Reset reproduction of a master's thesis proposal submitted November 2, 1976 to the Massachusetts Institute of Technology Electrical Engineering Department, reprinted November 16, 1976 as MIT Artificial Intelligence Laboratory Working Paper 133. Reset in LaTeX by Jon Doyle December 1995. Available via http://www.medg.lcs.mit.edu/doyle. Author's current address: Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA.

The Use of Dependency Relationships in the Control of Reasoning

Jon Doyle DOYLE@MIT.EDU
Massachusetts Institute of Technology, Artificial Intelligence Laboratory
Cambridge, Massachusetts 02139, U.S.A.

Abstract

Several recent problem-solving programs have indicated improved methods for controlling program actions. Some of these methods operate by analyzing the time-independent antecedent-consequent dependency relationships between the components of knowledge about the problem for solution. This paper is a revised version of a thesis proposal which indicates how a general system of automatically maintained dependency relationships can be used to effect many forms of control on reasoning in an antecedent reasoning framework.

1 Introduction

A major problem encountered in building automatic problem-solving systems is the necessity of providing controls on program reasoning and action. When faced with this problem, however, some researchers have divided reasoning into the categories of antecedent (data-directed) and consequent (goal-directed) reasoning. They then claim that the problem of control is just that of deciding on a proper balance between antecedent and consequent representations of program knowledge. In contrast to this, we argue that the dichotomy of reasoning into antecedent and consequent reasoning is misleading, and is due to an inadequate understanding of the nature of control of reasoning. Consequent reasoning is only one method among many available for the control of program actions. In this paper, we indicate how dependency relationships between facts may be used to effect various controls on reasoning, including consequent reasoning, in an antecedent reasoning framework. This embedding allows the processes controlling program reasoning to enjoy the many advantages of antecedent reasoning.

1.1 The Basic Reasoning Process

The basic reasoning process available to programs is a generative one, in which current knowledge is used to deduce or create additional knowledge. This type of reasoning process is best captured by antecedent, event-triggered reasoning, since antecedent-driven processes enjoy the important qualities of being non-chronological and additive. Since antecedent processes are non-chronological in nature, they provide a knowledge representation free of many problems of interactions induced by time-dependence of computations. A concomitant benefit of antecedent reasoning is that it makes the reasoning involved in dependencies between facts clear and easily recorded; the dependencies are derived from the logical dependencies between facts, and are not dependent upon spurious chronological accidents. That is, the reasons for the program's knowledge about a fact consist only of the facts used in its derivation, and not on the particular chronological history of the program's actions. (This quality of event-triggered or event-enabled reasoning is also attractive from the viewpoint of Dijkstra [1976], whose guarded commands are essentially event-enabled processes.) An important consequence of the non-chronological nature of antecedent reasoning is additivity. As the order of program actions is not critical, so the time and order in which new rules become known is not particularly critical either. This additivity allows a distributed style of computation in which missing knowledge does not necessarily cause program failure, but merely limited program capabilities.

The major limitation on the use of the basic antecedent reasoning process is that it must be controlled. Effectiveness and efficiency demand that there be mechanisms for focusing and directing the actions taken by the program, since in infinite domains (domains in which infinitely many facts are derivable), and in finite domains with more answers than interesting questions, the vast majority of program actions and deductions are irrelevant to solving the problems of interest and should be avoided. This is the genesis of the problem of control of reasoning, as in standard axiomatizations of most domains the combinatorial nature of the method of combining small actions and deductions into a solution necessarily dominates the considerations of the designers of problem-solving programs. Even though the range of program actions may be limited by employing self-limiting representations or axiomatizations for the components of knowledge of the domain (such as are employed in Nevins' [1974] geometry theorem prover and in Minsky's [1974] theory of Frames), the domain may still be large enough so that direction is needed in the slot-filling process.

1.2 Controls on Reasoning

Many methods have been devised for controlling program actions. The most fundamental method is that of explicitly programming the logic of the solution construction process. For instance, one form this method assumes is the programming of the process as a decision tree, in which all possible cases must be considered. This method has long been known to suffer by being highly inflexible when extensions and modifications to the program are required.

A number of control strategies have been developed in the context of resolution theorem proving [Robinson 1965] (see Chang and Lee [1973] for examples), but in general these have proved inadequate; apparently these strategies are more directed towards reducing the effort involved in generating and testing a proposed solution, rather than aiding in reducing the effort involved in searching the space of problem solutions. That is, these strategies are primarily optimizations of theorem prover operations which are independent of domain, and provide little or no help in introducing domain-dependent control mechanisms [de Kleer 1976a].

Another strategy for controlling reasoning is that of consequent reasoning, the strategy of reasoning from goals to known facts. (In a theorem-proving context, the use of consequent reasoning is usually called the set-of-support strategy.) Consequent reasoning is a very common method of controlling search, as it is a simple mechanism for enforcing a certain form of relevance in actions toward deciding the current questions by only using rules of inference or action which mention a current goal in their conclusion. It is important to note, however, that anything deducible via consequent reasoning is also deducible by antecedent reasoning - that is, consequent reasoning is just a particular way of controlling antecedent reasoning. (Indeed, Nevins' [1974] geometry theorem prover's use of consequent reasoning is essentially identical to its use of antecedent reasoning, and, as de Kleer [1976a] observes, could apparently be made superfluous by a slight modification to the main control loop of the program. See [Moore 1975] for additional discussions of these issues.)

Backtracking is another method of controlling reasoning. Backtracking allows the focusing of program attention on just one set of possibilities at a time, considering other sets of possibilities only if the first set of choices leads to failure. Automatic backtracking mechanisms have fallen into disfavor since the advent of MICRO-PLANNER, in which chronological, side-effect-free automatic backtracking was made an virtually unavoidable program control mechanism. This form of automatic backtracking has many problems associated with it. (See [Sussman and McDermott 1972] for a discussion of these problems.) Many of the unpleasant features of automatic backtracking are remedied, however, by the use of a non-chronological, dependency-directed backtracking system [Stallman and Sussman 1976].

Other methods for controlling reasoning include the use of meta rules, protocols, and task networks. Meta rules [Davis 1976] provide a mechanism for controlling program actions at times when more than one action is known to be feasible. Meta rules have the feature of being useful at many levels: in controlling the basic program actions (first level use), in selecting which strategy to use (second level use), in deciding how to select a strategy (third level use), etc. The use of specialized protocols finds related applications in guiding program actions. Examples of protocols are the choice and rephrasing protocols of NASL [McDermott 1976], the NOAH plan refinement loop [Sacerdoti 1975], the SIMPLE ATN [Goldstein and Miller 1976], and Nevins' partitioned deduction loop [Nevins 1974]. These protocols are invoked in particular types of situations, and frequently allow considerable efficiency in performing standard patterns of action. Task networks [Sacerdoti 1975, McDermott 1976] make an explicit representation of past, present, and future program actions available to the program itself. Thus task networks, like meta rules and protocols, allow the direction of current and future reasoning to be influenced by the program's knowledge about its own state of knowledge and past, present, and future plans.

1.3 The Antecedent Embedding of Control

The view proposed here is that the view of these mechanisms as controls on reasoning is best combined with the use of dependency relationships among facts to implement these forms of control within the framework of antecedent reasoning, and that this embedding of control mechanisms alleviates many problems encountered in their standard implementations.

One such problem affected by this embedding is the problem of backtracking. In standard implementations, backtracking is usually used in conjunction with consequent reasoning in a chronological fashion. Traditionally, backtracking is accomplished by rechoosing the most recently made reversible choice, a process which normally requires the consideration of many obviously irrelevant sets of choices. By using dependency relationships, which when produced from the process of antecedent reasoning are already free of chronological dependencies, the efficient method of dependency-directed backtracking of Stallman and Sussman [1976] can be implemented. In this form of backtracking, only those choices relevant to the current failure are considered for change. In addition, recording of the demonstrably infeasible sets of choices reduces future search considerably by ruling out any proposed set of choices including any of the the known infeasible sets of choices.

Consequent reasoning also benefits from being embedded in an antecedent reasoning framework by being transformed into a non-chronological process. Traditional implementations of consequent reasoning have been chronological in nature. Because of this, the effects of incomplete knowledge have been harder to deal with, in that the chronological order of unsuccessful searches, assumptions, and discoveries becomes significant. The wrong chronological orderings may so induce an inconsistency in the program knowledge which, without usefully recorded dependency information, is difficult to analyze and remedy. Also, additional work is induced as successive queries about a problem or about related problems normally have to perform the same search effort as in the first query. This latter problem arises from the difficulty of recording dependencies in a chronological system, which forces searches to recompute previously derived, but discarded information, so causing subsequent queries to require the duplication of entire searches or subsearches. (Fikes [1975], however, has proposed some mechanisms for alleviating these problems in a chronological system. These mechanisms amount to some of the fragments of the ARS dependency system which are easy to use in a chronological system.)

2 Facts and Dependencies

Each component of program knowledge is recognized as a distinct entity called a *fact*. Facts are used in describing the dependency relationships between the different components of program knowledge. Belief in the truth of a particular fact may or may not be supported by belief in the knowledge embodied in other facts. If a fact is known to be true by virtue of its relationships with other facts, we say the fact is *in*; otherwise the fact is *out*. The distinction between *in* and *out* is not the same as the distinction between true and false. To represent the true/false dichotomy, each fact may have a *negative*. The negative of a fact represents the assertion which is true if and only if the fact is false. The negative of a fact is a fact itself, and also will be either *in* or *out* depending upon its support by other facts. We define six predicates on facts which describe the possible states of knowledge about the fact:

$$
\begin{array}{lcl}
\texttt{IN}(f) & \equiv & f \texttt{ is } in \\
\texttt{OUT}(f) & \equiv & \neg\ \texttt{IN}(f) \\
\texttt{NEGIN}(f) & \equiv & \texttt{the negative of } f \texttt{ is } in \\
\texttt{NEGOUT}(f) & \equiv & \neg\ \texttt{NEGIN}(f) \\
\texttt{KNOWN}(f) & \equiv & \texttt{IN}(f) \vee \texttt{NEGIN}(f) \\
\texttt{UNKNOWN}(f) & \equiv & \neg\ \texttt{KNOWN}(f).
\end{array}
$$

Observe that `IN`, `NEGIN`, and `UNKNOWN` correspond to the classical divisions of `TRUE`, `FALSE`, and `UNKNOWN` of 3-valued logic. It is therefore a contradiction for both a fact and its negative to be *in* simultaneously.

Each fact derives its support from its *antecedent set*. Each antecedent in the antecedent set of a fact is a boolean function of the above status predicates of other facts. A fact is *in* if one of its antecedent functions is true, and is *out* otherwise. If a fact is *in*, a single one of its antecedents may be designated as the fact's *support*; the support of an *out* fact is the entire antecedent set of the fact. Thus the status of a fact remains unchanged in new deductions if the status of each of the facts in its support is unchanged.

This structure for the antecedents of a fact is a generalization of the ARS dependency system [Stallman and Sussman 1976]. In this system, each antecedent in the antecedent set of a fact corresponds to an alternate derivation of the fact. Some systems (such as those of Fikes [1975] and McDermott [1976]) also record the support dependencies of facts, but only record one derivation at a time. The need to record all derivations was first realized in the ARS system, as it was discovered that the single-antecedent scheme allowed the process of backtracking to produce circularities in the support of facts. Since ARS uses reasoning about the antecedents of a fact to control search, this phenomenon required the introduction of antecedent sets and "fact garbage collection."

2.1 Truth Maintenance

Antecedents in ARS are of a simple and monotonic nature, in that the *in*ning of a fact cannot (except in the case of contradictions) cause the *out*ing of other facts. Because of this monotonicity, determination of the statuses of facts in ARS is accomplished by means of two processes termed fact garbage collection and un*out*ing. Fact garbage collection occurs each time the status of a fact is changed from *in* to *out*, an occurrence typically due to the rechoosing of a choice during backtracking. ARS's fact garbage collector examines each fact to choose an antecedent which provides non-circular support for the fact. Un*out*ing is the complementary process: when new support is derived for an *out* fact, the fact's status is changed from *out* to *in*, and all of its currently *out* consequences which then have support are un*out*ed recursively.

The generalized dependency system described above requires the unification of fact garbage collection and un*out*ing into a uniform process called "truth maintenance." The possibility of the status of a fact depending upon another fact being *out* makes the system of dependencies non-monotonic. This non-monotonicity means that un*out*ing is no longer possible: *whenever* the support for a fact is changed, truth maintenance must be performed to provide each fact with well-founded support. However, truth maintenance need only be concerned with those facts whose support may actually be affected by the initially changed fact. Accordingly, this means that the process of truth maintenance can be incremental. This realization shows that the ARS system, which garbage collects *all* facts whenever any fact is *out*ed, is doing unnecessary work, and could be profitably changed to use an incremental fact garbage collector.

In addition to requiring a unified truth maintenance system, the dependency schema introduces new forms of inconsistencies which must be recognizable. Three types of contradictions are recognizable in general: explicit contradictions (facts which are known to represent contradictions), contradictions arising from both a fact and its negative having support and thus being considered *in*, and unsatisfiable dependency relationships among facts. (An example of the latter type of contradiction is a fact f whose antecedent set is `OUT(`f`)`. This antecedent set would force f to be *in* if and only if it was *out*—a contradiction.)

A necessary property of unsatisfiable dependencies is that they must involve strong circularities in the dependency structure. (I use the adjective "strong" here as these circularities are really just the strongly connected components of the directed graph arising from the natural interpretation of the dependency structure as a directed graph on antecedent sets.) For instance, in the above example of a fact f which is *in* iff it is *out*, a strong circularity is evident: well-founded support for f cannot be chosen until well-founded support is available for its antecedents, in this case f itself.

Not all strong circularities involve unsatisfiable dependency relationships, however. Strong circularities arise naturally in situations involving equalities and equivalences. In these situations there is a natural solution to the problem: the status of one of the facts is arbitrarily chosen to be *out*; this then determines the status of the other facts involved in the circularity. Modifying this method, however, are considerations due to the possibility of noncircular dependencies between facts in distinct strongly connected components. This possibility requires that the choices in distinct strong circularities be consistent. Such consistent choices can be found by a process of topologically sorting the strong circularities coupled with backtracking. In such cases, inconsistencies are manifested as the event of determining supposedly well-founded support for a fact with arbitrarily chosen status, such that the newly determined status differs from the chosen status. In this easily detectable event, one of the choices involved in the inconsistency must be rechosen. If no choice can be rechosen without creating another inconsistency, the antecedent structure involved is determined to be unsatisfiable.

It should be pointed out that while the problem of finding a satisfiable assignment of statuses to facts is NP-complete, and whose solution by the above algorithm is potentially exponentially expensive, all the known examples of such strong circularities are part of a monotonic dependency relation, and thus are amenable to the method of choosing of all involved facts to be *out*. Indeed, I have been unable to construct a natural example in which unsatisfiable antecedent structures occur, and so have not been worried by the potentially expensive computations involved. The next section supports this attitude by demonstrating the nature of several common dependency structures, all of which are easily managable.

2.2 Typical Dependency Structures

This dependency scheme is sufficient to express many types of dependency relationships. We give five simple examples of their use.

AND

If f represents $\texttt{AND}(f_1, \ldots, f_n)$, then f has an antecedent of the form $\texttt{IN}(f_1, \ldots, f_n)$. In addition, the negative of each f_i has an antecedent of the form

$$\texttt{IN}(f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_n) \wedge \texttt{NEGIN}(f).$$

(We extend the six status predicates above to predicates of arbitrary sets of facts in the natural fashion by defining a predicate on a set to be the conjunction of the predicate on the elements.)

OR

If f represents $\texttt{OR}(f_1, \ldots, f_n)$, then f has an antecedent of the form $\texttt{IN}(f_i)$ for each f_i. In addition, each f_i has an antecedent of the form

$$\texttt{NEGIN(}f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_n\texttt{)} \wedge \texttt{IN(}f\texttt{)}.$$

XOR

If $f_1, \ldots, f_n$ are mutually exclusive and exhaustive cases, then the negative of each f_i has an antecedent of the form $\texttt{IN(}f_j\texttt{)}$ for each $j \neq i$.

Ordered Choices

If $f_1, \ldots, f_n$ represent mutually exclusive choices, with a preference for choosing f_i over f_{i+1}, then the following relationships exist: f_1 has the antecedent $\texttt{NEGOUT(}f_1\texttt{)}$, and for $i > 1$, f_i has the antecedent

$$\texttt{NEGOUT(}f_i\texttt{)} \ \wedge \ \texttt{NEGIN(}f_1, \ldots, f_{i-1}\texttt{)}.$$

PRESUMABLY

If f is to be assumed true unless it is provably false, f can be given the antecedent $\texttt{NEGOUT}(f)$.

The above example of the use of this dependency system to describe the `PRESUMABLY` operator demonstrates the flexibility of this system as compared with a simple 3-valued logic of `TRUE`, `FALSE` and `UNKNOWN`. The concept of `PRESUMABLY` is considerably harder to describe and maintain automatically in such systems.

3 Consequent Reasoning

For the purposes of efficient computation we distinguish each component of a certain subset of the knowledge of a program as being of a special type of fact called a *rule*. Rules are not just static knowledge, but have an imperative meaning attached to them. The basic structure of antecedent rules is that of

```
<trigger> ⟹ <body>,
```

where `<trigger>` is a set of facts (named, of course, after Roy Rogers' horse). The operation of the rule is such that when all of the facts in the trigger are asserted, `<body>` is performed. (Or, as GLS suggests, when the trigger is pulled, the body is executed.)

A straightforward use of such rules in implementing consequent reasoning is as follows. For simplicity, we describe only the case of deduction. To effect a rule

```
R: A → C
```

in consequent fashion, an antecedent rule of the form

```
R1: IN((OPERATIONAL R)) ∧ IN(A)
        ⟹
    assert C with antecedent (AND (IN A) (IN R))
    assert (SATISFIED (GOAL (DEDUCE C)))
        with antecedent (AND (IN A) (IN R))
```

is asserted with antecedent (IN R), as is another antecedent rule of the form

```
R2: IN((GOAL (DEDUCE C))) ∧
    NEGIN((SATISFIED (GOAL (DEDUCE C))))
       ⟹
    assert (GOAL (DEDUCE A))
        with antecedent
        (AND (IN (GOAL (DEDUCE C))) (IN R)
             (NEGIN (SATISFIED (GOAL (DEDUCE C)))))
    assert (PRESUMABLY (NOT (SATISFIED (GOAL (DEDUCE A)))))
        with antecedent
        (AND (IN (GOAL (DEDUCE C))) (IN R)
             (NEGIN (SATISFIED (GOAL (DEDUCE C)))))
    assert (OPERATIONAL R)
        with antecedent
        (AND (IN (GOAL (DEDUCE C))) (IN R)
             (NEGIN (SATISFIED (GOAL (DEDUCE C))))),
```

and finally, the fact

```
F: (PRESUMABLY (NOT (SATISFIED (GOAL (DEDUCE C)))))
```

is asserted with antecedent (IN R).

3.1 An Example

To demonstrate the operation of this method of consequent reasoning, we present a simulation of the following familiar example in detail. We leave out some computations which might occur but which do not substantially affect the flavor of the process, and ignore a number of questions raised by facts with variables in their statements and the process of rule creation. The initially known (*in*) facts are

```
F1: (GREEK SOCRATES)
F2: (HUMAN SOCRATES)
F3: (HUMAN TURING).
```

There is also a consequent rule,

```
F4: (HUMAN ?X) → (FALLIBLE ?X).
```

(As is usual, free "?" variables are assumed to be universally quantified.) By the above process, F4 produces two antecedent rules,

```
R1: IN( (OPERATIONAL F4) ) ∧ IN( (HUMAN ?X) )
       ⟹
    assert (FALLIBLE ,X)
        with antecedent (AND (IN F4) (IN (HUMAN ,X)))
    assert (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X))))
        with antecedent (AND (IN F4) (IN (HUMAN ,X)))
```

```
R2: IN( (GOAL (DEDUCE (FALLIBLE ?X))) ) ∧
    NEGIN( (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X)))) )
        ⟹
    assert (GOAL (DEDUCE (HUMAN ?X)))
        with antecedent
        (AND (IN (GOAL (DEDUCE (FALLIBLE ?X)))) (IN F4)
             (NEGIN (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X))))))
    assert (PRESUMABLY
            (NOT (SATISFIED (GOAL (DEDUCE (HUMAN ?X))))))
        with antecedent
        (AND (IN (GOAL (DEDUCE (FALLIBLE ?X)))) (IN F4)
             (NEGIN (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X))))))
    assert (OPERATIONAL R1)
        with antecedent
        (AND (IN (GOAL (DEDUCE (FALLIBLE ?X)))) (IN F4)
             (NEGIN
              (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X)))))),
```

and the fact

```
F5: (PRESUMABLY
     (NOT (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X))))))
```

with antecedent `(IN F4)`. (Here the "`,`" prefix indicates a substitution of the variable's value.) `F5` then triggers a rule for interpreting `PRESUMABLY`, which asserts

```
F6: (NOT (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X)))))
```

with antecedent `(AND (IN F5) (NEGOUT F6))`.

We now ask if there are any fallible Greeks.

```
F7: (QUERY (AND (FALLIBLE ?X) (GREEK ?X)))
```

This triggers a rule for handling queries by translating them into goals with the appropriate antecedents. In this case, the rule asserts

```
F8: (PRESUMABLY
     (NOT (SATISFIED
           (GOAL (DEDUCE (AND (FALLIBLE ?X) (GREEK ?X)))))))
```

with antecedent `(IN F7)`, causing

```
F9: (NOT (SATISFIED
          (GOAL (DEDUCE (AND (FALLIBLE ?X) (GREEK ?X))))))
```

to be asserted with antecedent `(AND (IN F8) (NEGOUT F9))`, then causes

```
F10: (SATISFIED (GOAL (DEDUCE (AND (FALLIBLE ?X) (GREEK ?X)))))
```

to be asserted with no antecedents, and then asserts

```
F11: (GOAL (DEDUCE (AND (FALLIBLE ?X) (GREEK ?X))))
```

with antecedent `(AND (IN F7) (NEGIN F10))`. `F11` now triggers a rule for reducing conjunctive goals to rules. This rule asserts the new rule

```
R3: IN( (FALLIBLE ?X) ) ∧ IN( (GREEK ?X) ) ∧ IN(F11)
        ⟹
    assert (AND (FALLIBLE ,X) (GREEK ,X))
        with antecedent
        (AND (IN (FALLIBLE ,X)) (IN (GREEK ,X)))
    assert (SATISFIED (GOAL (DEDUCE (AND (FALLIBLE ?X)
                                         (GREEK ?X)))))
        with antecedent
        (AND (IN (FALLIBLE ,X)) (IN (GREEK ,X)))
```

and asserts, or causes the assertion of

```
F12: (PRESUMABLY
      (NOT (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X))))))
        with antecedent (IN F11)
F13: (PRESUMABLY
      (NOT (SATISFIED (GOAL (DEDUCE (GREEK ?X))))))
        with antecedent (IN F11)
F14: (NOT (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X)))))
        with antecedent (AND (IN F12) (NEGOUT F14))
F15: (SATISFIED (GOAL (DEDUCE (FALLIBLE ?X))))
        with no antecedent
F16: (NOT (SATISFIED (GOAL (DEDUCE (GREEK ?X)))))
        with antecedent (AND (IN F13) (NEGOUT F16))
F17: (SATISFIED (GOAL (DEDUCE (GREEK ?X))))
        with no antecedent
F18: (GOAL (DEDUCE (FALLIBLE ?X)))
        with antecedent (AND (IN F11) (NEGIN F15))
F19: (GOAL (DEDUCE (GREEK ?X)))
        with antecedent (AND (IN F11) (NEGIN F17)).
```

The new goal `F18` now triggers `R2`. Since `F14` is *in*, `R2` is fully triggered, and asserts

```
F20: (GOAL (DEDUCE (HUMAN ?X)))
F21: (PRESUMABLY (NOT (SATISFIED (GOAL (DEDUCE (HUMAN ?X))))))
F22: (OPERATIONAL F4),
```

each with `(AND (IN F18) (IN F4) (NEGIN F15))` as antecedent. `R1` had previously been triggered by `F2` and `F3`, but could not proceed due to the lack of `F22`. Now with `F22` asserted, `R1` asserts

```
F23: (FALLIBLE SOCRATES)
F24: (FALLIBLE TURING),
```

with the respective antecedents `(AND (IN F2) (IN F4))` and `(AND (IN F3) (IN F4))`. `R3` is now fully triggered, and asserts

```
F15: (AND (FALLIBLE SOCRATES) (GREEK SOCRATES))
```

with antecedent `(AND (IN F23) (IN F1))`. `R3` also asserts `F10` with antecedent `(AND (IN F23) (IN F1))`, which changes the status of `F11`'s support, so that `F11` is *out*ed. But `F11` supports the subgoals `F18` and `F19`, and `F18` supports `F20`, and so these are *out*ed also. Thus the query is answered, the goals all *out*ed, and the computation finished.

The primary features of this style of implementing consequent reasoning are that:

1. Goals exist only as long as some supergoal requires their existence: the removal of a goal removes all of the then unsupported subgoals.

2. The computations available in consequent rules are performed only when a question exists making the computations relevant.

3. Deactivation of a rule does not invalidate any consequences of the rule.

4. If goals remain unsatisfied, they can later be satisfied by the addition of new rules generating new facts and so allowing their computations to proceed.

It should be apparent that in many applications a more refined notion of goal state is required than simply `SATISFIED` or `NOT SATISFIED`. The remedy for this is to pass from simple goals to task networks, and to use taxonomies of task states such as those defined in McDermott's [1976] NASL system. The exact form of such mechanisms is still a topic of research.

4 Discussion

4.1 Relation to Other Research

Until the BUILD program of Fahlman [1974], the dominant paradigm embodied in problem-solving programs was that of the MICRO-PLANNER language [Sussman, Winograd and Charniak 1971], a simplified version of the PLANNER language developed by Hewitt [1972]. The problem-solving style captured in MICRO-PLANNER is the style of most problem-solving systems constructed up to its time: heuristic search of AND-OR goal trees constructed by a collection of antecedent and consequent pattern-invoked programs driven by an automatic chronological backtracking control structure. (See Winston [1976] for discussions of these techniques.) In this style of problem solving, embodied in programs like GPS [Ernst and Newell 1969], SAINT [Slagle 1963], Gelernter's [1963] Geometry-Theorem proving machine, Black's [1968] question answerer, STRIPS [Fikes and Nilsson 1971], SHRDLU [Winograd 1972], and Goldstein's [1973] geometry theorem prover, search through different branches of the goal tree have essentially no influence on each other. The major forms of information communicated between subgoals, if any, usually amounted to signals of failure (for instance, the micro-planner FAIL command) or indications of high expected difficulty (as in gps and saint).

A competing problem-solving paradigm of the time was that embodied in resolution theorem provers such as qa3 [Green 1969], a paradigm formalizing the effectively non-deterministic qualities of the micro-planner approach. In these systems, all formulas following from the problem model could theoretically be used once derived. This channel of communication between attempts was usually limited in practice as theorem provers turned to various restricted resolution schemes [Chang and Lee 1973] constraining the use of formulas in resolutions as a method for gaining efficiency, often resulting in the derivation of formulas which could not be used at any point of the solution. In addition, these systems suffered from monotonicity problems [Minsky 1974], as the theorem provers were unable to conclude any results from their inability to derive a formula, and could not decide to remove a proven formula from their set of formulas (although occasionally new axioms could be checked for consistency [Green 1969]). In particular, these theorem provers had no way to make assumptions based on their inability to prove a certain formula. These limitations were primarily due to an enforcement of completeness and consistency at each step. As we have seen, by using dependency relationships between facts it is possible to maintain consistency even in the presence of assumptions.

The first effort to break away from the micro-planner problem-solving paradigm was the build program of Fahlman [1974]. This program used analysis of its own goal structure to correct its mistakes and maneuver out of blind alleys. The techniques of reasoning about relationships involving the reasons for goals and plan steps were then further developed in hacker [Sussman 1975], mycroft [Goldstein 1974], and noah [Sacerdoti 1975]. These programs demonstrated certain information which, if recorded during the problem solving process, allowed simple algorithms for the detection and correction of a number of mistakes arising from the use of simple planning heuristics. Fikes and Nilsson [1971] and Waldinger [1975] also present representations for plan steps which allow the construction of new plans from old ones by reasoning about relationships among the old plan steps.

Analysis of the reasoning involved in the construction of plans facilitates the processes of replanning and execution monitoring. This type of reasoning is used to good advantage in the work of Hayes [1975], who presents an explicit representation for some of the reasoning involved in planning, and uses this information in replanning upon discovering obstructing changes in the world. Similar information is made explicit in nasl [McDermott 1976]. noah also retains a complex representation of the structure of a plan which is modified and interrogated to check plan execution progress and to replan from mistakes. strips and planex [Fikes and Nilsson 1971] also use a complex representation of the structure of plans in execution monitoring and replanning, but this representation is less informative and thus less powerful than the representation employed by noah. Although strips and noah maintain a record of plan structure, unlike Hayes' system and nasl they do not also represent explicitly the reasoning involved in creating the plans and thus require different and more extensive analyses of the plans to modify them. It is interesting to compare these approaches to those of build and hacker, which both plan by a process of continual simulated execution monitoring and replanning. hacker, however, represents little more than a program for the plan itself explicitly, forcing it to do a considerable amount of analysis to discover the nature of its errors. build represents even less information explicitly, and relies on the "hairy control structure" of CONNIVER [McDermott and Sussman 1972] to control the analysis and correction of errors.

Reasoning about the reasoning involved in the design and internal structure of plans, devices or beliefs plays a great role in current work on failure localization and debugging. These problems have been or are being investigated in the contexts of reconciling beliefs, debugging knowledge bases, troubleshooting electronic circuits, and constructing, explaining and debugging computer programs. TOPLE [McDermott 1974] reconciles its conflicting beliefs by making assumptions based on the nature of the conflict, and later analyzes the relationships it has assumed if it finds cause to doubt the truth of a related fact. MYCIN [Shortliffe 1975, Davis 1976] employs a system of dependencies among its beliefs, goals and rules which admits thorough explanations of reasons for program behavior to a human expert adding to or locating deficiencies in the program knowledge base. WATSON [Brown and Sussman 1974, Brown 1974, 1975] uses knowledge of the teleology of circuits (knowledge of the reasoning behind the design of the circuits) to localize failures of circuits to the failure of specific circuit components. INTER [de Kleer 1976b] also localizes failures in electronic circuits, but reasons about the relationships among its assumptions about component behavior to determine component failures, rather than relying on knowledge of the teleology of circuits.

Analysis of program assumptions and their relation to other program knowledge has recently seen substantial application in the dependency-directed backtracking scheme developed in ARS. Another backtracking scheme which analyzes the relations between the choices involved in a failure is used in the graphical deduction system of Cox and Pietrzykowski [1976]. Latombe [1976] employs reasoning about dependencies in choosing choice points for backtracking purposes, but does not explicitly describe the exact mechanisms used.

Analysis of past and present reasoning can also be used in the control of current and future reasoning. The MYCIN system employs a hierarchy of meta-rules for this purpose. In this system of production rules, a first level of meta-rules is used to control the application of the regular production rules. Successive levels of meta-rules are each used to control the application of meta-rules at the next lower level. In this way strategies, hints for choosing strategies, and higher level control information may be encoded. (In actuality, MYCIN at present contains only first level meta-rules. Although its implementation permits higher level meta-rules to be used, none have been formalized to date. In addition, the form of rules and meta rules used in MYCIN is very restricted, which simplifies the processes of explanation and acquisition, but limits the flexibility of the system.) In addition to controlling the current reasoning tasks, the NASL system provides elegant tools for interpreting the current state of reasoning and for specifying both future tasks and continuing policies for controlling task execution. NASL embodies taxonomies for the description of partially ordered task networks, providing semantics for a generalization and extension of Sacerdoti's [1975] procedural nets. (See, for example, the implementation of a NOAH-like Blocks World planner in NASL [McDermott 1976, Doyle and McDermott, forthcoming].) As well as specifying the semantics of task networks, NASL provides basic tools for modifying the task network and the the relationships among tasks in several ways.

Reasoning about past reasoning and careful recording and use of dependency relationships among facts also work together to aid problem-solving in worlds with influential actions. Classic approaches to this problem have included specifying frame axioms for all actions and conditions [McCarthy and Hayes 1969, Raphael 1971], specifying add and delete lists for operators [Fikes and Nilsson 1971], embedding add and delete lists in simulating procedures [Fahlman 1974, Sussman 1975], distinguishing between primary and secondary knowledge [Fahlman 1974, Fikes 1975], using demons to embody tendencies [Rieger 1975] and change-observers [Rieger 1976]. Automatically maintained dependency relationships among facts work smoothly with most of these approaches, providing easy access to possibly invalidated knowledge to a program confronted with contradictions, and a simple system for updating the program knowledge upon recognizing change. Such mechanisms would seem to allow considerably easier access to suspicious assumptions than other representations of world states, such as those of Waldinger [1975], Kowalski [1973], Warren [1974], Hewitt [1975], and Sacerdoti [1975].

In addition to their uses in reasoning about influential actions, dependency records resulting from the reasoning process can also be used to easily work with hypothetical worlds. A dependency based context scheme, such as that employed in ARS provides a convenient channel by which reasoning in distinct hypothetical worlds can interact and supplement the general body of knowledge. Facts derived while investigating a hypothetical world may also be valid in the non-hypothetical world, and thus will not be lost if the hypotheses are abandoned. In addition, the consistent merging of two distinct contexts becomes a trivial (and invisible) operation, as opposed to the difficulties it entails in CONNIVER-type context mechanisms for hypothetical reasoning [Waldinger 1975, McDermott 1975].

Appropriately recorded dependency information also simplifies the construction of articulate, self-explaining experts. SHRDLU [Winograd 1972] answers certain types of questions by examining relationships between past and present goals and subgoals. NOAH uses similar information to answer questions about purposes and methods. MYCIN analyzes recorded dependency relationships to produce explanations of its reasoning, a feature significantly aiding in the processes of reassuring human experts and detecting and correcting knowledge base errors and deficiencies. The EL-ARS system provides a similar power of explanation, but makes no attempt to give its explanations in English.

4.2 Conclusions

Non-chronological dependency relationships can be easily recorded in an antecedent reasoning framework. These dependency relationships can be used in many ways; for explanation, as a means towards maintaining consistency in a database, and most importantly, as tool to use in the control of the reasoning process. In particular, such dependency relationships can be used to effect the efficient control method of dependency-directed backtracking, and can be used to implement non-chronological consequent reasoning in an antecedent reasoning framework.

Acknowledgements: The content of this work has improved through advice from and discussions with Johan de Kleer, Scott Fahlman, Ben Kuipers, Tomas Lozano-Perez, Drew McDermott, Richard Stallman, Guy L. Steele Jr., Gerald Jay Sussman, and Kurt VanLehn. Guy Steele also provided editorial help. This research is supported by a Fannie and John Hertz graduate fellowship.

Research reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract N00014-75-C-0643.

BIBLIOGRAPHY

[Black 1968]
Fischer Black, "A Deductive Question Answering System," in Minsky, *Semantic Information Processing*, pp. 354-402.

[Brown 1974]
Allen L. Brown, "Qualitative Knowledge, Causal Reasoning, and the Localization of Failures," MIT AI Lab, Working Paper 61, March 1974.

[Brown 1975]
Allen Brown, "Qualitative Knowledge, Causal Reasoning, and the Localization of Failures," MIT EECS Ph.D. Thesis, 1975.

[Brown and Sussman 1974]
Allen L. Brown and Gerald Jay Sussman, "Localization of Failures in Radio Circuits: A Study in Causal and Teleological Reasoning," MIT AI Lab, AI Memo 319, December 1974.

[Chang and Lee 1973]
Chin-Liang Chang and Richard Char-Tung Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[Cox and Pietrzykowski 1976]
Philip T. Cox and T. Pietrzykowski, "A Graphical Deduction System," Department of Computer Science Research Report CS-75-35, University of Waterloo, July 1976.

[Davis 1976]
Randall Davis, "Applications of Meta Level Knowledge to the Construction, maintenance and Use of Large Knowledge Bases," Stanford AI Lab Memo AIM-283, July 1976.

[de Kleer 1976a]
Johan de Kleer, "Domain Specific Control of Search in Predicate-Calculus Theorem Provers," informal draft, MIT AI Lab, April 26, 1976.

[de Kleer 1976b]
Johan de Kleer, "Local Methods for Localization of Faults in Electronic Circuits," MIT AI Lab Working Paper 109, August, 1976.

[Dijkstra 1976]
Edsger W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Doyle 1976]
Jon Doyle, "Analysis by Propagation of Constraints in Elementary Geometry Problem Solving," MIT AI Lab, WP 108, June 1976.

[Ernst and Newell 1969]
George W. Ernst and Allen Newell, *GPS: A Case Study in Generality and Problem-Solving*, Academic Press, New York, 1969.

[Fahlman 1974]
Scott E. Fahlman, "A Planning System for Robot Construction Tasks," *Artificial Intelligence*, *5* (1974), pp. 1-49.

[Fikes 1975]
Richard E. Fikes, "Deductive Retrieval Mechanisms for State Description Models," *IJCAI4*, September 1975, pp. 99-106.

[Fikes and Nilsson 1971]
R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence 2*, (1971), pp. 189-208.

[Fikes, Hart and Nilsson 1972]
Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, (Winter 1972) pp. 251-288.

[Gelernter 1963]
H. Gelernter, "Realization of a Geometry-Theorem Proving Machine," in Feigenbaum and Feldman, *Computers and Thought*, pp. 134-152.

[Goldstein 1973]
Ira Goldstein, "Elementary Geometry Theorem Proving," MIT AI Lab, AI Memo 280, April 1973.

[Goldstein 1974]
Ira P. Goldstein, "Understanding Simple Picture Programs," MIT AI Lab, TR-294, September 1974.

[Goldstein and Miller 1976]
Ira P. Goldstein and Mark L. Miller, "Structured Planning and Debugging: a Linguistic Approach to Problem Solving," MIT AI Lab, Working Paper 125, June 1976.

[Green 1969]
C. Cordell Green, "Theorem-Proving by Resolution as a Basis for Question-Answering Systems," in Meltzer and Michie, *Machine Intelligence 4*, pp. 183-205.

[Hayes 1975]
Philip J. Hayes, "A Representation for Robot Plans," *IJCAI4*, September 1975, pp. 181-188.

[Hewitt 1972]
Carl E. Hewitt, "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," MIT AI Lab, TR-258, April 1972.

[Hewitt 1975]
Carl Hewitt, "How to Use What You Know," *IJCAI4*, September 1975, pp. 189-198.

[Kowalski 1973]
Robert Kowalski, "Predicate Logic as a Programming Language," University of Edinburgh, Department of Computational Logic, DCL Memo 70, 1973.

[Latombe 1976]
Jean-Claude Latombe, "Artificial Intelligence in Computer-Aided Design: The TROPIC System," SRI AI Center, TN-125, February 1976.

[McCarthy and Hayes 1969]
J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in Meltzer and Michie, *Machine Intelligence 4*, pp. 463-502.

[McDermott 1974]
Drew Vincent McDermott, "Assimilation of New Information by a Natural Language-Understanding System," MIT AI Lab, AI-TR-291, February 1974.

[McDermott 1975]
Drew V. McDermott, "Very Large PLANNER-type Data Bases," MIT AI Lab, AI Memo 339, September 1975.

[McDermott 1976]
Drew Vincent McDermott, "Flexibility and Efficiency in a Computer Program for Designing Circuits," MIT EECS Ph.D. Thesis, September 1976.

[McDermott and Sussman 1974]
Drew V. McDermott and Gerald Jay Sussman, "The CONNIVER Reference Manual," MIT AI Lab, AI Memo 259a, January 1974.

[Minsky 1974]
Marvin Minsky, "A Framework for Representing Knowledge," MIT AI Lab, AI Memo 306, June 1974.

[Moore 1975]
Robert Carter Moore, "Reasoning From Incomplete Knowledge in a Procedural Deduction System," MIT AI Lab, AI-TR-347, December 1975.

[Nevins 1974]
Arthur J. Nevins, "Plane Geometry Theorem Proving Using Forward Chaining," MIT AI Lab Memo 303, January 1974.

[Raphael 1971]
B. Raphael, "The Frame Problem in Problem Solving Systems," in *Artificial Intelligence and Heuristic Programming*, pp. 159-169, (eds. N. V. Findler and B. Meltzer) Edinburgh University Press, 1971.

[Rieger 1975]
Chuck Rieger, "One System for Two Tasks: A Commonsense Algorithm Memory that Solves Problems and Comprehends Language," MIT AI Laboratory, Working Paper 114, November 1975.

[Rieger 1976]
Chuck Rieger, "Spontaneous Computation in Cognitive Models," Department of Computer Science TR-459, University of Maryland, July 1976.

[Robinson 1965]
J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *JACM*, *12* (January 1965), pp. 23-41.

[Sacerdoti 1975]
Earl D. Sacerdoti, "A Structure for Plans and Behavior," SRI AI Center, TN 109, August 1975.

[Shortliffe 1976]
E. H. Shortliffe, *MYCIN: Computer-Based Medical Computations*, American Elsevier, 1976.

[Slagle 1963]
James R. Slagle, "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus," in Feigenbaum and Feldman, *Computers and Thought*, pp. 191-203.

[Stallman and Sussman 1976]
Richard M. Stallman and Gerald Jay Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," MIT AI Memo 380, September 1976.

[Sussman 1975]
Gerald Jay Sussman, *A Computer Model of Skill Acquisition*, American Elsevier Publishing Company, New York, 1975.

[Sussman and McDermott 1972]
Gerald Jay Sussman and Drew Vincent McDermott, "From PLANNER to CONNIVER - A genetic approach," *Proc. AFIPS FJCC*, 1972, pp. 1171-1179.

[Sussman and Stallman 1975]
Gerald Jay Sussman and Richard Matthew Stallman, "Heuristic Techniques in Computer-Aided Circuit Analysis," *IEEE Transactions on Circuits and Systems*, Vol. CAS-22, No. 11, November 1975, pp. 857-865.

[Sussman, Winograd and Charniak 1971]
Gerald Jay Sussman, Terry Winograd and Eugene Charniak, "MICRO-PLANNER Reference Manual," MIT AI Lab, AI Memo 203a, December 1971.

[Waldinger 1975]
Richard Waldinger, "Achieving Several Goals Simultaneously," SRI AI Center, Technical Note 107, July 1975.

[Warren 1974]
David H. D. Warren, "WARPLAN: A System for Generating Plans," University of Edinburgh, Department of Computational Logic Memo No. 76, June 1974.

[Winograd 1972]
Terry Winograd, *Understanding Natural Language*, Academic Press, New York, 1972.

[Winston 1976]
Patrick Henry Winston, *Artificial Intelligence*, MIT Artificial Intelligence Laboratory, September, 1976.