A Distributed Plan Verifier*

Yan Xiao† Russell Greiner‡

Abstract

This paper describes the architecture of an efficient plan verifier that can first detect faults in a planner's plans and use these observed errors to identify possible problems in the planner's knowledge base and suggest appropriate corrections.

1 Introduction

We routinely hire consultants to use their specialized knowledge of some domain to verify our plans. This serves two purposes: (*i*) to confirm that our plans achieve our intended goal, and that these plans cause no bad side-effects and (*ii*) to improve our understanding of the world (by acquiring missing information and correcting faulty assumptions), to prevent us from producing such faulty plans again. In a multi-agent environment, other agent(s) can similarly help a planning agent (P) by examining, and correcting, its proposed plan (Γ). In general, as these other agents may have superior knowledge about a certain domain, or a special facility for recognizing certain types of errors in plans, they can often correct faults in Γ. They can often also find and correct the problems in the planner that led to these faults. This can be an efficient way for agents to cooperate, as it allows each agent to specialize in certain domains and tasks, rather than forcing one agent to know everything. This concept of distributed knowledge source is especially useful in a dynamic situation, where the different agents have access to different information (*e.g.*, if they are located in different places, etc.). Rather than requesting the other agents to send all of their information to P, which leads to a major communication problem, these agents could instead simply check P's plans for faults, and then communicate only the information necessary to correct the faults found; *i.e.*, telling P of missing or faulty statements in its knowledge base. Therefore, this plan verification process can lead to an efficient way for agents to communicate relevant information.

This paper proposes an overall architecture for such a system, one that allows various cooperating agents (called plan verifiers, or PVs) to efficiently help a planning agent P to produce an effective plan, and also to inform P of possible errors or missing information in its knowledge base. Section 2 first summarizes how this work is related to several other research projects. Section 3 then presents a simple blocks world example, to illustrate the relevant situations. The rest of the paper presents a more formal description of this overall system. Section 4 lists the assumptions underlying this work. Section 5 describes

*This work is supported by an OGS scholarship to the first author, and an NSERC operating grant to the second.
†Department of Industrial Engineering, University of Toronto, Toronto, Ontario M5S 1A4
‡Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4

the structure of the overall plan verification system, including descriptions and algorithms for two classes of *PV*s. Section 6 discusses the benefits of this "distributed plan verifier" approach.

2 Related Work

Our approach shares many similarities with several other systems. First, many systems exploit the synergy of multiple agents working together to achieve a single goal. These agents are called "knowledge sources" (KSs) in BlackBoard systems [HR85]. Our use of these agents is slightly different, as (1) each KS is assumed to be an expert in its own domain; by contrast, we assume that *P* and *PV* share a great deal of information, but *PV* knows more of that domain, more accurately. This leads to a second difference: (2) while a blackboard system's only objective is to work on the task at hand, one of *PV*'s goals is to "enlighten" *P*.

This ties our Plan Verification approach in with several learning systems. One example of this genre is the LEAP system [MMS85]. Here again one part of a system verifies the plan of other, and uses this information to enlighten the "assistant". It differs in that LEAP tries to educate itself, while *PV* tries to educate *P*. Another difference is that LEAP's underlying task is to convert a sub-optimal plan into better one; by contrast, our *P*-and-*PV*-system is trying to obtain any acceptable plan.

Our work is also related to cooperative planning. As one example, Malyankar [Mal90] considers how multiple agents, having different views of the common world, can cooperatively plan a route. In that work, each agent predicts potential conflicts between its planned route with the others', and uses these predictions to determine what the other agents need to know. This approach requires that each agent be able to produce a plan and have equal "width" of knowledge base, whereas in our approach, the *PV*s need not have any explicit planning ability, and their knowledge base needs only be superior to *P*'s in certain specific sub-domains.

We use plan verification as a way to critique plans. In many systems, the critic must itself be able to plan [LS83, Mil83]. This is not required in our work; instead, our critics simply check the plan by analyzing parts of the planner's proof and knowledge base that it used to generate the plan. We will see that this allows the critics to have narrower range of knowledge, yet still be useful, provided it has superior knowledge in its own sub-domain.

Once incorporated with plan recognition system [All83, KA86, SSG78], the *PV* system can also be developed into a practical and comprehensive warning system (which was the original objective of this research), capable of detecting possible problems of some ascribed plan and of giving both warnings and helpful information.

3 Simple (Blocks World) Example

We start with the classic AI approach to planning [GN87]: Given a goal ρ, a planning agent *P* generates a plan Γ (*i.e.*, a sequence of actions) that it thinks will achieve the goal ρ. (That is, we insist that *P* be "rational" [New81, p8].) As an example, consider the simple block world situation shown in Figure 1, and imagine the goal of our planner, *P*, is to build a tower with block `a` on top of block `b`. — *i.e.*,

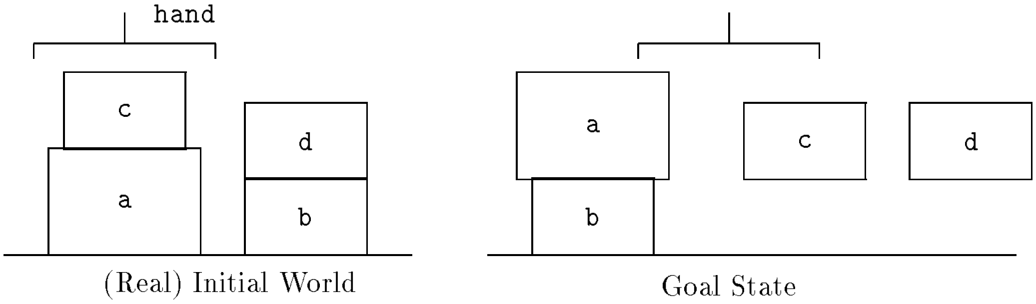

Figure 1: Blocks world: Initial and Goal configurations

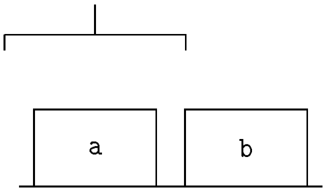

Figure 2: P's (Incorrect) Representation of Initial State

ρ = `on(a, b, ?s)`. If P's representation of the world is inaccurate or incomplete, then it may generate inappropriate plans. We consider two classes of problems in plans that have fatal errors: ones that do not achieve their goals, and that cause bad side-effects. As an example, imagine P's representation of the world is as shown in Figure 2: notice it believes that `a` has a clear top. It may then generate the plan:

$$\Gamma_1 \;=\; \texttt{on(a, b, PutOn(b, PickUp(a, S0)))} \tag{1}$$

that is, pick up `a`, then put it on `b`, as it thinks that this plan will achieve its goal. (Of course, `S0` represents the initial state. See Figure 3.)

This plan, of course, will not work. In particular, the fact

$$\texttt{On(c,a,S0)} \tag{2}$$

together with the rule

$$\texttt{On(?y,?x,?s)} \Rightarrow \neg\texttt{ClearTop(?x,?s)} \tag{3}$$

mean that

$$\neg\texttt{ClearTop(a,?s)} \tag{4}$$

This means that the hand will be unable to simply pick up `a`, and so P's plan (of picking up `a`, etc.) will not succeed. As P is rational, it would not have produced Γ_1 if it knew Fact 4.

To show how our system would critique P's plan Γ_1, assume it includes the plan verifier, PV_1^A, that is an expert on the whereabouts of the block `c`, and its effects.[1] As it knows, in particular, Facts 2 and 3, it

[1]This is, of course, a trivial and relatively uninteresting situation. For a more interesting example, imagine that P is

`ClearTop(a,S0)`   `EmptyHand(S0)`   `ClearTop(?x,?s)`$\wedge$`EmptyHand(?s)` $\Rightarrow$ `InHand(?x,PickUp(?x,?s))`

[`S1` $=$ `PickUp(a,S0)`]

`InHand(a,S1)` `ClearTop(b,S1)`   `InHand(?x,?s)`$\wedge$`ClearTop(?y,?s)` $\Rightarrow$ `On(?x,?y,PutOn(?y,?s))`

[`S2` $=$ `PutOn(b,S1)`]

`On(a, b, S2)`

Figure 3: P's proof of plan Γ_1

can conclude Fact 4.

How can PV_1^A, who knows this information, help P? As one approach, P could send all it knows (*i.e.*, P's knowledge base KB_P) to PV_1^A, and have it check and correct errors it finds. (Here PV_1^A would observe that KB_P does not include `On(c,a,S0)`, `Green(c,S0)`, `Dirty(c,S0)`, ..., and so would would relay that information to P. Notice only the first fact is relevant here.) This can be very expensive, as it can involve arbitrarily difficult derivations; and worse, much of this effort will be irrelevant, as PV_1^A may find many errors that would never be manifest in any of P's actions.

This paper follows another approach: Here, PV_1^A instead checks only the proof tree (called Γ_1 in Figure 3), and reports errors it finds here. This "as needed" approach can be relatively efficient, as it allows PV to check only the parts of P's knowledge base that are actually used for some task, rather than all of KB_P. (Here it only deals with Fact 2

We will soon discuss how PV_1^A will deal with this situation. First, observe that this plan verifier can be partial — that is, only knows about information related to `c`; if `c` had been elsewhere, PV^A would have considered `a`'s top to be clear,[2] and so would not have vetoed P's plan, even that the fact that `b` is not clear would also have doomed the plan.

We can, therefore, consider using a *set* of different plan verifiers, each a specialist in some area, and each capable of vetoing the overall plan, if necessary. We could, for example, also employ PV_2^A, an expert on whereabouts of block `d`, as well as PV_1^B and PV_2^B, experts on the weights of blocks and their effects on other blocks (resp., the `hand`). Here, PV_2^A would know `On(d,b,S0)` and therefore, that $\neg$`ClearTop(b,S0)`

---

preparing a drug therapy for a patient; and this PV_1^A knows about the effects of some medical condition. We can think of PV_1^B (discussed later) as an expert on the bad side-effects of some drug; etc.

[2]This, of course, involves a version of the Closed World Assumption.

(using Fact 3). As PV_1^B knows

$$\texttt{Weight(a, 1ton)} \tag{5}$$

and

$$\texttt{On(?x, ?y, ?s)} \;\&\; \texttt{Weight(?x, 1ton)} \;\Rightarrow\; \texttt{Break(?y, ?s)} \tag{6}$$

it can conclude `Break(b, ?s)`, which it knows is a bad thing to happen, *i.e.*,

$$\texttt{Break(?x, ?s)} \Rightarrow \texttt{Bad(?s)} \tag{7}$$

PV_2^B is similar to PV_1^B, but includes

$$\texttt{PickUp(?x, ?s)} \;\&\; \texttt{Weight(?x, 1ton)} \;\Rightarrow\; \texttt{Break(hand, ?s)}$$

rather than Fact 6. It can, therefore, conclude `Break(hand, ?s)`, which it knows to be a bad effect.

As mentioned above, the fact that P, a rational agent, produced the plan Γ_1 (shown in Equation 1), means that P thought that its plan would succeed, and would not lead to these bad side-effects. These errors in P's plan, therefore, mean that P either (1) misunderstood the preconditions of its intended actions (or, perhaps, misunderstood some facts), or thought that both `a` and `b` were clear; and furthermore, (2) did not think that either block `b` or the `hand` would break as a result of the actions proposed (perhaps because P was unaware of either `a`'s weight, or of its effect), or did not realize that breaking things was bad.

As we can see, each of these PVs correctly knows some information that P does not. In the following section, we develop a plan verification system that can first find certain classes of faults in a planner's proposed plan, and then use these observed errors to identify possibly problems in the planner's knowledge base.

4 Assumptions and Notation

There are, in general, many ways a plan can go wrong: *e.g.*, it might not lead to its assigned goal, have bad side effects, be sub-optimal, self-conflicting, etc. This paper considers PVs that attempt to detect faults of the first two types. Each member of the first, identified as PV_i^A, checks whether the proposed plan will Achieve the goal. The second, PV_j^B, detects Bad side effects in P's plan.

To prevent P from committing these same errors again, these PVs also help P find the causes of these faults. There are several possible causes: the planner may not have correct or complete knowledge of the relevant parts of the world, or its search during planning may not be adequate, etc. Our PVs deal only with the first of these, by telling P of differences between P's encoding of the world and "reality" (that is, the PV's knowledge base), based on these observed faulty plans. To simplify our discussion, we will assume that P's reasoning is sound and complete.

We assume that each PV's knowledge base is accurate, though it may not have complete knowledge about the relevant task. These knowledge bases include both information about some relevant aspects of the world, and specific information needed to debug plans. For each PV_j^B, we assume it also knows about certain types of bad side effects $B = \{\beta_1, \cdots, \beta_n\}$, and includes the facts, $\{\, \beta_i \Rightarrow \texttt{Bad}(?s) \,\}_{i=1}^{n}$. Each PV

takes as input

- P's knowledge base, KB_P (implicit in Figure 2, etc.)
- P's plan Γ (which involves its intended goal ρ); (see Equation 1) and
- the proof tree (Σ) P used to prove that the plan is sufficient.[3] (See Figure 3.)

5 Two Types of PVs

We consider two types of problematic plans:

A: Γ does not achieve ρ.
As P is rational, we know that P believes that Γ will achieve ρ; unfortunately, it will not:

$$KB_P \models \Gamma \Rightarrow \rho \qquad KB \not\models \Gamma \Rightarrow \rho$$

where KB is an omniscient representation of reality. We'll use $\vdash_P$ as P's proof process.

B: Γ has a bad side effect (*i.e.*, leads to some $\beta \in \mathtt{B}$).
Again by rationality assumption, P believes that Γ will not bring any side effect but in fact it will:

$$KB_P \not\models \Gamma \Rightarrow \mathtt{Bad} \qquad KB \models \Gamma \Rightarrow \mathtt{B}$$

As mentioned above, we will use PV_i^A (resp., PV_j^B) to represent a plan verifier that checks for problematic plans of type A (resp., type B). These two types of problems are basically "duals", in that P believes Γ will achieve ρ but does not believe Bad will happen; but in reality Γ will not achieve ρ and/or will achieve Bad. We will exploit this property to avoid redundancy and thereby simplify our discussion in the following subsections.

5.1 The PV^A System

There are two ways a PV^A can confirm that Γ will achieve ρ: The standard (if overly strong) sense requires that PV^A actually *prove* that Γ will achieve ρ:

$$KB_{PV^A} \models \Gamma \Rightarrow \rho$$

A weaker, more useful form requires only that $\Gamma \Rightarrow \rho$ be consistent, *i.e.*,

$$KB_{PV^A} \not\models \neg[\Gamma \Rightarrow \rho]$$

We assume that each PV^A have its own proof process which we can label as "$\vdash_{PV^A}$" which, in general, corresponds to one of these two processes (*i.e.*, either derivability or consistency).

[3] Given P's knowledge base and its plan Γ, PV could re-construct a derivation of the goal. This practice is, however, very expensive, and is also unnecessary, as PV is a "collaborator" of P.

Notice that, if one believes all of the statements used in a sound proof, then one must believe the conclusion. We need, therefore, only confirm all of the elements of Σ to believe that $\Gamma \Rightarrow \rho$ is accurate.

We can view each PV^A process as:

```
PV^A( Γ, ρ, Σ )
            Let   Φ⁻ = {σ ∈ Σ | KB_{PV^A} ⊬_{PV^A} σ}
                                %% Recall that Σ contains the reasons why P thinks Γ achieves ρ
                                %% Φ⁻ contains the elements of KB_P that are not
                                %%derivable from or consistent with KB_{PV^A} — i.e., may not be true
LA1:        if   Φ⁻ = {}
            then                %% Plan seems to achieve the goal.
                   return   SUCCESS
            else                %% Plan will NOT achieve the goal.
LA2:               Tell P: "Plan Γ may not achieve goal ρ"
LA3:               Tell P: "The claims in Φ⁻ are questionable"
                   return   ABORT
end PV^A.
```

Line LA1 is based on the observation that if PV^A agrees with every $\sigma \in \Sigma$, then it must agree with the conclusion these statements collectively prove — *viz.*, that $\Gamma \Rightarrow \rho$. Otherwise, PV^A does not need to agree with the conclusion; and tells P this in Line LA2. Line LA3 points to the set that contains exactly the contraversial elements Φ^-.

This PV^A algorithm, of course, depends on $\vdash_{PV^A}$. If $\vdash_{PV^A}$ is "derivable", then each underivable element in Σ will be considered as a possible fault and therefore PV^A will veto Γ. The "unconfirmed-plan-is-faulty" approach means PV^A will abort every plan unless it knows everything related to the planning task. This resembles the situation where a master checks an apprentice's plan. We might relate it to the scenario where failure of a plan is catastrophical.

The alternative is to use $\vdash_{PV^A}$ in the weaker sense: checking whether it is consistent for PV^A to believe every element in Σ.[4] The PV_1^A of Section 3 is an example of this algorithm. From its knowledge base, it can proves that ClearTop(a,S0), a node of Σ_1, is false.

5.2 PV^B System

Like PV^A, each PV^B system depends critically on its underlying proof process, $\vdash_{PV^B}$. Here, again, there is an obvious split between those systems which use derivability versus those based on a simpler consistency checking, and these two proof processes actually lead to two ways of checking Σ if: (*i*) PV^B can not prove that there is no bad side effects versus *ii*) PV^B can prove that side effects will happen. In our daily life, few cases exist where we try to prove that bad side effects will not happen. Rather, we assume that bad side effects, in general, does not happen. meaning, we will pass Γ if we can not prove Γ leads to some bad side effect. Likewise, PV^B accepts a plan if $PV^B \not\vdash_{PV^B} \Gamma$ Bad. The general algorithm is:

[4] A slight extension of the algorithm would check the consistency of $(KB_{PV^A} \cup \Sigma)$. This enlarges the extent to which PV^A can find errors in Γ: to see whether it can prove: $\forall \sigma \in \Sigma \; KB_{PV^A} + \Sigma' \vdash_{PV^A} \neg\sigma$, where $\Sigma' \subset \Sigma$ *and* $\sigma \notin \Sigma'$.

$PV^B(\, KB_P,\, \rho,\ \Sigma\ )$

LB1: Let $S_{PV^B} = \textsc{Support}(KB_{PV^B} \cup \Sigma, \texttt{Bad})$
    %% S_{PV} lists why (PV^B thinks) Γ leads to bad side effects.
LB2: Let $\Phi^+ = \{\sigma \in S_{PV^B} \mid KB_P \not\vdash_P \sigma\}$
LB3 `if` $\Phi^+ = \{\}$
    `then` *%% plan does lead to bad side effects.*
        `return SUCCESS`
    `else`
LB4: `Tell` P: "`Plan` Γ `leads to bad side effect,` β"
LB5: *%% Φ^+ lists elements that are true, but not in* KB$_\mathrm{P}$.
        `Tell` P: "`Add` Φ^+"
        `return ABORT`

`end` PV^B.

`LB1` uses the $\textsc{Support}(\cdots)$ subroutine, which takes a collection of propositions (a.k.a., a knowledge base) and a proposition, and returns the support for this proposition from this knowledge base — *i.e.*,

$$\textsc{Support}(\, KB, \sigma\,) \;=\; \tau \qquad if \qquad \tau \subset KB \quad \& \quad \tau \vdash_{KB} \sigma$$

(We also insist that this τ is minimal, in that each proposition in τ is used in the proof of σ.)[5] Line `LB2` collects those propositions in Φ^+ that P can not derive and therefore are necessary for it to know, in order to believe its plan has some bad side effect. The proof process ($\vdash_P$) in `LB2` can be largely simplified to a 'lookup' process,see Section 5.3.

For example, as PV_1^B knows Facts 6 and 5, from `On(a,b,S2)`$\in \Sigma_1$ (shown in Figure 3) it is able to derive `Break(b, S2)`.

5.3 Issues

Two types errors in KB_P: As this paper does not deal with faulty reasoning process, (*i.e.*, we assume that P is a rational agent with a sound and complete reasoning process), the faults that cause problems in Γ must lie in P's knowledge base KB_P. This work deals with two types of problems in KB_P: **misbeliefs**: KB_P includes some statement that is not true, and **missing beliefs**: KB_P does not include some statement that is true. PV's second objective is greatly simplified by the observations that P's misbeliefs leads to error type A (*i.e.*, to plans that do not really achieve their goal); and P's missing beliefs leads to error type B (*i.e.*, plans with bad side effects).

Efficiency: The verifying process shown here can be very efficient. As shown above, procedures for identifying the faulty or missing beliefs are essentially the same: each involve examining each element in a given proof tree, to determine which members are not derivable from or consistent with a given knowledge base. We use $\vdash$ for this derivability or consistency test only to be completely general. In most situations, however, this test reduces to the much simpler $\sigma \in KB$ test, as the σs within the proof tree are, in general, simple "primitive statements", that are either explicitly in the knowledge base or, if not, are not derivable

[5] Technically, there can several support sets for any proposition; *i.e.*, this $\textsc{Support}(\cdots)$ really should be a multi-function, rather than a function. This does not lead to any conceptual complications; we use this form to keep the presentation simple.

from the knowledge base. That is, most of these σs have the property that, from most knowledge bases, $KB \Rightarrow \sigma \quad iff \quad \sigma \in KB$.

Global Problems: There are, of course, certain potential problems with our "multiple cooperating agents" approach to planning. In particular, this overall system may be unable to detect "global problems" with a plan, as each verifier only sees that the plan is "locally correct". (One such plan involves painting the ladder before painting the ceiling *c.f.*, [Sac77].) This work does not deal with such interactions.

6 Conclusions

In general, a planner must (correctly) know every fact relevant to any goal. This places quite a burden on both the planner itself, and to its author. One way around this problem is to distribute the information, forming sub-modules (or "agents") that can each "specialize" in certain domains. This work describes one way for these agents to interact: by using one as planner, and the others as critics whose common task is to find (certain classes of) errors in both the plan produced, and in the knowledge base of the planning agent that produced the plan. This approach allows each individual agent to specialize in its own area of expertise, knowing that the other agents will prevent P from forming erroneous plans, and can, in fact, also correct errors in P's knowledge base.

Another potential advantage is in terms of the planning-time efficiency — as it can be more efficient for one agent to produce a possibly-buggy plan and then have another agent fix it, than for a single agent to produce a correct plan. In general, the time to produce a plan is (roughly) exponential in the number of facts the planner knows — *i.e.*, $O(2^n)$. If we distribute the information, into k agents, then on average each agent will need to deal with n/k facts. Hence the planner Pwould require the time $O(2^{\frac{n}{k}})$. Of course, this agent might be wrong, as its plan may, in fact, require the other $\frac{(k-1)}{k}n$ chunks of information, now resident in the other agents. We therefore expect each of these other agents to confirm P's plan, each of these derivation processes also require $O(2^{\frac{n}{k}})$ time. The advantage, here, is that these times can add, rather than multiply. Hence, the total time requires is only $O(k \times 2^{\frac{n}{k}})$, which is usually much less than $O(2^n)$. Furthermore, the obvious parallel implementation would require $k-1$ processors and a throughput time of only $O(2 \times 2^{\frac{n}{k}})$ — the first $O(2^{\frac{n}{k}})$ for planning, and the next, for verification. Of course, the overall system may require several iterations — after some PV_i vetos P's plan, P will produce a correct plan.

Surely this makes many assumptions about the nature of the domain. In particular, this assumes that the different parts are (at worst) semi-decomposable [Sim69]. More research is required to determine, empirically or analytically, which domain qualify.

Knowing that we have a separate plan verifier means that we can build a planner that can efficiently generate plans without having all the information, but that the plans it will produce are as good as the ones produced by a single agent, but more efficiently. The present architecture describes one efficient way of checking plans for certain types of errors and of correcting the related errors in the planner's knowledge base. Still, this approach could not be verified without an implement. Some interesting future work is deciding how to deal with the interaction among parts of a plan, and to coordinate multiple agents.

References

[All83] James Allen. Recognizing intentions from natural language utterances. In M. Brady and R.C. Berwick, editors, *Computational Models of Discourse*, page 114. MIT Press, Cambridge, Mass, 1983.

[GN87] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[HR85] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence: An International Journal*, 26(3):251–321, July 1985.

[KA86] H.A. Kautz and J.F. Allen. Generalized plan recognition. In *AAAI-86*, pages 32–37, 1986.

[LS83] C.P. Langlotz and E.H. Shortliffe. Adapting a consultation system to critique user plans. *Int. J. Man-Machine Studies*, 19:479–496, 1983.

[Mal90] Raphael Malyankar. *Coorperative Route Planning by Multiple Agents.* Master's Thesis, University of New Hampshire, 1990.

[Mil83] P.L. Miller. Attending: Critiquing a physician's management. *IEEE Trans. on Pattern Analysis and Machine Inteligence*, PAMI-5:449–461, 1983.

[MMS85] Thomas M. Mitchell, Sridhar Mahadevan, and Louis I. Steinberg. LEAP: A learning apprentice for VLSI design. In *IJCAI-85*, pages 573–80, Los Angeles, August 1985.

[New81] Alan Newell. The knowledge level. *AI Magazine*, 2(2):1–20, Summer 1981.

[Sac77] E.D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier North-Holland, New York, 1977.

[Sim69] Herbert Simon. *Science Of the Artificial.* MIT Press, Cambridge, 1969.

[SSG78] C.F. Schmidt, N.S. Sridharan, and J.L. Goodson. The plan recoginition problem. *Artificial Intelligence: An International Journal*, 11:45–83, 1978.