CHOICES WITHOUT BACKTRACKING

Johan de Kleer
Intelligent Systems Laboratory
XEROX Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

ABSTRACT

Artificial Intelligence problem solvers are frequently confronted with the necessity to make choices among equally plausible alternatives. These may concern the choice of which goal to try to achieve next, which priority to assign to a task or which plausible inference to draw from incomplete data. Choices can be wrong: later problem solving may determine that an earlier choice was incorrect. In such cases most problem solvers invoke some form of backtracking to undo the faulty choice, retract the inferences that were made from that choice and make some other choice. This paper discusses a method of dealing with choice that does not involve any backtracking yet explores no more alternatives than the best backtracking schemes. It has an additional advantage over backtracking schemes that it is possible to easily compare two alternative incompatible choices — this cannot be done in backtracking schemes because of their necessity of requiring a globally consistent set of assertions.

MOTIVATION

The results of this paper are born out of frustration. Qualitative reasoning and constraint languages, the primary topics of my research, both require making choices among alternatives. Various packages for dealing with choice have been developed, primarily derived from (Doyle, 1979) and (McAllester, 1980). These systems have proven to be woefully inadequate for even simple qualitative reasoning tasks. The reasons for this are twofold. First, they are intrinsically incapable of working with multiple contradictory choices at once — something one needs to do all the time in qualitative reasoning. Second, they are very inefficient in both time and space. Very simple problems fill up all the memory of Symbolics LM-2 or 3600 in short order (reported by various researchers in qualitative reasoning (Forbus, 1982) (de Kleer and Brown, 1984) (Williams, 1983)). Timing analysis shows that the reasoner spends the majority of its time in the backtracking algorithms. The term "non-monotonic" reasoning is a misnomer as far as memory space is concerned: the number of justifications grows monotonically as problem solving proceeds.

This paper presents an alternative solution to using a general choice mechanism such as a TMS. It is the result of carefully analyzing the kind of backtracking actually needed for a class of problem-solving tasks and designing a matching choice mechanism. This mechanism is as general as any TMS for the task, can handle multiple contradictory choices at once, is extremely efficient. The technique is appropriate for tasks in which: (a) as in a standard TMS it must be possible to attribute all conclusions to a small set of antecedents it depends on, otherwise no TMS can do much good; (b) the user is interested in many or all of the solutions which achieve the goal (if one is only interested in a single solution, a standard TMS is probably better); (c) few combinations of assumptions are consistent; (d) there are finitely many (but not necessarily bounded) solutions and choices.

Additionally, the proposed scheme performs better if: (e) the number of alternatives for each choice are finite and exclusive; however, there is no necessity for the choices, the number of choices or the alternatives of the choices to be known a priori; (f) there is no single solution which requires infinite time to explore (a standard TMS can often be controlled to avoid such holes). These six requirements hold for many kinds of constraint satisfaction problems, and in particular for qualitative reasoning[1]. The strategy proposed in this paper has been implemented and used successfully in (de Kleer, 1979) and (de Kleer and Brown, 1984) with great success (time spent handling assumptions is in the noise). It is also used as the mechanism for handling disjunction in a constraint language under development.

ARCHITECTURAL ASSUMPTIONS

For the purposes of this paper an extremely simplified model of problem solving suffices. The reasoning system consists of a procedure for performing problem solving, and a data base for recording the state of the problem-solving process. Most of the problem-solving task consists of deriving new inferences from data and previously made inferences. All these are added to the data base. Sometimes the problem solver must make choices among which there is no preferred option. Each choice may involve substantial additional work before it proves to be contradictory or fruitless. However, a choice must be made for problem solving to proceed. If a choice is subsequently discovered to be incorrect, problem solvers typically backtrack to some previous alternative. In doing so the data base must be updated by erasing those entries which no longer hold.

---

[1] A typical "easy" problem taking a few minutes total on a LM-2 involves tens of thousands of new assertions, thousands of assumptions and hundreds of solutions. If a conventional TMS were used for the same problem (this is hypothetical because a TMS has certain conceptual limitations which make it nearly impossible to use for qualitative reasoning), it would either exhaust the address space of the machine (if ining or outing were not handled carefully), or fail to converge for hours.

The subject of this paper is how this process can be made more efficient. Without additional information, choice cannot be avoided. However, improvements can be made in the preservation of relevant information when a choice is retracted. Suppose the problem solver has produced data-base state S_1 using $\{A, B, C\}$ and subsequently explores implications of choices $\{A, B, D\}$. The question is how much of state S_1 are valid in the data-base state that would result if the consequences of choices $\{A, B, D\}$ were explored. This problem can be viewed as an "internal" version of the classic AI frame problem: how much of the description of the problem-solving state is changed when some action is performed.

TECHNIQUES FOR DEALING WITH CHOICE

Consider the task of the problem solver which performs the following sequence. (It must first do one of A or B, then one of C or D, then one of E or F, and then G. Admittedly, any well-designed problem solver would do G first as it doesn't require a choice — but this ordering best brings out the issues I want to address.)

$$A \vee B$$

$$C \vee D$$

$$E \vee F$$

$$G$$

Assume the disjunctions are exclusive and the letters represent potentially complex operations.

The simplest and most often used strategy is exponential: Enumerate all the possibilities and try each one until a solution is found (or all solutions are found): $\{A, C, E, G\}$ $\{A, C, F, G\}$ $\{A, D, E, G\}$ $\{A, D, F, G\}$ $\{B, C, E, G\}$ $\{B, C, F, G\}$ $\{B, D, E, G\}$ $\{B, D, F, G\}$
Usually many of the combinations are inconsistent. And often these inconsistencies can be detected in small subsets of choices. This suggests a modification of the brute-force enumeration where each option is attempted, but whenever an inconsistency is detected, the problem solver backtracks to the most recent choice. If $\{A, C\}$ and $\{D, F\}$ are inconsistent the search order is:

$\{A\}$ $\{A, C\}$ $\{A, D\}$ $\{A, D, E\}$ $\{A, D, E, G\}$ $\{A, D, F\}$ $\{B\}$ $\{B, C\}$ $\{B, C, E\}$ $\{B, C, E, G\}$ $\{B, C, F\}$ $\{B, C, F, G\}$ $\{B, D\}$ $\{B, D, E\}$ $\{B, D, E, G\}$ $\{B, D, F\}$

The advantage of this technique is that it explores far fewer complete sets of options than brute-force enumeration. In this example, it explores four complete sets, while enumeration explores eight. This strategy is called chronological backtracking and one of its early applications was in QA4 (Rulifson, Derkson and Waldinger, 1972).

Chronological backtracking has three serious faults which result in it doing far more work than necessary. Consider the case where $\{C, G\}$ is inconsistent. Chronological backtracking will search in the following order:

$\{A, C, E, G\}$ $\{A, C, F\}$ $\{A, C, F, G\}$ $\{A, D\}$ ... $\{B, C, G\}$

The second two steps are futile. As G is inconsistent with the choice of C from $C \vee D$, backtracking to the last choice, i.e., E from $E \vee F$ has no effect as it has no influence on the contradiction. When a contradiction is discovered the search should backtrack to a choice which contributed to the contradiction, not to the most recent choice. The second defect is illustrated by the third and last step. Once it is discovered that $\{C, G\}$ is inconsistent the problem solver should never explore that possibility again. The final defect of chronological backtracking is that it requires more problem solving operations than necessary. Suppose that choosing the combination $\{E, G\}$ results in a great deal of the problem-solving effort that is solely dependent on E and G. Chronological backtracking might search the space of assumption sets as:

$\{B, C, E, G\}$ $\{B, D\}$ $\{B, D, E, G\}$

In doing so it will do any computations involving both E and G twice. It will add the inferences resulting from $\{E, G\}$ to the data base, then backtrack to $\{B, D\}$ erasing the inferences, and finally rederive these erased inferences while exploring $\{B, D, E, G\}$.

A solution to all three of these defects is enabled by maintaining records of the dependence of each inference on earlier ones. When a contradiction is encountered these dependency records are consulted to determine which choice to backtrack to. Reconsider the example where C is inconsistent with G when such records are available. The dependency records state that G is given, but C is a choice from $C \vee D$. Thus the problem solver should backtrack to the choice $C \vee D$. In general, the problem solver should immediately backtrack to the most recent choice which influences the contradiction. This technique is called *dependency-directed backtracking.*

Whenever a contradiction is discovered the dependency records are consulted to determine which choices caused the contradiction, so this choice can be avoided in the future. These are called the *nogood* sets (Steele, 1979) as they represent choices which are mutually contradictory.

Dependency records also solve the third problem of chronological backtracking (illustrated by working on $\{E, G\}$ twice). The dependence of b on a is recorded with b; but b is a consequence of a and this is also recorded. Thus, whenever some option is included in the current set, the dependency records can be consulted to determine the consequences of those options. Thus, the consequences of $\{E, G\}$ need only be determined once. This can be affected quite directly within the data base. Entries are marked as temporarily unavailable (i.e., out) if they are not derivable from the set of choices currently being explored.

These techniques are the basis of the TMS strategies of (Doyle, 1979) and (McAllester, 1980). In the more general TMS strategies it is not necessary to specify the overall ordering of the search space (so far we have been presuming some simple-minded enumeration algorithm). They, in effect, choose their own enumeration but this ordering can be controlled somewhat by specifying which parts of the search space to explore first.

It is important to note that all these strategies are equivalent in the sets of options that are explored. The most sophisticated TMS will find as many consistent solutions as pure enumeration. The goal

is to enhance efficiency without sacrificing completeness.

PROBLEMS WITH USING TMS

Truth maintenance systems are the best general-purpose mechanism for dealing with choice. However, they have certain limitations which in appropriate circumstances can be avoided.

The single state problem. Given a set of choices which admits multiple solutions, the TMS algorithms only allow one solution to be considered at a time. This makes it extremely difficult to compare two equally plausible solutions. For example, suppose A, D, E, G and B, C, E, G are both solutions. It is impossible to examine both of these states simultaneously. However, this is often exactly what one wants to do in problem solving — differential diagnosis to determine the best solution.

Overzealous contradiction avoidance. Suppose A and B are contradictory. In this case, the TMS will guarantee that if A is believed, B will not be, and if B is believed, A will not be. This is not necessarily the best problem-solving tactic. All a contradiction between A and B indicates is that any inference dependent on both A and B is of no value. But it is still important to draw inferences from A and B independently. A discovery of a contradiction between A and B will result in one of A or B being abandoned until another contradiction is encountered.

Switching states is difficult. Suppose that the problem solver decides to temporarily change a choice (i.e., not in response to a contradiction). There is no convenient mechanism to facilitate this. The only direct way to change the current choice set is to introduce a contradiction, but once added it cannot be removed so the knowledge state of the problem solver is irreconcilably altered. Suppose the change of state was somehow achieved. There is no way to specify the target state. All a TMS can guarantee is that it is contradiction-free. So, in particular, there is no way to go back to a previous state. The reason for these oddities is that a TMS has no useful notion of global state. All TMS guarantees is that each justification is satisfied in some way. One inelegant mechanism that is sometimes utilized to manipulate states is to take snapshots of the status and justifications of each assertion and then later reset the entire database from the snapshot. This approach is antithetical to the spirit of TMS for it reintroduces chronological backtracking. Information garnered within one snapshot is not readily transferred to another.

The dominance of justifications. A TMS solely uses justifications, not assumptions. Furthermore, what (Doyle, 1979) calls an assumption is context-dependent: an assumption is any node whose current supporting justification depends on some other node being out. Thus, as problem solving proceeds the underlying support justifications and hence the assumptions underlying assertions change. This is particularly problematic for problem solvers which must often consult the assumptions and justifications for assertions.

The machinery is cumbersome. The TMS algorithm, partly because it is very general often spends a surprising amount of time to find a solution that satisfies all the justifications. A particularly expensive operation can result from the dependency-directed backtracking in response to a contradiction. The backtracking may require extensive search, and the resolution of the contradiction often results in other contradictions. Eventually, all contradictions are resolved, but only after much backtracking. During this time the status of some assertion may have changed between in (believed) and out (not believed) many times.

Unouting. Suppose the option set $\{B, C, E, G\}$ is explored, a contradiction is discovered and then some time later the option set $\{B, D, E, G\}$ is explored. The use of dependency records assures that the inferences derived from $\{B, E, G\}$ in the first set $\{B, C, E, G\}$ will carry through to the second set $\{B, D, E, G\}$ without additional computation. The situation is unfortunately not so simple. Suppose that in this example the set $\{C, E, G\}$ is contradictory but that this was not discovered until after extensive problem solving effort. Once the contradiction is discovered there is no longer any point to working on the current state and therefore dependency-directed backtracking begins. Work on the situation $\{B, C, E, G\}$ was never permitted to go to conclusion. In particular, not all inferences based on $\{E, G\}$ may have been made. When the set $\{B, D, E, G\}$ is explored the earlier derived consequences of $\{E, G\}$ can be included, but problem solving must continue for $\{E, G\}$. The difficult task, for which TMS is of no aid, is how to fill the "gaps" of the consequences of $\{E, G\}$ without redoing the entire computation.

There are four styles of solutions to this task, none completely satisfactory. (1) Even though a contradiction occurs during analysis of $\{E, G\}$ this computation could be allowed to go on. The difficulty here is that a great deal of effort may be spent working on a set of choices that may be irrelevant to an overall solution. (2) Another technique is to store a snapshot of the problem-solver's state (its pending task queue) which can be reactivated at a later time. (3) It is also possible to restart the computation from $\{E, G\}$, taking advantage of the previous result by looking at the consequences and examining which ones are missing. (4) The easiest technique is just to restart the computation from $\{E, G\}$ without taking any effort to see which consequences should be unouted. All expensive problem-solving steps are memoized so no time-consuming steps are repeated. For example, (de Kleer and Sussman, 1980) uses this technique to cache all symbolic GCD computations.

A GENERAL SOLUTION

My solution is to include with each assertion, in addition to its justifications, the set of choices (assumptions) under which it holds. For example, each assertion derived from assumption A is labeled with the set $\{A\}$, each assertion derived from both assumptions A and B is labeled with the set $\{A, B\}$. Thus if $x = 1$ under assumption A and $x + y = 0$ under assumption B then we deduce $y = -1$ under assumption set $\{A, B\}$. (For clarity, I'll call the combination of an assertion with its assumptions and justifications a *value* and notate it: $\langle assertion, \{assumptions\}, justification(s)\rangle$. Thus, the preceding inference is written as: if $\langle x = 1, \{A\},\rangle$ and $\langle x + y =$

$0, \{B\},>$, then $<y = -1, \{A, B\},>$.) Unlike a TMS where the same node can be brought in and out an arbitrary number of times, a value is removed from the data base only if its assumption set is found to be contradictory. For example, the database can contain both $<x = 1, \{A\},>$ and $<x = 0, \{B\},>$ without difficulty. $x = 1$ contradicts $x = 0$ but this provides no information about $x = 1$ or $x = 0$ individually. However, these two values imply that the assumption set $\{A, B\}$ is contradictory. Thus, if the database contained $<z + q = 0, \{A, B\},>$ it would be removed[2] because the set $\{A, B\}$ is contradictory. As this scheme is primarily based on assumptions, not justifications I term it *assumption-based* as opposed to *justification-based* TMS systems.

Abstractly, one possible[3] mode of interaction between the the problem solver and the data base is as follows. A list is maintained of every assumption set discovered to be contradictory. Whenever the problem solver discovers two values with contradictory assumptions, the combined assumption set is placed on this list, and every value based on it or any of its super sets is erased from the data base. Suppose every problem-solving step can be formulated as: from a and b determine $f(a, b)$ where f takes some problem solving work. Then the interaction with the data base of values should be: for all values of form $\alpha : <a, A_a,>$ and $\beta : <b, A_b,>$ add value $<f(a, b), A_a \bigcup A_b, (\alpha, \beta),>$ unless the assumption set $A_a \bigcup A_b$ a superset of some known contradictory set of assumptions. Note that this scheme does not have or require any notion of context. The equivalent notion in the assumption-based scheme is just a set of assumptions: implicitly, a set of assumptions selects all those values whose assumption set is a subset of the context's assumption set.

It is not necessary to be this extreme. A more sophisticated mode of interaction would be to explore only a part of the solution space, i.e., only perform inferences using those values whose assumptions are a subset of the current set of interesting assumptions. Then, when a contradiction is discovered, the set of interesting assumptions is changed but nothing is done to the data base.

These are just two of many possible modes of interaction between the problem solver and the data base. Regardless of the mode of interaction, the basic assumption-based solution addresses the problems discussed earlier:

The single state problem. The assumption-based scheme allows arbitrarily many contradictory solutions to coexist. Thus, it is simple to compare two solutions.

Overzealous contradiction avoidance. The presence of two contradictory assertions does not terminate work on the overall knowledge state, rather only those assertions are removed which depend on the two contradictory assertions. This is exactly the result desired from a contradiction — no more, no less.

Switching states is difficult. Changing state is now trivial or irrelevant. A state is completely specified by a set of assumptions. Problem solving can be restricted to a current context (i.e., a set of assumptions) or all states can be explored simultaneously. In either case, values obtained in one state are "automatically" transferred to another. For example, if $<x = 1, \{E, G\},>$ is deduced while exploring $\{B, C, E, G\}$, then $x = 1$ will still be present while exploring $\{B, D, E, G\}$

The dominance of justifications. As assumptions, not justifications are the dominant representational mode it is easy to compare sets of assumptions underlying assertions. For example, it is easy to find the assertion with the most assumptions or the least; it is easy to determine whether the presence of an assertion implies the presence of another (a implies b if the assumptions of b are a subset of the assumptions of a). Also the justifications underlying a value never change.

The machinery is cumbersome. The underlying mechanism is simple. There is no backtracking of any kind — let alone dependency-directed backtracking. The assumptions underlying a contradiction are directly identifiable. A value once added is never removed unless it is removed permanently, thus it is not necessary to explicitly mark entries as believed or disbelieved.

Unouting. The unouting problem is partially resolved. Consider the analog to the TMS problem of preserving assertions while moving from state $\{B, C, E, G\}$ to $\{B, D, E, G\}$ in response to a contradiction. In the simplest assumption-based scheme, all possibilities are explored simultaneously. Hence, a contradiction within $\{B, C, E, G\}$ merely implies that any exploration of that state ceases, i.e., any inferences involving sets which contain $\{B, C, E, G\}$ as a subset are avoided. Work on state $\{B, D, E, G\}$ continues as if the contradiction never occurred. Assertions derived from $\{E, G\}$ are automatically part of every superset hence are also part of $\{B, D, E, G\}$ (and the contradictory $\{B, C, E, G\}$ for that matter).

Unfortunately, the assumption-based approach fails to address all of the unouting problem. Suppose that under the assumption $\{E, G\}$ the problem solver has determined that $z = 1$ and has consequently gone through the difficult process of determining

$$\int_0^{z=1} \frac{1}{\sqrt{(1-x)x^2}} = 2\pi\frac{\sqrt{3}}{3}$$

and that $z = 1$ is also deduced under assumptions $\{G, H\}$ (i.e., an independent derivation). Not all consequences of $z = 1$ using $\{E, G\}$ carry over to $\{G, H\}$. In addition there are derivations from $z = 1$ possible under $\{G, H\}$ but not under $\{E, G\}$. Consider a simplified example. Suppose the problem solver deduced $\alpha : <x = 1, \{A\},>$, $\beta : <x + y = 0, \{B\},>$, $\gamma : <z = 1, \{A, B\},>$, and $\lambda : <z = 0, \{A, B\},>$. In this situation $y = -1$ is not derivable from α and β as the set $\{A, B\}$ is contradictory. However, if $\epsilon : <x = 1, \{C\},>$ is discovered later, $<y = -1, \{C, B\}, (\beta, \epsilon)>$ is derivable. Thus, $<x = 1, \{C\},>$ has consequence $y = -1$, but $<x = 1, \{A\},>$ does not. This unouting problem exists whether assumption-based or justification-based techniques are used.

There is an inelegant fix to this problem which we have implicitly adopted earlier in this discussion: two values are considered the same only if both their assertion and their assumptions are the same. This is

---

[2] It is not necessary to remove it, because unlike a conventional logical system a contradiction does not imply everything. For some tasks, as pointed out in (Martins and Shapiro, 1983), there is some utility in deriving further contradictory values.

[3] But extremely inefficient.

contrary to the way TMS's are usually used (two values are the same if their assertion is the same). Thus, the unouting problem produced by simply changing statuses is completely avoided, but unouting problems produced by adding significantly different justifications is still with us.

More research is required to find more elegant solutions. Unouting is a open problem for both approaches. It just shows up in a different place in the assumption-based approach than in the justification approach. Depending on the characteristics of problem-solving task, an system implementor must choose which which inadequacy he can live with.

REDUNDANCY AND INCOHERENCY

The problem solver will invariably discover multiple derivations for some assertions. If the only goal is to identify the statuses of assertions, the problem solver should throw out all values whose assumptions are equal to or a superset of the assumptions of some alternate derivation.

For many tasks the statuses of the assertions are as important as their derivations. For example, causal reasoning carefully analyzes the derivations for quantities to determine device functioning. For such tasks the problem solver cannot be so cavalier in throwing away derivations. Strictly speaking, if the goal is to discover all possible derivations as well as all possible assertions, the problem solver should never throw away any derivation. Unfortunately, there is no guarantee that the rules of inference the problem solver are using are logically independent. Redundancy in the inference rules results in syntactically different but essentially identical derivations. This problem of logical independence is outside of the scope of the TMS, but is one every problem solver which examines derivations must cope with [4]

Another problem that arises if derivations are important is that of incoherency. If the data base contained $\alpha : \langle x + x = y, \{\}, \rangle$, $\beta : \langle x = 1, \{A\}, \rangle$, and $\gamma : \langle x = 1, \{A\}, \rangle$, three values would be deduced: $\langle y = 2, \{A\}, (\alpha, \beta, \beta) \rangle$, $\langle y = 2, \{A\}, (\alpha, \gamma, \gamma) \rangle$, and $\langle y = 2, \{A\}, (\alpha, \beta, \gamma) \rangle$. This last value is incoherent in that its derivation uses x twice with a different derivation for x each time. Thus it should be discarded.

INTERPRETATION CONSTRUCTION

One of the advantages of the assumption-based approach is that the notion of global consistent state does not appear. With the inference algorithm suggested in this paper it is not even known how many globally consistent states, if any, there are. However, at the termination of the problem-solving effort some notion of global consistent state is often required. We call the global choice sets *interpretations* and the process of computing them *interpretation construction*. Most of the complexity of interpretation construction results from the goal of maintaining global coherence and is outside the scope of this paper.

(de Kleer, 1979) and (de Kleer and Brown, 1984) use a very simple technique to manage the construction of interpretations. All non-contradictory inferences are permitted to proceed unchecked. After the data base reaches quiescence a second process is invoked to construct all possible globally consistent states. It can be viewed as a straight-forward set-manipulation algorithm. Its task is to construct maximal sets of assumptions, such that the addition of any assumption results in selecting a contradiction or an incompatibility and the removal of any assumption removes all values for some assertion.

SOME (MORE) COMPUTATIONAL TECHNIQUES

More research is required to determine which implementation techniques are best for assumption-based approaches. Here I present and evaluate some possible implementation options.

The basic data structure is the set and its representation can be optimized (e.g., as cdr-coded lists, arrays or bit-vectors). The same set can be arrived at by unioning many different combinations of other sets. So it is important to (a) uniquize sets, and (b) quickly determine whether the given set has been created earlier. These goals are achieved with a canonical form for sets and a hash table for these canonicalized sets. Thus, for example, once a set of assumptions is determined to be contradictory its unique structure can be marked as such.

The most common operations of the assumption-based algorithms are set operations, therefore they can be optimized by creating an explicit subset/superset lattice such that subset/superset computations can proceed quickly (this is only possible of course if the sets are uniquized). Like the justification-based approaches, assumption-based approaches must somehow record and access the nogood sets. The simplest technique is to maintain a list of all the contradictions and whenever a new set is created by unioning two it is checked to determine whether the new set is a superset of any known nogood set. As sets are uniquized this operation need only be performed once per set. Given a lattice data-structure contradiction manipulation can be surprisingly efficient. As every set is entered into the lattice structure as it is created, the fact that it is a superset of some nogood set is computed by a simple intersection of the nogood sets with the subsets of the new set (which takes linear time for ordered data structures). Furthermore, when a new contradiction is discovered it is a simple matter to mark all its supersets as contradictory and stop all problem solving on them.

However, unless the problem is extremely large (i.e., tens of thousands of assumptions in an LM-2) these advantages do not outweigh the costs of maintaining the data-structures in the first place. In our experience the extra effort incurred in maintaining this data-structure does not turn out to be worth the cost (storing sets as ordered cdr-coded lists or bit-vectors speeds up subset computations sufficiently).

[4] A solution that has worked in practice: A and B are two derivations for the same assertion. Suppose A is derived under assumption set a and B is derived under assumption set b. If a is a proper subset of b, B should be discarded. If a and b are the same, compute all the assertions A and B depend on, call these α and β. If α is a proper subset of β, B should be discarded. Otherwise, both derivations should be kept and used.

(Martins and Shapiro, 1983) uses the technique of marking each assertion with the super-sets of its assumption-set which are nogood. ((Martins and Shapiro, 1983) calls these super-sets the restriction sets and the assumption set the origin set[5]) This has the advantage that it is extremely simple to determine whether a newly created set is contradictory as only these super-sets need be consulted, not the entire set of contradictions. However, whenever a contradiction is discovered an extensive computation must take place to determine whether the restriction sets of any assertion must be updated (this is roughly equivalent to entering a set into a lattice).

Although the ordering of the inferences is irrelevant to completeness it has significant influence on efficiency. For efficiency, the problem solver should work on values with fewer assumptions first. The overall efficiency of the problem solving is roughly proportional to the number of assumptions and inferences (i.e., the number of constructed values). Fortunately, with the computational techniques outlined in this section, performance degrades very slowly with the number assumptions. However, each inference involves a separate problem-solving step, so, roughly speaking, efficiency is linearly related to the number of inferences. This provides a strong motivation for reducing the number of inferences. There are two classes of inferences which are guaranteed to be futile: values whose assumption sets are later discovered to be contradictory and values which are superseded by other values with identical assertions whose assumptions are a subset of the original assumptions. Both of these inferences are avoided by working on values with smaller assumption sets first. One way to achieve this is to introduce new assumptions as late as possible; this ensures that any values following from the new assumption will not be superseded with values with subset assumption sets and no subsets are contradictory.

The exclusive-or between choices produces many contradictions which tend to clutter the contradiction recording mechanism. For example, the choice

$$A \vee B \vee C$$

introduces the four nogood sets $\{A, B\}$ $\{A, C\}$ $\{B, C\}$ and $\{A, B, C\}$. A significant increase in efficiency can be obtained by marking each individual choice with the choice set it is a member of. Thus, two different choices of the same set can be trivially checked to see whether their combinations results in a contradiction. Two choices are *compatible* if they are not members of the same choice set. This approach is used in (de Kleer, 1979) and (de Kleer and Brown, 1984).

RELATED WORK AND SAMPLE IMPLEMENTATIONS

There are many applications for which the architectural proposal of this paper is applicable.

LOCAL (de Kleer, 1976) is a program for troubleshooting electronic circuits which was incorporated in SOPHIE III (Brown, Burton and de Kleer, 1983). It uses propagation of constraints to make predictions about device behavior from component models and circuit measurements. As the circuit is faulted, some component is not operating as intended. Thus, at some point, as the correct component model does not describe the actual faulty component, the predictions will become inconsistent. The assumptions are that individual components are functioning correctly. A contradiction implies that some underlying assumption is violated, hence the fault is localized to a particular component set (i.e., the nogood set). The best measurement to make next is the one that provides maximal information about the validity of the yet unverified assumptions. This program requires that the assumptions of an inference be explicitly available and that multiple contradictory propagations be simultaneously present in the data base. Hence, for this task the assumption-based approach is better.

QUAL (de Kleer, 1979) and ENVISION (de Kleer and Brown, 1984) produce causal accounts for device behavior. QUAL can determine the function of a circuit solely from its schematic. Qualitative analysis is inherently ambiguous, and thus multiple solutions are produced. However, for any particular situation a device has only one function. Qual selects the correct one by explicitly comparing different solutions — something that is only possible using assumption-based schemes.

(Martins and Shapiro, 1983), (McDermott, 1983) and (Barton, 1983) all attempt to unify assumption-based and justification-based approaches. Each of these is powerful enough to formulate the scheme proposed in this paper. However, for many tasks, the complexities and inefficiencies introduced by a general scheme are unnecessary. No matter how making choices is formulated, it is important to first identify the essential problem-solving work it is providing — the topic of this paper.

MBR (Multiple Belief Reasoner) (Martins and Shapiro, 1983) is a general reasoning system which allows multiple, including contradictory and hypothetical, beliefs to be represented simultaneously. It is based on a relevance logic which explicitly takes into account assumptions underlying underlying wffs. In this system multiple agents can interact with the same data base, each individually possessing consistent beliefs, but beliefs that well may contradict beliefs that other agents have entered into the data base.

McDermott (McDermott, 1983) has proposed a very generalized and relatively complicated scheme which unifies assumption-based and justification-based techniques. It uses constraint satisfaction among justifications and the assumptions marking contexts.

XRup (Barton, 1983), an equality-based reasoning system, uses an assumption-based context mechanism instead of the justification-based framework of Rup (McAllester, 1982). As a consequence it has many of the advantages of the assumption-based approach, e.g., easy switching between contexts. Interestingly, XRup's equality mechanism is also used to construct equivalence classes of assumptions and as a consequence it is possible to identify syntactically different but essentially equivalent assumption sets. With the scheme presented in this paper, there is no necessity for contexts and their associated overhead.

[5] In his actual implementation he subtracts out the origin set from each restriction set, but this is conceptually unnecessary.

The inefficiencies of backtracking for constraint satisfaction problems was recognized quite early. (Mackworth, 1977) summarizes the difficulties and proposes a number of efficient techniques. Although not formulated in terms of assertions, justifications, assumptions, his technique explores the solution space nearly as efficiently as TMS-like schemes.

ACKNOWLEDGMENTS

I thank Daniel Bobrow and Brian Williams who helped sort out many of the technical issues and forced me to clean up the implementation.

BIBLIOGRAPHY

[1] Barton, G.E., "A Multiple-Context Equality-based Reasoning System," Artificial Intelligence Laboratory, TR-715, Cambridge: M.I.T., 1983.

[2] Brown, J.S., D. Burton and J. de Kleer, "Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III," in *Intelligent Tutoring Systems*, edited by D.Sleeman and J.S. Brown, Academic Press, 1983.

[3] de Kleer, J. and J.S. Brown, "A Qualitative Physics Based on Confluences," to appear in *Artificial Intelligence.*

[4] de Kleer, J., "Causal and Teleological Reasoning in Circuit Recognition," Artificial Intelligence Laboratory, TR-529, Cambridge: M.I.T., 1979.

[5] de Kleer, J. and G.J. Sussman, "Propagation of Constraints Applied to Circuit Synthesis," *Circuit Theory and Applications*, Vol. 8, 1980.

[6] de Kleer, "Local Methods of Localizing Faults in Electronic Circuits," Artificial Intelligence Laboratory, AIM-394, Cambridge: M.I.T., 1976.

[7] Doyle, J., "A Truth Maintenance System," *Artificial Intelligence,* Vol. 12, No. 3, 1979.

[8] Forbus, K.D., "Qualitative Process Theory," Artificial Intelligence Laboratory, AIM-664, Cambridge: M.I.T., 1982.

[9] Mackworth, A.K., "Consistency in Networks of Relations," *Artificial Intelligence,* Vol. 8, No. 1, 1977.

[10] Martins, J.P. and S.C. Shapiro, "Reasoning in Multiple Belief Spaces," IJCAI-1983, 1983 (see also: Department of Computer Science, Technical Report No. 203, Buffalo, New York: State University of New York, 1983).

[11] McAllester D., "An Outlook on Truth Maintenance," Artificial Intelligence Laboratory, AIM-551, Cambridge: M.I.T., 1980.

[12] McAllester D., "Reasoning Utility Package User's Manual," Artificial Intelligence Laboratory, AIM-667, Cambridge: M.I.T., 1982.

[13] McDermott D., "Contexts and Data Dependencies: A Synthesis," 1983.

[14] Rulifson, J.F., J.A. Derkson, and R.J. Waldinger, "QA4: A Procedural Calculus for Intuitive Reasoning," Artificial Intelligence Center, Technical Note 73, Menlo Park: S.R.I., 1972.

[15] Steele, G., "The Definition and Implementation of a Computer Programming Language based on Constraints," Artificial Intelligence Laboratory, TR-595, Cambridge: M.I.T., 1979.

[16] Williams, B.C., "Qualitative Analysis of MOS Circuits," M.I.T. AI Laboratory TR-567, 1983.