To be presented at the 6th International
Joint Conference on Artificial Intelligence,
Tokyo, Japan, August 20-24, 1979

REASONING IN INCOMPLETE DOMAINS

Steven Rosenberg

January 1979

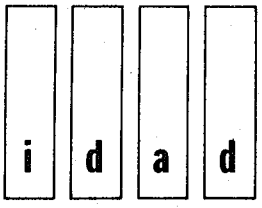
Prepared for the U. S. Department of Energy
under Contract W-7405-ENG-48

LBL-8721

REASONING IN INCOMPLETE DOMAINS

Steven Rosenberg

Information Methodology Research Project
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720



INFORMATION &
DATA
ANALYSIS
DEPARTMENT

Reasoning in Incomplete Domains

by

Steven Rosenberg

Abstract

Most real world domains differ from the micro-worlds traditionally used in A.I. in that they have an incomplete factual database which changes over time. Understanding in these domains can be thought of as the generation of plausible inferences which are able to use the facts available, and respond to changes in them. A traditional rule interpreter such as Planner can be extended to construct plausible inferences in these domains by (A) allowing assumptions to be made in applying rules, resulting in simplifications of rules which can be used in an incomplete database; (B) monitoring the antecedents and consequents of a rule so that inferences can be maintained over a changing database.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract No. 14-75-C-0643. The author was with the Massachusetts Institute of Technology Artificial Intelligence Laboratory. He is now with the Information Methodology Research Project, Lawrence Berkeley Laboratory, Berkeley, CA 94720.

In this paper I will discuss a method of rule interpretation which responds to the problem of incompleteness and instability in a domain. I will focus the discussion around the evolution of a program for inferencing. The model I shall discuss is not itself a theory of problem solving strategy, since it does not involve a commitment to a particular approach such as means-end analysis [5] or procedural nets [8]. Rather, it is an approach to the interpretation and construction of rules which allows them to be used successfully in certain types of real world domains.

Suppose that you were a farmer, and each week you must make decisions about your wheat crop based on the price you expect to get for it. You formulate the question to yourself: "Will the price of wheat rise?" Answering this question is a form of problem solving. It involves taking a stream of information (ranging from weekly reports on global demand to your own knowledge about the weather and state of your crop), and seeing whether the available facts can be organized to support the hypothesis of rising wheat prices. Most real world databases, such as our hypothetical farming one, share certain features of incompleteness and instability which make traditional reasoning processes break down. Such domains often are:

A) Incomplete, because not all the information we might need in order to make an inference is available at a particular time. For instance, during the planting season, a farmer has to decide if the price of wheat is going to rise, when only partial information concerning the supply of, and demand for, wheat is available.

B) Unstable, since the particular subset of information available can change fairly rapidly in the real world. For example, a farmer

receives daily weather reports, weekly crop surveys, daily market prices, and so on.

INCOMPLETE DOMAINS

How can a traditional reasoning program, such as Planner [9], be extended for use in a domain with incomplete information? Consider, for example, how such a program might go about making decisions concerning a crop of wheat, based on the problem: "Will the price of wheat rise?". Let's call our program Sleuth. Initially Sleuth's reasoning processes will resemble Planner. We formulate the problem as follows:

```
(Thgoal (price-increase wheat))
```

This goal is a hypothesis about the price of wheat. Sleuth makes inferences on the current set of assertions to see if the hypothesis can be supported. If Sleuth knows the following theorems and assertions, it will be able to construct a plausible chain of inference:

```
(Thassert Supply Wheat 180,000,000 bushels)
(Thassert Demand Wheat 170,000,000 bushels)
(Thassert Carryover Wheat 15,000,000 bushels)
(Thassert old-supply wheat 182,000,000 bushels)

(Thconse Thm1 (X) (price-increase ?X)
    (Thor (Thgoal (Supply-&-demand ?X))
          (Thgoal (Speculation ?X)))

Thconse Thm2 (X S D C) (Supply-&-demand ?X)
    (Thcond ((Thand (Thgoal (supply ?X ?S bushels))
                    (Thgoal (demand ?X ?D bushels))
                    (Thgoal (carryover ?X ?C bushels))
             (greaterp (?C) (- ?S ?D))))
```

Thm1 specifies that to show a price increase for wheat, try either of two subgoals. The first subgoal specifies a pattern which matches that of Thm2. Thm2 will succeed only if the difference between supply and demand is less this year than last. If so, Thm1 will in turn

succeed. A price increase for wheat is inferred, and wheat plantings can be increased. (Note that we are simplifying the decision processes involved in farming. Sleuth has not looked at demand for alternate crops; whether production costs on wheat have gone up; nor what sort of growing season is predicted. However, our purpose is not to show how Planner can be used in farming, but to illustrate some limitations of Planner.)

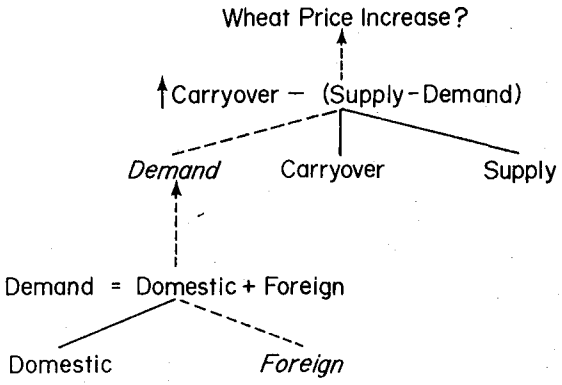
Current demand for wheat is reported weekly by the U.S.D.A., based on domestic reports, and satellite observations of foreign lands. Suppose that the next week, due to very foggy weather in central Asia, no satellite photos are taken. As a result, the U.S.D.A. issues no new demand statistics. This time, when Sleuth is asked about the price of wheat, the previous inferences would fail, since no assertion matching the pattern for (Demand Wheat ?D bushels) would be found. For such cases, Planner provides a strategy. If an assertion is not in the database, Planner will try and prove it:

```
(Thconse Thm3 (X DD FD) (demand ?X)
  (Thcond ((Thand (thgoal (domestic-demand ?DD))
                  (thgoal (foreign-demand ?FD)))
           (Plus ?DD ?FD))))
```

Thm3 states that to deduce demand for a commodity, find the foreign and domestic demand for this commodity, and add these together. If there are assertions for foreign and domestic demand, this theorem would succeed. However, since the total demand reported by the U.S.D.A. is based on the missing estimates of foreign demand, Thm3 will also fail. In this case, if there are no other methods for proving the missing assertion, Sleuth must give up.

(In this and subsequent diagrams italics and a dashed line are used to indicate antecedents which are missing. A solid line is used

to connect antecedents to rules; similarly, a solid line and arrow is used to indicate deduced links between rules and assertions, while a dashed line and arrow indicates inferences which have failed.)



This presents a fairly brittle mechanism for dealing with domains which share the properties of incompleteness. If the needed assertions are not in the database, and if they cannot be infered, the inference attempt will fail. It will fail, however, not necessarily because it is wrong, but because not enough information exists to make inferences. People are not quite so brittle reasoners, since they often cannot postpone decisions until more knowledge is available. For example, a farmer might reason that as long as the supply of wheat is decreasing, it will be worthwhile to plant more regardless of demand. (Note: I am not suggesting that this is the "best" decision; merely that it is a plausible decision, and represents the sort of flexible reasoning of which people are capable.) By being willing to make assumptions, a farmer is able to use a form of a rule which can operate on the information available. We might express this by adding a second clause to the Thcond of Thm2:

```
(Thconse Thm2A (X S D C) (Supply-&-demand ?X)
  (Thcond ((Thand (Thgoal (supply ?X ?S bushels))
                  (Thgoal (demand ?X ?D bushels))
                  (Thgoal (carryover ?X ?C bushels)))
           (greaterp ?C (- ?S ?D)))
          (Thor (Thgoal (supply-decrease ?X))
                (Thgoal (demand-decrease ?X)))))
```

If the full set of assertions concerning supply, demand and carryover are unavailable, Thm2A now suggests either trying to prove that supply has decreased, or demand has increased. By creating goals that require only a subset of the assertions that the original theorem required, Thm2A starts to encode the notion of rule simplification.

However, Thm2A does not quite capture our intuitions about simplifications. A rule's simplifications are simply other rules which under certain conditions can be used to achieve the same goal as the original rule. Thus, in the above example, if, a) Thm2 fails because of missing information, and b) we are willing to assume the missing information will not change the outcome, then c) we can specify another goal, such as calculating only an increase in demand, d) which does not require the missing information. If this goal is achieved, we can consider it a simplification of our original rule, and replace our original goal with this new goal.

Thus, we consider a rule simplification to be any rule which achieves a particular goal, and not a specific rule. The knowledge concerning which goals can be used in generating particular instances of simplifications, and the conditional assumptions which must be made in order to substitute those goals for the original ones should be encoded in a more flexible manner than Thm2A. A rule should give advice about which other theorems can function in its stead as simplifications. We can then choose to use this advice or not,

depending on our strategy. There may be many alternatives to choose among, of different degrees of plausibility. By making the simplification a part of Thm2A, we lose this intuition. Given the conditional circumstances of the original theorem failing, and our willingness to make assumptions, the simplifications, for the purposes of making inferences, are considered equivalent to the original theorem although ordinarily they achieve different goals than the theorem they replace.

A rule can have more than one procedural counterpart. Part of Planner's contribution to the notion of pattern directed invocation of rules was the insight that a rule has both a consequent and antecedent meaning. These can be expressed as two classes of theorems, antecedent and consequent theorems, which can be invoked by different patterns. We can extend this a step further by postulating that a rule has another bundle of procedural counterparts corresponding to its simplifications.

Actually, these simplifications will simply be other rules. However the knowledge of when these other rules can be used as simplifications, and which rules can be used, must be represented. An intelligent rule interpreter can make use of this metaknowledge [3] to substitute simpler theorems for a rule which fails. The appropriate place to specify this metaknowledge [4] is in a separate class of theorems. e.g.

```
(Thconse Thm2 (X S D C) (Supply-&-demand ?X)
    (Thcond ((Thand (Thgoal (supply ?X ?S bushels))
                    (Thgoal (demand ?X ?D bushels))
                    (Thgoal (carryover ?X ?C bushels)))
             (greaterp ?C (- ?S ?D)))))

(Thassume Thm4A (X) (Supply-&-demand ?X)
    (Thgoal (supply-decrease ?X))
    (Thcaveat (Default)))

(Thassume Thm4B (X) (Supply-&-demand ?X)
    (Thgoal (demand-increase ?X))
    (Thcaveat (Thnot (Thgoal (supply-increase ?X)))))
```

We now introduce a new class of theorems, such as Thm4A and B, indicated by the label Thassume. These theorems contain information concerning simplifications and assumptions. A Thassumption will specify A) a goal; theorems satisfying this goal can function as a simplification; B) the assumptions involved in using that simplification. These are expressed as a caveat. If the assumption being made is that the missing antecedents can be ignored, the caveat will contain a Default. If there is some particular condition which must obtain in order to assume that the missing antecedents can be ignored, this will be expressed in the caveat. (An alternative approach is to use a generative theory of simplifications in which a rule can be examined and a simplification generated dynamically. The present solution can be thought of as the end step of such a process. The specification of the domain and metaknowledge necessary to achieve this is a complex task. However, see Carr and Goldstein [1] for a model of how this metaknowledge looks in one domain.)

In Thm4A, proving a decrease in wheat supply can function as a simplification of Thm2 (proving a decrease in the difference between supply and demand), and hence can be used to prove the goal of an increased price for wheat, should we choose to use a simplification.

(Of course, using one reduces the plausibility of the reasoning; thus, in the current example, simplifications are tried only after a rule fails. Under other constraints, our strategy might be different; for instance, if we wanted a "quick and dirty" answer.) Since no assumptions are specified in the Caveat, these can be ignored. This is explicitly expressed in the caveat as a default. If instead we use the goal of an increase in demand as a simplification, as in Thm4B, we must take account of the caveat that supply must not have increased for this simplification to be valid.

(Thus, the rule interpreter can, in using a simplification, check the validity of its caveat. Such a check can in turn involve applying theorems and attempting proofs. The degree to which we wish to pursue such a validation check will depend on our strategy. Potentially, of course, one could expend as much energy on these proofs as on the main proof. Such attempts have the virtue that they make the simplification more plausable. However, they can be expensive to perform. Planner itself provides one level of control, by allowing a theorem to indicate whether a proof of missing antecedents should be attempted, or if the theorem should simply fail.)

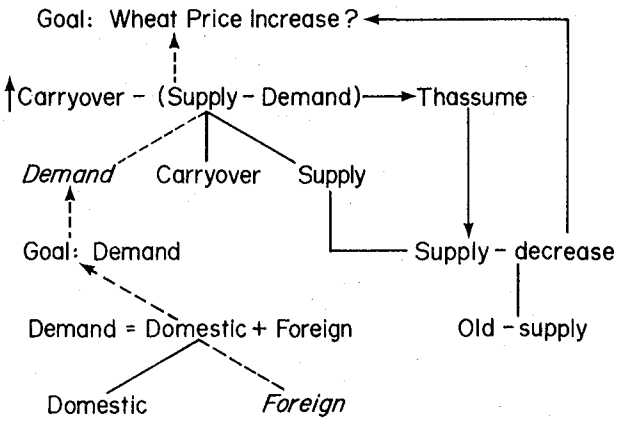
When Thm2 fails, Sleuth can choose to make assumptions which will allow a simplification to succeed on the assertions which are available, by using Thm4A or B. A rule and its associated simplifications are related through the set of assumptions they embody. When a farmer decides to ignore the demand for wheat, he is doing so because he is willing to assume that if demand changes, it will not change in a direction or quantity which would invalidate his reasoning. By making explicit this notion of assumptions, we can

extend the list of options available in using a theorem to achieve a goal.

Returning to our example, after Thm2 fails, simplifications will be considered, and Thm4A found. Thm4A first tries to satisfy the goal (Thgoal (Supply-decrease ?X)). No assertion matching this pattern exists. However, a theorem, Thm5, can be used to prove this assertion.

```
(Thconse Thm5 (X S OS) (Supply-decrease ?X)
  (Thcond ((Thand (Thgoal (Supply ?X ?S bushels))
                  (Thgoal (old-supply ?X ?OS bushels)))
           (Thcond ((greaterp ?OS ?S))))))
```

The value for old-supply was one of the initial set of four assertions. Since current supply and old-supply are known, Thm5 will succeed, and support the hypothesis of higher wheat prices.



This chain of inferences results in a less plausible scenario than one requiring no simplifications.

UNSTABLE DOMAINS

The outcome of an inference attempt on some set of data in support of a hypothesis will be a set of assertions and rule instances, joined by various labeled links. This is a process trace. We will call the part of a process trace associated with the use of a particular instance of a rule its annotation. The process trace is more than a trace of a proof since failed rules and unsuccessful proofs are also recorded. As the database changes, our goals may remain relatively stable. Thus we would like to maintain current goals, and to prove failed hypotheses when changes in the database allow this. This will be reflected in the changing set of annotation associated with each hypothesis. We assume that support for an hypothesis is conditional on the assertions available at the time it was first considered (i.e., the inferences which were possible at that time). Hence this support must be monitored and changed as the database changes. All proofs are conditional on the validity of the assertions used by the rules in the proof. However in many deductive systems, once deductions are made from a set of assertions, no effort is made to insure that while the results of those deductions are used, the assertions still hold true. Generally, the user does not expect the database to change so as to invalidate prior inferences.

We can extend our concept of rule interpretation by making the maintenance of goals a function of the interpretation of rules. We implement this in Sleuth by giving each active rule the autonomy to respond to changes in its environment. As each rule is interpreted, an associated Sentinel is created for that instance of the rule. The Sentinel gives the rule instance the knowledge of how to respond to

changes in its antecedents or consequents. The result is the maintenance of hypotheses through a method of local autonomy.

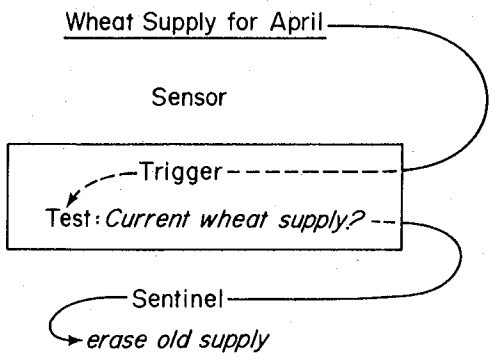
By using Sentinels, we extend the basic idea of a rule which is evaluated successfully if its antecedents are satisfied at the initial time of evaluation. A rule instance must be continuously enabled while it is used in support of some hypothesis. Essentially, we wish any rule to be enabled as long as we maintain interest in the top level goal which first evoked it, although the rule itself may have failed. This can be extended one step further. The hypothesis (top level goal) may have failed. However, the goal of supporting that hypothesis is maintained by keeping that goal assertion in the database. In that case, the theorems attempted are still active. If, at a later time, they can succeed, they will reactivate the attempt.

Sentinels (developed by myself and Jim Stansfield as an aid in instantiating frames and automating the recognition of simple sequences of events within a changing database, using FRL [7]) are associated with the application of rules, by making the use of Sentinels a property of the rule interpreter. A Sentinel associated with a rule instance will place triggers in the database which respond to changes in goal assertions, or antecedents.

A triggered Sentinel can take a variety of actions. The standard ones are to A) erase itself; B) erase the annotation; C) reinvoke a goal.

A Sentinel has sensors which report to it. A sensor has a two part condition. The first part, a trigger, is a demon which responds to changes in the pattern that triggers it. For example, in the following case, the trigger responds to any addition or deletion of

patterns involving the wheat supply. The sensor then tests the pattern against some criterion. For instance, this sensor is only interested in assertions concerning current wheat supply:



A Sentinel can have many sensors which report to it. The Sentinel is satisfied when some arbitrary logical conjunction of its sensors succeed. Although for the task of maintaining hypotheses more complex relations are not needed, a Sentinel has the capacity to evaluate conditional relations among its sensors, and even to remove current sensors and place new ones as a response to these conditional constraints. It can also make use of the temporal dimension in conjunction with the logical organization of its sensors. For instance, an "and" relation among these sensors can be created in which all sensors are satisfied at the time the Sentinel is evoked, or the relation among the sensors can be that they were all satisfied at some preceeding time (and, if desired, in some specific order) but at the time the last sensor is satisfied, and evokes the Sentinel, the state of the other sensors is unknown.

The sensors function as triggers for the sentinel, which is "data driven". A sentinel and its sensors are theorems which are created

for a specific purpose. Unlike other theorems in the database, they have a limited lifespan. A sentinel can choose to erase itself and its sensors upon completing its goal. For the current task, sentinels are not required when their associated rule instance is no longer enabled. In this case, the sentinel will erase itself.

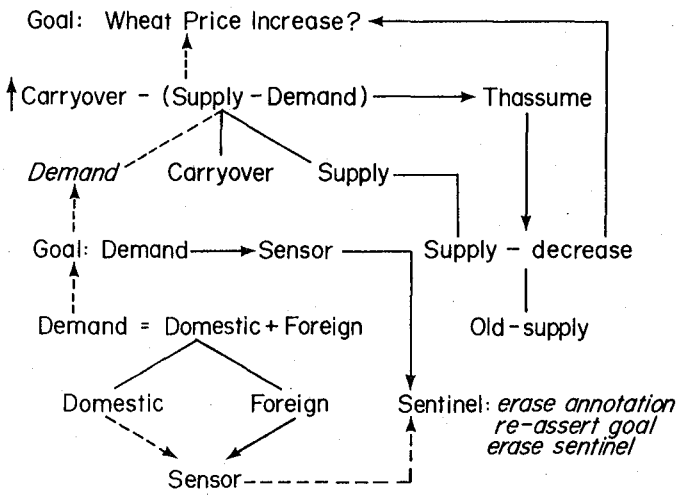
There are several aspects of a rule's environment we may wish monitored. Typically, we will want to be able to monitor a rule's antecedents, the goal which invoked the rule, and more powerful rules which could supercede a rule if they were used. Not all of these will always be monitored. The interpreter, at the time a rule is applied, will examine its environment and decide which of these should be monitored.

Individual rules will succeed or fail as a function of their antecedents. When a successful rule's antecedents change, its sentinel will be triggered. When this happens the sentinel causes the goal the rule was supporting to be re-evaluated. It removes the old annotation, as new annotation is created for the new evaluation of the goal. At this point the sentinel can erase itself. (Note: A sentinel causes a goal to be re-evaluated. There is no constraint that the same rule be used again. At this point another rule may now be the best choice. However, in this and the following examples, it is assumed that there have been no other changes in the state of the system that would cause another rule to be selected first.)

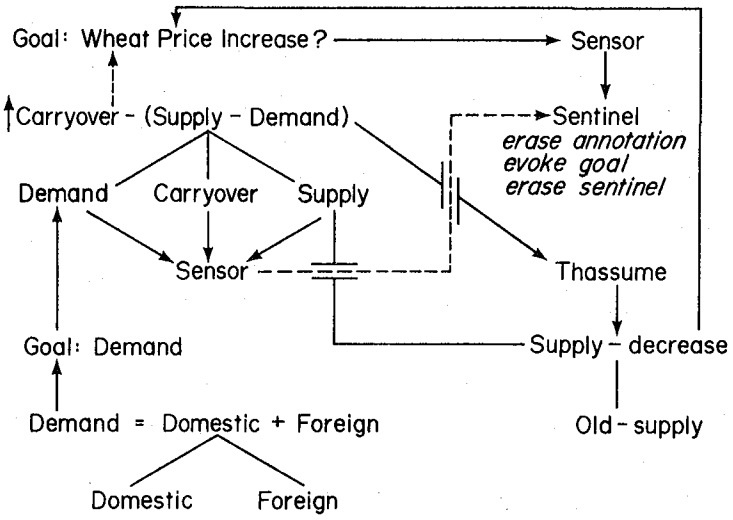
If a rule's goal is erased, its sentinel will also be triggered. In this case we do not wish to re-evaluate the goal. The sentinel will remove itself and erase the associated annotation. Thus the rule instance will no longer be active, since no trace of it will remain.

Now let's consider how this local association of rule instances with sentinels can give rise to the right global behavior. Figure 1 represented the current state of our deductions.

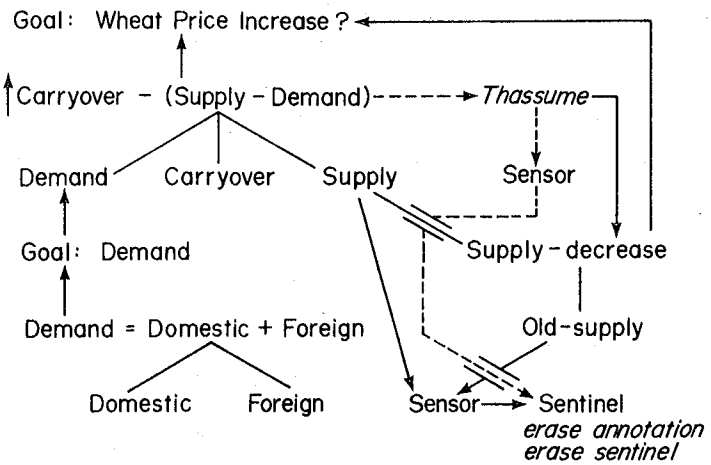
Suppose that a failed rule now is capable of succeeding, through its missing antecedent being asserted. For instance, the missing foreign demand for wheat can be asserted.



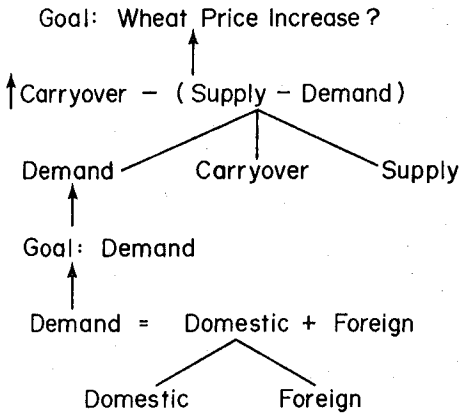
This will trigger the associated sentinel to erase the annotation for this rule instance, reassert the goal as something to be proved, and then to erase itself. This time the rule succeeds, resulting in a proof of the missing demand for wheat. This will in turn trigger the sentinel associated with the rule instance of Thm2 for which the missing demand is an antecedent:



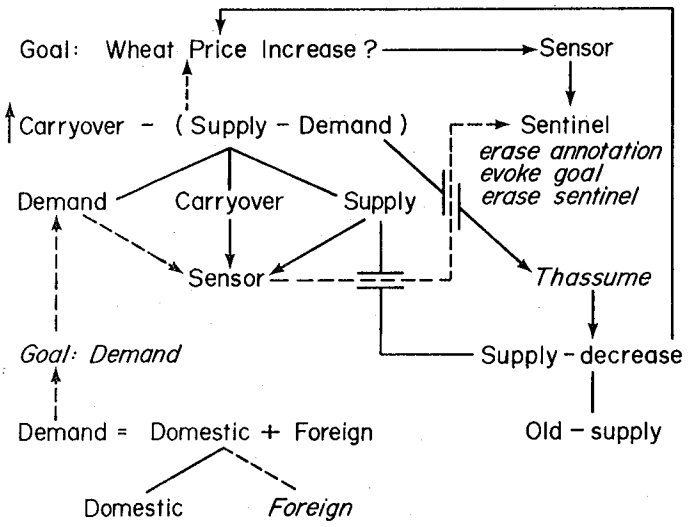
This sentinel repeats the actions of the prior sentinel. However, in erasing the annotation, it erases the record of the assumption made. This will trigger the sentinel on the rule which is a simplification. Since the use of a simplification is conditional on another rule failing, the sentinels associated with theorems used as simplifications monitor the annotation recording that failure, so that they will know when the simplification is no longer required. They will then respond to the erasure of this annotation by erasing the annotation for the simplification.



If Thm2 again failed due to a missing antecedent, Sleuth would once more try a simplification. Since the formerly missing antecedent for demand has now been infered, Thm2 succeeds, and results in the following final inference tree:

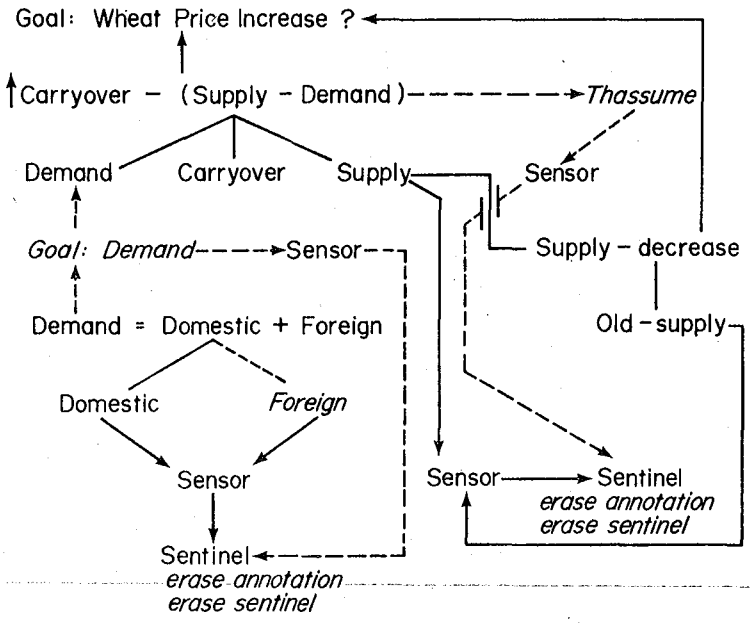


In this next example, the missing assertion for current wheat demand is asserted, although the rule involved (Thm2) has already "succeeded" by using a simplification. This will trigger the sentinel associated with the rule instance of Thm2:

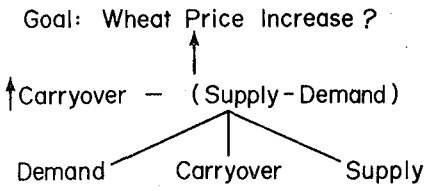


When Thm2 is re-evaluated it succeeds without recourse to either

using a simplification or trying to prove the now present antecedent. There is no explicit mechanism responsible for removing the now unneeded rule instances. Instead, by erasing its annotation, Thm2 triggers the sentinels associated with the subgoal of proving demand, and the simplification:



Both these sentinels, since the sub-goal has been removed, erase the annotation, and then erase themselves. This results in the following final state:



Thus, unneeded rule instances will know when to remove themselves. Through local propagation, the representation responds to changes in the available database. Consequently, once a question has been

specified, a dynamic process is invoked which once attempted can be locally data-driven. These changes will reinvoke the goal of inferencing, which can then proceed in a goal driven fashion. Obsolete parts of the representation are able to remove themselves by noticing local changes in the environment.

Sleuth, once given a goal, would attempt to deduce this goal whenever the database contains the right set of assertions. Sentinels set in the interpretation of rules will individually call Sleuth to re-evaluate particular goals. Sleuth will develop new ways of supporting its hypotheses in response to these local calls for re-evaluation. Once applied, each sentinel has the autonomy to respond to changes in the database.

Local changes in the environment of a rule supporting a goal will always cause that goal to be re-evaluated, although the result may be the same. For instance, a new proof of an inferred antecedent may occur. Since the rule's antecedents have not changed, re-evaluating the goal can result in the same rule being chosen, and succeeding in the same manner. However, changes elsewhere in the annotation might affect our strategy, resulting in a different rule choice. Other strategies differentiate among the proofs of antecedents. This occurs, for instance, in systems where plausability is taken into account.

Sleuth does not represent a reasoning system per se. It is a set of features for an interpreter applying a reasoning strategy to some domain. These features are designed to allow reasoning to proceed over a changing database. Perhaps the closest approach to these ideas has been that of Doyle [4]. Doyle has provided a mechanism for

recording deductive dependencies so that when facts turn out to be incorrect, the entire "context" of dependent facts can be removed from the database. His work develops the notion of dependency directed backtracking by making the use of contexts very explicit, just as Conniver [10] did with the chronological backtracking of Planner. Sleuth does not allow explicit manipulation of deductive dependencies, since the changes it is designed to respond to are those imposed on the domain by "outside" changes in the data, and not ones due to deduced inconsistencies. However, Sleuth does provide an automatic control structure so that the user can ignore the problems of a changing database, and focus on the deductions he wishes to make.

The rule interpretation features of sentinels and simplifications were programmed in FRL (Frame Representation Language) [7], and not in Planner, which was chosen for the examples since it is a well understood language. While the features of the use of simplifications and of sentinels exist in the current version, the Planner-like features of chronological backtracking, and pattern-directed invocation of rules used in the examples exist in only a rudimentary form in the FRL language.

Sleuth emphasizes an inference-based approach to understanding, whereby links between pieces of knowledge are created through goal directed inferencing. This approach ignores issues which more knowledge based theories have focused on, such as the use of prior knowledge of the domain [2], or the usefulness of powerful sets of semantic primitives in making plans [11]. McDonald [6] has reviewed these three approaches in terms of the strengths and weaknesses of each approach. Ultimately, a complete theory of understanding will

have to incorporate elements of many of these partial models. One aspect of such a complete theory will explain how our world model is able to respond to the continual changes and incompleteness of actual situations.

ACKNOWLEDGEMENTS

The author would like to thank Ira Goldstein and Jim Stansfield for their helpful comments and advice.

REFERENCES

[1] B. Carr and I. P. Goldstein, "Overlays: A Theory for Computer Aided Instruction," MIT-AI Memo 406, 1977.

[2] E. Charniak, "A Framed PAINTING: the Representation of a Common Sense Knowledge Fragment," Cognitive Science, 1, 4, 1977.

[3] R. Davis and B. G. Buchanan, "Meta-Level Knowledge: Overview and Applications," Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Mass., 1977.

[4] J. Doyle, "Truth Maintenance Systems for Problem Solving," M.I.T. Artificial Intelligence Laboratory Technical Report 419, 1977.

[5] G. W. Ernst and A. Newell, GPS: A Case Study in Generality And Problem Solving. New York: Academic Press, 1969.

[6] D. McDonald, "Story Understanding: the Beginning of a Concensus," MIT-AI Working Paper 168, 1978.

[7] B. R. Roberts and I. P. Goldstein, "The FRL Manual," MIT-AI Memo 409, 1977.

[8] E. D. Sacerdoti, A Structure for Plans and Behavior. New York: Elsevier North-Holland, 1977.

[9] G. J. Sussman, T. Winograd and E. Charniak, "Micro-Planner Reference Manual," MIT-AI Memo 203A, 1971.

[10] G. J. Sussman and D. V. McDermott, "Why Conniving is Better Than Planning," MIT-AI Memo 255A, 1972.

[11] R. Wilensky, "Using Plans to Understand Natural Language," Proceedings of the ACM, Houston, 1976.

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720