# The Computational Value of Nonmonotonic Reasoning

**Matthew L. Ginsberg**
Computer Science Department
Stanford University
Stanford, CA 94305

## Abstract

A substantial portion of the formal work in artificial intelligence over the past decade has involved the formalization of nonmonotonic reasoning. Unfortunately, although much of the original motivation for this work argued that it was necessary for intelligent systems to jump to default conclusions quickly, little of the formal work has justified these arguments: Existing formal theories of nonmonotonic reasoning are nonsemidecidable, with implementations involving repeated calls to theorem provers.

We suggest that the computational value of nonmonotonic reasoning lies in its ability to focus the inference process in a hierarchical way. We show that a formal system capable of drawing tentative conclusions *without* performing a consistency check is capable both of jumping to default conclusions and, perhaps more significantly, of achieving substantial computational savings when a nondefault conclusion is required.

## 1 INTRODUCTION

In all of the applications that have been proposed for nonmonotonic reasoning, it is possible to draw default conclusions quickly; the computational expense of the default approach comes from the need for some sort of consistency check. In most cases, it is necessary to check that the default conclusions are consistent with the initial database.

A default approach to reasoning about action, for example, requires two invocations of a theorem prover in order to determine whether or not some domain fact is preserved when an action is executed. In the first, we simply use the frame axiom to conclude that it *is* preserved; in the second, we check the consistency of this conclusion.

Here, the first "proof" consists simply of an application of the frame axiom; the work is in the consistency check. Since a monotonic approach to reasoning about action involves checking a variety of persistence conditions in order to determine that the domain fact persists through the action, it is possible for the nonmonotonic approach to be the more efficient of the two. This argument is made quantitative in [Ginsberg and Smith, 1988].

Even though the nonmonotonic approach requires an additional call to a theorem prover, it is possible for it to be more efficient than the conventional one because each of the two proof attempts may be much more efficient than its monotonic counterpart. In this paper, we suggest that this is the true role of nonmonotonic reasoning in artificial intelligence: the replacement of one complex proof with two or more simpler ones.

The computational benefits of this sort of approach are well known. Consider the basic idea underlying island-driven search [Erman and Lesser, 1975] or hierarchical planning [Sacerdoti, 1977]. By identifying an island through which a plan or search must pass, it is possible to split one search problem into two smaller ones; since the difficulty of the problem grows exponentially with its size, this leads to computational savings.

In general, it is possible that we are not *certain* that the solution to a given search problem passes through some specific island, but only *expect* it. In this case, we search first for a solution passing through the proposed island, and subsequently for one that avoids it. Provided that the cost of identifying the hypothetical island is outweighed by the expected gain in solving the smaller search problem, the identification of the island is a justifiable expense.

As we will see, the identification of the island corresponds to the discovery of a default solution to the problem, and the subsequent consistency check has an analog in the search for a solution that passes through the island in question. This allows us to rephrase the observation in the previous paragraph as follows: Provided that the cost of the first default proof is outweighed by the expected gain in checking the consistency of this default proof as opposed to solving the original problem, the use of default information is a justifiable expense.

In the next section, we will discuss this issue in terms of an extended example involving the use of an ATMS to solve a synthesis problem. A discussion of the computational advantages offered by this idea is contained in Section 3. Planning examples are discussed in Section 4, and an examination of the nature of the "consistency check" required by existing nonmonotonic formalisms is the topic of Section 5. Connections with existing work on search control and hierarchical problem solving are examined in Section 6.

## 2 SYNTHESIS PROBLEMS

The focus of this paper will be on synthesis problems, such as planning or circuit design. As discussed in [Green, 1969], one way in which to implement a planning system is to use resolution to prove that there exists a situation in which the goal has been achieved and to then recover the actions needed to achieve the goal by examining the situation so constructed.

[Finger, 1987] hints at another possible approach. Here, the idea is that we have some (probably inconsistent) set $A$ of assumptions, and are looking for a consistent subset $S$ of $A$ that allows us to conclude that the goal has been achieved. Our assumption set $A$ might consist of a variety of assumptions about which actions were taken at what times; if $A$ includes both the assumption that we crossed the street at time $t$ and the assumption that we remained stationary at that time, $A$ itself will be inconsistent and we will need to commit to at most one of these actions when actually constructing our plan.

Of course, finding a consistent subset of $A$ that entails some specific goal $g$ is what ATMS's are all about [Reiter and de Kleer, 1987]. (The restriction that $S$ above be consistent is the same as de Kleer's condition that $S$ not contain a nogood [de Kleer, 1986].) We will refer to this type of problem in general as follows:

**Definition 2.1** *Suppose that we have fixed a base theory $T$, a set of assumptions $A$ and a goal $g$. We will call the problem of finding a subset $S$ of $A$ that is consistent with $T$ such that $T \cup S \models g$ the $(T, A, g)$* synthesis problem.

The thrust of this paper is that we are able to use *default* assumptions to focus this process. Thus in planning a trip from Stanford to MIT, I may generate the incomplete plan of getting to San Francisco airport (SFO) and thence to MIT via Logan; I am *assuming* that the intermediate steps of getting from Stanford to SFO, from SFO to Logan, and from Logan to MIT are all things that I can accomplish if I consider the problem in more detail.

Let us suppose for the moment that we are actually in a position to prove rigorously that we can get from Stanford to MIT via the airports suggested. (In practice, of course, all sorts of things can go wrong, and we must leave many small details of the trip unplanned until we actually encounter them.) If this is the case, then our decision to go via SFO and Logan really has done little more than focus our search effort – the detailed plan could have been generated initially instead.

What we have done is to augment our assumption set $A$ with a set of default assumptions $D$ (in this example, the assumptions that we can get from Stanford to SFO and so on). Then, having solved the planning problem given the default assumptions, we attempt to eliminate these assumptions by showing that they are in fact provable from our original set $A$ after all. Here, then, is the procedure for solving the $(T, A, g)$ synthesis problem using the default set $D$:

**Procedure 2.2** *To solve the $(T, A, g)$ synthesis problem:*

1. *Solve the $(T, A \cup D, g)$ synthesis problem to obtain a solution $\{a_1, \ldots, a_m, d_1, \ldots, d_n\}$.*
2. *Set $A_0 = \{a_i\}$ and $T_0 = T \cup A_0$ and solve the $(T_0, A, d_1)$ problem to obtain $A_1$.*
3. *Now set $T_1 = T_0 \cup A_1$ and solve $(T_1, A, d_2)$ to obtain $A_2$. Repeat to eventually solve $(T_{n-1}, A, d_n)$ to obtain $A_n$.*
4. *Return $\cup A_i$ as a solution to the original problem.*

This procedure works by first solving the original problem using the defaults, and then replacing the defaults one at a time by sets of nondefault assumptions that entail them. As we have described it, we are assuming that it is always possible to replace the defaults in this fashion, so that they are all consequences of information elsewhere in the database; in practice, this may not be the case. In the next section, we will modify the above procedure to cater to this possible difficulty.

Before proceeding, however, I should point out that the idea of using defaults to represent assumptions involved in solving synthesis problems is not a new one. The connection between ATMS's and nonmonotonic reasoning is well-understood and has been described by a variety of authors [Ginsberg, 1989, Reiter and de Kleer, 1987, and others]; the THEORIST system is built around the idea that defaults represent hypotheses that can be used in problem solving [Poole *et al.*, 1985]. The contribution made by this paper is not in identifying the connection between nonmonotonic reasoning and hypothetical solutions to synthesis problems, but in proposing that these hypothetical solutions can be used to focus the search for more certain results.

## 3 COMPUTATIONAL VALUE

Whether or not using defaults to focus search is a good idea from a computational point of view depends upon a variety of assumptions about the set of defaults $D$: Can these defaults really be used effectively to shorten the length of the original proof? Are the defaults likely to point the way to a solution to the original synthesis problem, or are they likely to be blind alleys? And so on.

In order to give a quantitative estimate of the time Procedure 2.2 needs to solve a particular query, we will need to make a variety of assumptions about these issues. Specifically, we assume that:

1. The relative cardinalities of the sets $D$ and $A$ is given by $\lambda$. We take
$$\lambda = \frac{|D|}{|A|}$$
the ratio of the number of defaults to the number of original assumptions. We will go further and assume that since the number of assumptions in the default synthesis problem $(T, A \cup D, g)$ is $1+\lambda$ times the number of assumptions in the original synthesis problem $(T, A, g)$, the branching factor in the new problem is $(1 + \lambda)b$, where $b$ is the branching factor in the original problem.
2. If $S$ is a solution to some synthesis problem with branching factor $b$ and there are $m$ assumptions in $S$, then the time needed to solve the synthesis problem is of order $b^m = b^{|S|}$. This will be valid if $b$ is the effective branching factor between additions to the set of assumptions being constructed as a solution to the problem.
3. Suppose that $S$ is the smallest solution to a synthesis problem $(T, A, g)$. Then there is some $n \geq 1$ such that the expected size of the smallest solution to the default synthesis problem $(T, A \cup D, g)$ is $|S|/n$.

   Roughly speaking, $n$ here is the number of original assumptions that we expect to be covered by a single default rule; if $n = 1$, we do not expect the defaults to be useful in shortening the length of the solution.
4. There is some fixed $l$ and probability $p$ such that for an arbitrary default $d \in D$ and an arbitrary subset $A' \subseteq A$ consistent with $T$, $p$ is the probability that the synthesis problem $(T \cup A', A, d)$ admits a solution of length $l$ or less.

   We clearly expect $l \geq n$, since each default covers on average $n$ base-level assumptions; if $p$ is relatively large and $l$ relatively small subject to this condition, each default is likely to expand into a fairly small number of nondefault assumptions. In general the probability that the $(T \cup A', A, d)$ synthesis problem admits a solution may vary depending on the sets $A$ and $A'$ and the particular default $d$; it suffices for the probability that this synthesis problem admits a solution to be no *less* than $p$.

The procedure that we will consider is a slight variant of Procedure 2.2. Here it is:

**Procedure 3.1** *To solve the $(T, A, g)$ synthesis problem:*

1. *Solve the $(T, A \cup D, g)$ synthesis problem to obtain a solution $\{a_1, \ldots, a_m, d_1, \ldots, d_n\}$.*
2. *Set $A_0 = \{a_i\}$ and $T_0 = T \cup A_0$ and attempt to solve the $(T_0, A, d_1)$ problem to obtain $A_1$, requiring that $|A_1| \leq l$.*
3. *If no such $A_1$ exists, abandon the default solution and solve the original synthesis problem $(T, A, g)$ directly.*
4. *Otherwise, set $T_1 = T_0 \cup A_1$ and attempt to solve $(T_1, A, d_2)$ to obtain $A_2$ with $|A_2| \leq l$. Repeat to either eventually solve $(T_{n-1}, A, d_n)$ to obtain $A_n$; if $A_i$ doesn't exist for some $i$, solve $(T, A, g)$ directly.*
5. *Return $\cup A_i$ as a solution to the original problem.*

As remarked at the end of the previous section, the reason that we use this modified procedure is that it is possible that the default solution found in step 1 does not expand into a full solution, and we need to bound the amount of time spent trying to expand the various defaults in subsequent steps.

Note also that if step 2 or 4 fails to find a solution using the extended assumption set $A \cup D$, we simply give up, as opposed to either backtracking to another default solution or attempting to modify the existing solution in a way that addresses the difficulty. Even given this extremely conservative behavior, however, Procedure 3.1 leads to computational savings:

**Theorem 3.2** *Let $(T, A, g)$ be a synthesis problem for which the shortest solution is of length $k$. Then provided that $l \ll k$, it will be more efficient to solve this synthesis problem using Procedure 3.1 than by conventional means whenever*
$$p > \frac{1+\lambda}{b^{n-1}} \tag{1}$$

**Proof.** First note that for a problem with solution of length $k$, the length of the solution to the synthesis problem that uses the defaults is expected to be $k/n$. Since each assumption in this solution (and there are at most $k/n$ of them) has probability at least $p$ of being successfully replaced with $l$ or fewer nondefault assumptions, it follows that the probability that Procedure 3.1 solves the synthesis problem using the defaults is at least $p^{k/n}$. Thus the total time needed to solve the problem is bounded by

$$[(1+\lambda)b]^{k/n} + \frac{k}{n}b^l + \left(1 - p^{k/n}\right)b^k \qquad (2)$$

The first two terms here represent costs that are always incurred – that of solving the default synthesis problem with branching factor $(1+\lambda)b$ and depth $k/n$ and of attempting to replace the at most $k/n$ defaults (each leading to a problem of depth $l$ and branching factor $b$). The final term reflects the cost of solving the problem without using the defaults; as argued in the previous paragraph, the probability that this cost is incurred is at most $1 - p^{k/n}$.

In order for our ideas to lead to computational savings, the expression in (2) needs to be less than $b^k$, the expected cost of solving the problem using conventional methods.

Note first that in light of the condition that $l \ll k$, the second term in (2) is dominated by the first and third; what we are arguing here is basically that the problem of replacing the default assumptions is far easier than that of solving the original synthesis problem itself. Thus the condition that we need to satisfy becomes:

$$[(1+\lambda)b]^{k/n} + \left(1 - p^{k/n}\right)b^k < b^k$$

or

$$[(1+\lambda)b]^{k/n} < p^{k/n}b^k \qquad (3)$$

Raising both sides to the power $n/k$ gives us

$$(1+\lambda)b < pb^n$$

which is clearly equivalent to the conclusion of the theorem. □

Theorem 3.2 is of substantial practical value. Consider, for example, the conservative assumptions that $n = 2$ (so that every default corresponds to only 2 non-default assumptions) and $\lambda = 1$ (so that including the default assumptions doubles the size of the assumption set). Now (1) becomes

$$p > \frac{2}{b}$$

Since most synthesis problems have fairly large branching factors, this is a very weak restriction.

Theorem 3.2 can also be rewritten in the following form:

**Corollary 3.3** *Procedure 3.1 can be expected to lead to computational savings whenever*

$$\frac{t(default)}{t(original)} < prob$$

*where $t(default)$ is the expected time needed to solve the default synthesis problem, $t(original)$ is the time needed to solve the original one, and $prob$ is the probability that the default solution can be expanded to a complete one.*

**Proof.** Given the arguments in the proof of Theorem 3.2, this is a straightforward rewriting of (3). □

Roughly speaking, it is worth risking an "investment" of 10% of the time needed to solve a particular problem if we have more than a 10% chance of solving the problem in this fashion. It is interesting to note that the cost of replacing the defaults in the tentative solution is sufficiently small that it does not enter into the above result.

# 4 PLANNING EXAMPLES

The idea that default rules can be used to generate islands in a synthesis problem allows us to reinterpret the default information typically used by systems that reason about action. As an example, consider the frame axiom. This tells us that facts true in one situation will remain true in subsequent situations:

$$\texttt{holds}(p,s) \wedge \neg\texttt{ab}(a,p,s) \supset \texttt{holds}(p, \texttt{result}(a,s)). \qquad (4)$$

Applied to planning, (4) suggests that when planning for a conjunctive goal, we simply achieve the subgoals individually. The frame axiom gives us reason to assume that previously achieved subgoals will not be clobbered by subsequent actions.

Of course, this is not valid in general. But much of the computational expense involved in planning is a consequence of the need to regress all of the goals through the various actions being considered. In situations where this regression is not needed, an optimistic approach will succeed much more quickly than the conventional one; gambling that this will indeed be the case is likely to be worthwhile because the payoff is quite large if we are correct. This observation is also made in [Elkan, 1990].

As another example, consider what might be called the "qualification assumption." By this we mean the assumption commonly made when reasoning about action that actions are unqualified in the absence of information to the contrary. Thus we assume that an attempt to start a car will succeed even if there is no information available explicitly stating that there is no potato in the car's tailpipe. Perhaps more realistically, attempts to drive the car are assumed to succeed even in the absence of specific information indicating that none of the tires is flat.

Applying this idea to planning leads us to assume that our intended actions are unqualified. This will typically be the case, and we will be saved the effort of worrying about potential qualifications as we construct the plan.

Two other advantages also come to mind here, although we will need to extend Procedure 3.1 in order to exploit them. The first involves cases where although the qualification assumptions hold (i.e., there is no potato in the tailpipe), we do not *know* that they hold. In this case, there may well be no solution to the original problem, since the necessary knowledge (about the state of the tailpipe) is simply missing from our database.

This is rather different from the previous ideas that we have considered. From a formal point of view, we are presented with incomplete domain information in the sense that our base set $T$ is only a subset of the "complete" base set $T'$. Although we really want a solution to the $(T', A, g)$ synthesis problem, we are forced to attempt to solve the $(T, A, g)$ synthesis problem. Using the defaults, we solve the $(T, A \cup D, g)$ synthesis problem instead, and observe:

1. If the defaults used in the solution are all consequences of $T'$ (as is the case with qualification assumptions), then the solution obtained will in fact be a solution to the $(T', A, g)$ synthesis problem.
2. Even if not all of the defaults used are consequences of $T'$, the solution can still focus our subsequent efforts in a fashion similar to that described in Section 3.

Finally, it may be the case that the qualification assumption actually fails – perhaps there is a potato in the tailpipe after all. In this case, the default solution to the synthesis problem can be expected to be similar to the solution to the problem itself (which presumably involves clearing the tailpipe and then proceeding as planned). Making practical use of this idea will involve considerable work in extending Procedure 3.1, of course.

# 5 THE CONSISTENCY CHECK

In the synthesis problems we have examined, we proceeded by first finding a default solution to the query, and then filling in the details of this skeletal solution. Thus instead of simply using a default rule to conclude that it is possible to fly from San Francisco to Boston, we can consult with a travel agent and confirm that it is indeed possible, and that American Airlines flies the route in question.

Default inference does not generally proceed in this fashion. Instead, approaches such as that in [Reiter, 1980] function first by finding a default solution to a problem, and then attempting to prove that the default solution *cannot* be extended to a certain one. Instead of finding a flight from San Francisco to Boston, we would attempt to show that there cannot be one. (Perhaps only Boeing 747's can fly transcontinentally, and the runways at Logan are too short for them to land.) When we fail to conclude that our default solution is *in*valid, we assume that it is in fact acceptable.

This is in contrast with the approach we have described, where we first draw a default conclusion and then attempt to prove it to be monotonically valid. But provided that our initial database is consistent, both approaches lead to the same result. Given that we can prove that our default conclusion is in fact valid, we do not need to attempt to prove it false. In our example, the existence of the American flight from San Francisco to Boston guarantees that we will fail in our attempt to prove that no such flight exists.

We have already remarked that in other instances it may be that there is no monotonic proof that we will be able to complete some portion of our skeletal plan. Consider the initial action of driving to the San Francisco airport: The highway may be closed, we may run out of gas, our car may be in the shop, and so on. Here, we may indeed take the alternative tack of considering possible proofs that we *can't* drive to the airport.

If the consistency check succeeds, so that we are unable to prove that there is *no* solution to our original problem that passes through the default island, we tentatively conclude that we will be able to complete the plan at some later date. Thus the final leg of our journey is left almost completely unspecified – I may have some vague notion that I'll rent a car when I get into Boston, but no idea about which car rental agency I'll use, how I'll find them, and so on. This is similar to the discussion of the qualification problem in the last section, where we conceded that it might be possible to find a solution to a synthesis problem given an incomplete description of the domain in question.

The consistency check may fail, however. Perhaps my car is in the shop and I can't drive to San Francisco airport or, as in the previous section, perhaps there is a potato in my tailpipe after all. Procedure 3.1 would now have us simply abandon our default plan and construct a monotonic plan after all. More effective would obviously be to "debug" the default solution to the original synthesis problem. Although there has been no formal work on this sort of a process, the approach we have outlined may be a useful first step in this direction. Again, we saw an example of this in the previous section.

Before turning to other issues, we should remark that there is another way in which we can understand our ideas generally. The reason that most approaches to nonmonotonic reasoning require a consistency check is that a declarative system is useless without it – there is simply no point in asserting a conclusion that is provably false.

In control problems, this is not the case. If it is possible to draw quick conclusions about the portion of a search space that should be expanded next, those conclusions have value even if subsequent consideration would show them to be false. The reason is that the potentially large time needed for this "subsequent consideration" may not be justified in terms of the time saved in the search itself. "Fast but not accurate" is unacceptable behavior for most declarative systems, but is *necessary* behavior in control work; because we are using the results of our default analysis primarily to control subsequent search, we are able to take advantage of default conclusions without explicitly checking their consistency.

# 6 RELATIONSHIP TO EXISTING WORK

## 6.1 HEURISTIC SEARCH AND METALEVEL ACTIVITY

One way to think of the ideas we have presented is as describing a connection between default reasoning and the control of search. The basic suggestion we have made is that an important use of nonmonotonic reasoning in AI is as a means of focussing search when considering a difficult problem. Previous investigations of the problem of search control have relied either on the use of heuristic rules or domain-independent techniques developed by considering statistical information about the contents of the database [Smith and Genesereth, 1985].

By using default information to guide search, we are applying a domain-independent heuristic to domain-dependent default information. The heuristic is to apply the default rules to narrow the scope of the problem; the domain-dependent information is in the form of default rules such as the frame axiom. The advantage of using the information in this fashion is that it simplifies the knowledge engineering problem to some extent: All of the domain-dependent information being used is in the form of declarative information about the domain being considered. In many instances, this declarative information will already be present when the synthesis problem is confronted – we can, for example, expect our planning system to already be equipped with some representation of the frame axiom. The idea that one should control search by applying domain-independent control rules to domain-dependent base-level information and not by a direct appeal to domain-dependent control rules also appears in [Ginsberg and Geddis, 1991].

There is a close connection between our ideas and those appearing in [Smith and Genesereth, 1985], which suggests examining a knowledge base to determine an ordering for the clauses in a conjunctive query. There is a clear connection between the "average number of solutions to a conjunct" [Smith and Genesereth, 1985, p. 180] and the validity of a default rule such as the frame axiom. It should be noted, however, that the goals of [Smith and Genesereth, 1985] are different from ours: The earlier work orders the conjunctive goals in the problem as stated, while we are introducing new ones (the islands) based on default information.

The approach we have proposed also makes it possible to draw default conclusions about the domain itself. Even if no travel agent is available, it is important to be able to *surmise* that it will be possible to fly from San Francisco to Boston. This point is even clearer for portions of a skeletal plan (such as driving to the airport) that cannot be expanded until the plan is executed. The argument that default information is needed by autonomous agents functioning in uncertain or unpredictable domains is hardly a new one; we have merely formalized the old idea that this same information can be used in controlling search as well.

## 6.2 HIERARCHICAL PLANNING AND DESIGN

The approach we have described is little more than a formalization of the conventional notion of hierarchical problem solving. Nevertheless, we feel that the contribution made by this paper is important for two separate reasons.

First, it provides a general description of the idea of hierarchical problem solving. This description is independent of whether the application is to planning, design or some other synthesis problem.

Far more importantly, the information used to describe the planning (or other) hierarchy is declarative. Multiple levels of hierarchy can clearly be handled using prioritized defaults; the key point here is that the split is not an *ad hoc* one introduced by the system designer, but instead a natural one generated from existing declarative information.

It might seem that this is a small point, and that all of the usual problems encountered in hierarchical problem solving (e.g., the fact that potential islands need to be identified by the system designer) remain in our approach, but this is not the case. As an example, consider just this difficulty – where do the defaults come from? We can offer the following suggestions:

1. They can be entered by the system designer as usual, with his intention being to control search. Our language is somewhat different, but no expressive power has been lost.
2. They can be part of the declarative knowledge base for another reason. An example is the frame axiom, which will surely be part of any system that plans and reasons about action. Although not made a part of the system for control reasons, it appears that the frame axiom can be used effectively in this fashion.
3. They can be learned from previous problem solving efforts. Caching partial plans can now be viewed declaratively as a matter of caching the

defaults indicating that these plans will be useful.

Another important feature of the declarative approach we have proposed is that it may allow us to understand the process of plan debugging. As an example, consider a hierarchical planner that has failed to expand a particular high-level plan into lower-level actions. If the planning hierarchy is described procedurally, there is no real way for the planner to overcome its difficulties. But if the planning hierarchy is a consequence of declarative default information, it will be possible to trace the failure to the failure of a specific default rule – and that may well be a necessary first step toward salvaging the high-level plan in some way.

**Acknowledgements**

This work has been supported by NSF under grant number DCR-8620059 and by the Rockwell Palo Alto laboratory. I would like to thank Karen Myers, Narinder Singh and David Smith for many useful discussions and suggestions.

## References

[de Kleer, 1986] Johan de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[Elkan, 1990] Charles Elkan. Incremental, approximate planning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 145–150, 1990.

[Erman and Lesser, 1975] Lee Erman and Victor Lesser. A multi-level organization for problem-solving using many diverse, cooperating sources of knowledge. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 483–490, Tbilisi,USSR, August 1975.

[Finger, 1987] Jeffrey J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, CA, 1987.

[Ginsberg and Geddis, 1991] Matthew L. Ginsberg and Donald F. Geddis. Is there any need for domain-dependent control information? In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1991. Submitted.

[Ginsberg and Smith, 1988] Matthew L. Ginsberg and David E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[Ginsberg, 1989] Matthew L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[Green, 1969] C. C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier, New York, 1969.

[Poole *et al.*, 1985] D. Poole, R. Aleliunas, and R. Goebel. THEORIST: A logical reasoning system for defaults and diagnosis. Technical report, University of Waterloo, 1985.

[Reiter and de Kleer, 1987] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 183–188, 1987.

[Reiter, 1980] Ray Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Sacerdoti, 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, New York, 1977.

[Smith and Genesereth, 1985] David E. Smith and Michael R. Genesereth. Ordering conjunctive queries. *Artificial Intelligence*, 26(2):171–215, 1985.