from the top of the new IS-A and processing any CANCEL which has either end attached to one of the activated nodes; then we perform a downscan from the lower end of the new IS-A, and again process any CANCEL links that are touched by the scan. The same procedure must be performed if an IS-A link is removed, and if a CANCEL link is added or removed. Any of these operations might create new merge points. In processing the CANCEL links, we must also look for UNCANCEL or UNEXCLUDE links that no longer point to merge points. These can cause trouble later and should be removed.

All of this is a good deal of work, but as we noted earlier, we assume that the network is modified relatively infrequently. We believe that it is better to try to keep the network consistent in this way than to require extra scans to check each CANCEL link every time an upscan or downscan is performed. We expect that the number of UNCANCEL and UNEXCLUDE links will be no more than a few times the number of CANCEL links in the network, depending on how tangled the network is.

An obvious question is whether we can detect that the consistency of the network has broken down, and how to correct it. This could happen, for instance, if the demands on the network are such that the consistency checking for a change cannot be done to completion. (If we record such events, we can come back later and finish the checking.) Some inconsistencies, perhaps most of them, will result in networks with race conditions: a normal upscan or downscan will result in a cancellation marker arriving at a node which already contains an activation marker. This should never happen in a consistent network, so it can alert us to the existence of an inconsistency local to the CANCEL link which loses the race. Such races will not occur for all inconsistencies, however, so this is not an infallible indicator. In fact, the only way to be really sure that a network is consistent in this sense is to re-check all of the CANCEL links, one by one. Perhaps the system can do this in its spare time.

VI. Concluding Remarks

In summary, we have argued that cancellation is an essential operation in semantic networks, that the simple scheme of [1] is not sufficient to handle all cases, and that the roots of the problem are deep in the nature of the marker-passing style of parallelism that we use in NETL. We have presented two ways of coping with this problem within NETL: check every relevant CANCEL link during every upscan and downscan, or keep the network consistent by adding or deleting auxiliary links every time the network is modified. Either solution is costly, but perhaps not as costly as introducing a more powerful kind of parallelism into our knowledge-base system.

We do not yet have a firm theoretical understanding of the kinds of problems that can be handled efficiently by marker-passing parallelism. In this work, we have explored one small set of problems in this new domain. We believe that much more exploration of the possible kinds of parallelism and their limitations will be needed in the years to come.

REFERENCES

[1] Fahlman, S. E. *NETL: A System for Representing and Using Real-World Knowledge.* Cambridge, MA: MIT Press, 1979.

[2] McDermott, D. V. and Doyle, J. "An Introduction to Non-Monotonic Logic." In *Proc. IJCAI-79.* Tokyo, 1979, 562-567.

[3] Brachman, R. J. "Theoretical Studies in Natural Language Processing, Annual Report", Technical Report 3888, Bolt Beranek and Newman, Inc., 1978.

[4] Fikes, R. and Hendrix, G. G. "A Network-Based Representation and its Natural Deduction System." In *Proc. IJCAI-77.* Cambridge, MA, 1977, 235-246.

[5] Bobrow, D.G. and Winograd, T. "An Overview of KRL, A Knowledge Representation Language", *Cognitive Science*, 1:1, 1977.

[6] Roberts, R. B. and Goldstein, I. P. "The FRL Manual", Technical Report 409, MIT Artificial Intelligence Lab, Cambridge, MA, 1977.

[7] Fahlman, S. E. "Design Sketch for a Million-Element NETL Machine." In *Proc. AAAI Conference*, 1980, 249-252.