Dynamic Abstraction Planning

Robert P. Goldman, David J. Musliner, Kurt D. Krebsbach, Mark S. Boddy
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{goldman,musliner,krebsbac,boddy}@htc.honeywell.com

Abstract

This paper describes Dynamic Abstraction Planning (DAP), an abstraction planning technique that improves the efficiency of state-enumeration planners for real-time embedded systems such as CIRCA. Abstraction is used to remove detail from the state representation, reducing both the size of the state space that must be explored to produce a plan and the size of the resulting plan. The intuition behind this approach is simple: in some situations, certain world features are important, while in other situations those same features are not important.

By automatically selecting the appropriate level of abstraction at each step during the planning process, DAP can significantly reduce the size of the search space. Furthermore, the planning process can supply initial plans that preserve safety but might, on further refinement, do a better job of goal achievement. DAP can also terminate with an executable abstract plan, which may be much smaller than the corresponding plan expanded to precisely-defined states. Preliminary results show dramatic improvements in planning speed and scalability.

Introduction

We are interested in constructing plans for controlling embedded real-time systems. By "embedded," we mean that these systems are interacting with a dynamic environment including unexpected, exogenous events. By "real-time," we mean that catastrophic failure is possible if a timely control action is not taken in certain situations. For these systems, control plans *must* provide guarantees that failures will not occur.

Classical planning research has typically involved the construction of a single path through a sequence of states from an initial state to a goal state. In contrast, constructing plans that take into account exogenous events and timing failures requires exploring all possible execution sequences and the resulting states. In the worst case, this involves examining a state space whose size is exponential in the number of propositions present in the state description.

This paper describes an automatic, dynamic abstraction planning technique that directly addresses this state-space explosion problem. Abstraction is used to omit detail from the state representation, reducing both the size of the state space that must be explored to produce a plan, and the size of the resulting plan itself. The abstraction method we describe has three useful features:

- First, the abstraction method does not compromise safety-preserving guarantees: the world model used for planning is reduced, but not in ways that affect the system's ability to make rigorous statements about the safety assurances of plans it is building.
- Second, the method is fully automatic, and dynamically determines the appropriate level of abstraction during the planning process itself.
- Third, the method uses different levels of abstraction in different parts of the search space, individually adjusting how much detail is omitted at each step.

The intuition behind DAP is fairly simple: in some situations, certain world features are important, while in other situations those same features are not important. An optimal state space representation would capture only the important features for any particular state. In essence, DAP allows a planner to search for useful state space abstractions at the same time it is searching for a plan.

This paper describes a prototype DAP planner that derives safety-preserving, goal-achieving reactive plans in exponentially large state spaces. We present both the theoretical and algorithmic descriptions of the planning technique, as well as preliminary results showing dramatic improvements in planning efficiency and scalability.

Background: Overview of CIRCA

We developed the prototype DAP planner to address the scalability issues faced by the state-space planner in CIRCA, the Cooperative Intelligent Real-time Control Architecture (Musliner, Durfee, & Shin 1993;

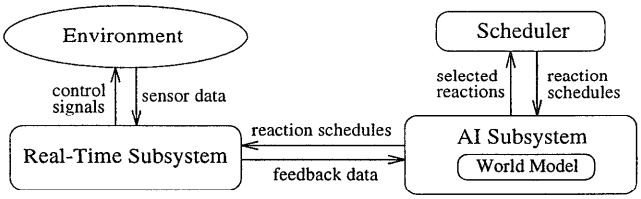

Figure 1: The Cooperative Intelligent Real-Time Control Architecture.

1995). CIRCA is designed to support both hard real-time response guarantees and unrestricted AI methods that can guide those real-time responses. Figure 1 illustrates the architecture, in which an AI subsystem (AIS) reasons about high-level problems that require its powerful but potentially unbounded planning methods, while a separate real-time subsystem (RTS) reactively executes the AIS-generated plans and enforces guaranteed response times. The AIS and Scheduler modules cooperate to develop executable reaction plans that will assure system safety and attempt to achieve system goals when interpreted by the RTS.

Example Domain

CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics and simulated autonomous aircraft. In this paper we draw examples from a domain in which CIRCA controls a simulated Puma robot arm that must pack parts arriving on a conveyor belt into a nearby box. The parts can have several shapes (e.g., square, rectangle, triangle), each of which requires a different packing strategy. The control system may not initially know how to pack all of the possible types of parts— it may have to perform some computation to derive an appropriate box-packing strategy. The robot arm is also responsible for reacting to an emergency alert light. If the light goes on, the system must push the button next to the light before a fixed deadline.

In this domain, CIRCA's planning and execution subsystems operate in parallel. The AIS reasons about an internal model of the world and dynamically programs the RTS with a planned set of reactions. While the RTS is executing those reactions, ensuring that the system avoids failure, the AIS is able to continue executing heuristic planning methods to find the next appropriate set of reactions. For example, the AIS may derive a new box-packing algorithm that can handle a new type of arriving part. The derivation of this new algorithm does not need to meet a hard deadline, because the reactions concurrently executing on the RTS will continue handling all arriving parts, just stacking unfamiliar ones on a nearby table temporarily. When the new box-packing algorithm has been developed and integrated with additional reactions that prevent failure, the new schedule of reactions can be downloaded to the RTS.

CIRCA's planning system builds reaction plans based on a world model and a set of formally-defined safety conditions that must be satisfied by feasible plans (Musliner, Durfee, & Shin 1995). Because this world model is the focus of the abstraction techniques discussed in this paper, a brief review of the model formulation is in order.

CIRCA's World Model

The world model is a directed graph representing the *worst-case* behavior of the environment and the actions which the RTS can take to avoid failure. The graph model has five elements $(S, \mathcal{F}, T_E, T_A, T_T)$:

1. A finite set of "states" $S = \{S_1, S_2, ..., S_m\}$, where each state S_i represents a description of relevant features of the world. The features of a state are represented by the set $F = \{F_1, F_2, ..., F_r\}$. Each feature $F_i \in F$ has a finite set of possible values $val(F_i)$.
2. A distinguished failure state $\mathcal{F}$, which subsumes all states that violate domain-specific safety constraints. The system strives to avoid the failure state.
3. A finite set of "event transitions" T_E that represent world occurrences as instantaneous state changes.
4. A finite set of "action transitions" T_A that represent actions performed by the RTS.
5. A finite set of "temporal transitions" T_T that represent the progression of time and continuous processes. We only represent the temporal transitions that lead to significant process state changes.

To describe a domain to CIRCA, the user inputs a set of transition descriptions that implicitly define the set of reachable states. For example, Figure 2 illustrates several transitions used in the Puma domain. The AIS plans by generating a nondeterministic finite automaton (NFA) from these transition descriptions. Beginning from a set of designated start states, the AIS enumerates the reachable states and assigns to each state either an action transition or `no-op`. Actions are selected to *preempt* transitions that lead to failure states and to move towards states that satisfy as many goal propositions as possible. The assignments determine the topology of the NFA (and so the set of reachable states): preemption of temporal transitions removes edges and assignment of actions adds them. System safety is guaranteed by planning action transitions that preempt *all* transitions to failure, making the failure state unreachable (Musliner, Durfee, & Shin 1995). It

```
EVENT emergency-alert                          ;; Emergency light goes on
      PRECONDS: ((emergency nil))
      POSTCONDS: ((emergency T))

TEMPORAL emergency-failure                     ;; Fail if don't attend to
      PRECONDS: ((emergency T))                ;; light by deadline
      POSTCONDS: ((failure T))
      MIN-DELAY: 30 [seconds]

ACTION push-emergency-button
      PRECONDS: ((part-in-gripper nil))
      POSTCONDS: ((emergency nil) (robot-position over-button))
      WORST-CASE-EXEC-TIME: 2.0 [seconds]
```

Figure 2: Example transition descriptions given to CIRCA's planner.

is this ability to build plans that guarantee the correctness and timeliness of safety-preserving reactions that makes CIRCA suited to mission-critical applications in hard real-time domains. However, this planning algorithm can be very time-consuming because it enumerates all reachable world states. In the following section, we show how dynamic abstraction can make the planning process more efficient and responsive.

Dynamic Abstraction Planning

In a state-space model like CIRCA's, one of the most straightforward ways of using abstraction is to simply remove a feature from the description of the world. This corresponds closely to the methods used in early work on abstraction planning systems to generate abstract operators by omitting less-critical elements of operator precondition lists (cf. ABSTRIPS (Sacerdoti 1974)). ABSTRIPS planned at an abstract level that then restricted the extent of the detailed planning required to build a final plan. The DAP technique is significantly different in that:

- The selection of which features to "abstract away" is performed automatically during planning.
- The abstractions are *local*, in the sense that different parts of the state space may be abstracted to different degrees.
- The abstractions preserve guarantees of system safety.
- The planning system need not plan to the level of fully-elaborated states to construct a feasible, executable plan.

The DAP concept itself is simple: rather than always using all of the available features to describe world states, we let the planner dynamically decide, for each new world state, the level of description that is necessary and desirable. By ignoring certain features, the planner can reason about *abstract states* that correspond to *sets* of "base-level" states, and thus can avoid enumerating the individual base-level states.

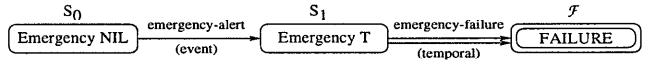

Figure 3: A partially-completed CIRCA plan.

Of course, during the planning process the system might realize that an abstract state that has already been reasoned about is not sufficiently detailed. For example, this occurs when the state description is not sufficiently refined to indicate whether a desirable action can, in fact, be executed (because the state description does not specify values for all of the features in the action's preconditions). In such situations, the planner must be able to dynamically increase the precision of that abstract state description by including one or more of the omitted features. We call this process of adding detail a "split" or "refinement."

In the language of finite automata, DAP starts with a very crude NFA and dynamically adds more detail. DAP refines the NFA when it is unable to generate a satisfactory plan[1] at the current level of detail. DAP refines the NFA by taking an existing state and splitting it into a number of more specific states, one for each possible value of a particular feature, F_i.

For example, let us consider the partially-completed plan given in Figure 3. Here there are three states: the failure state and two non-failure states, one for each value of `emergency`, a boolean proposition. This example is based on the domain model given in Figure 2. We assume that `emergency` is `nil` when the system begins operation.

The NFA in Figure 3 is not safe, because there is a reachable state, S_1, from which there is a transition to the failure state (`emergency-failure`) that has not been preempted. One way to fix this problem

[1] We will be more clear about what is "satisfactory" below.

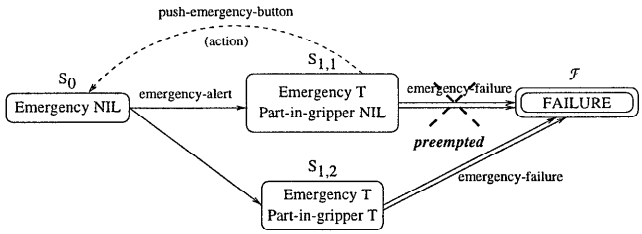

Figure 4: A refinement of the NFA in Figure 3.

would be to choose an action for S_1 that will preempt `emergency-failure`. The domain description contains such an action, `push-emergency-button`. Unfortunately, one of `push-emergency-button`'s preconditions is `part-in-gripper= nil` and S_1 is not sufficiently detailed to specify values for `part-in-gripper`. We can rectify this omission by splitting S_1 into a set of states, one for each value of `part-in-gripper`. The resulting NFA is given in Figure 4. We can now assign `push-emergency-button` to solve the problem posed by state $S_{1,1}$. Further planning is required to resolve the problem posed by $S_{1,2}$, either by finding a preempting action that does not require `part-in-gripper = nil` or by making $S_{1,2}$ unreachable.

One unusual aspect of DAP is that detail is added to the NFA only *locally*. In our example above, we only added the feature `part-in-gripper` to the part of the state space where the `emergency` feature took on the value `true`, rather than refining all of the states of the NFA symmetrically. This introduces new nondeterminism: because we do not have a complete model of the initial state, we cannot say whether the `emergency-alert` transition will send the system to state $S_{1,1}$ or $S_{1,2}$.

DAP in Theory

During its operation, DAP manipulates NFAs of a particular type. An NFA, $\mathcal{N} = \langle v(\mathcal{N}), e(\mathcal{N})\rangle$, will have a number of states (or vertices), $S_i \in v(\mathcal{N})$, each of which corresponds to a set of feature-value pairs; we will refer to these as $f(S_i)$. A state S_i *necessarily* satisfies a proposition, P ($S_i \models \Box P$) if $P \in f(S_i)$; it *possibly* satisfies P ($S_i \models \Diamond P$) if $\neg P \notin f(S_i)$ (these boolean definitions may be straightforwardly extended to non-boolean features).

The transitions in the NFA are generated by the transition descriptions, which are nondeterministic STRIPS operators. A transition t is possibly (respectively, necessarily) executable in a state when the transition's preconditions are all possibly (necessarily) satisfied by that state: $S_i \models \Diamond pre(t)(\Box pre(t))$. With some abuse of notation, for each transition t we define a function $t(S)$ from a state to a formula (in the general case, a disjunction), describing the state(s) that result from executing t in S.

DAP must construct NFAs in which there are no chains of non-preempted, possibly-executable transitions that lead to a failure states. To preempt a temporal transition in a state, DAP assigns to that state a *necessarily* executable action that can be executed before the preempted transition.

We maintain NFAs that contain edges for all possibly executable non-preempted event and temporal transitions (we refer to these collectively as "non-volitional transitions") and for all currently-assigned actions.

The refinement (or splitting) operation on an NFA $\mathcal{N}$ with respect to a state S_i and a feature F_j $r(\mathcal{N}, S_i, F_j) = \mathcal{N}'$ is defined as follows:

$$\begin{aligned} S' &= \{S | f(S) = f(S_i) \cup \{(F_j, z)\} \text{ for } z \in val(F_j)\} \\ v(\mathcal{N}') &= (v(\mathcal{N}) - S_i) \cup S' \end{aligned}$$

where S' is the set of newly-added states. New transitions must be added into and out of the replacement states:

$$\begin{aligned} e(\mathcal{N}') = \; & (e(\mathcal{N}) - \{v_1 \to v_2 | v_1 = S_i \text{ or } v_2 = S_i\}) \\ & \cup \{v \xrightarrow{t} S | v \models \Diamond pre(t), S \in S', S \models \Diamond t(v)\} \\ & \cup \{S \xrightarrow{t} v | S \in S', S \models \Diamond pre(t), v \models \Diamond t(S)\} \end{aligned}$$

DAP in Practice

The prototype DAP planner takes as input a domain model in the form of transition descriptions, a description of a set of initial states, and a conjunctive goal expression. The planner returns an NFA containing only reachable states. Each state of the NFA will be labeled with either an action or `no-op`, indicating to CIRCA how the RTS should react in that situation. Failure states will not be reachable in this NFA and the system will move towards states satisfying the goal expression whenever possible.

The planning problem may be very concisely described as a nondeterministic algorithm, given in Figure 5. In this presentation, **choose** and **oneof** are nondeterministic choice operators. An action is applicable if the state necessarily satisfies its preconditions and if the action preempts all transitions to failure from the state. Note that it is not sufficient to preempt transitions directly to the distinguished failure state. For example, if there is a state s with an event transition (i.e., a transition with a zero delay) to the failure state, then any edges into s must also be considered as transitions to failure.

In practice, we implement this algorithm through search, with choice points corresponding to the nondeterministic choice operators. The search fails when it encounters a state for which there is no acceptable action and for which there is no proposition on which

```
abstract-plan (isd);
    isd is initial state description
    let N = ∅; The graph
        openlist = ∅;
        is = make-initial-state(isd);
    N := N ∪ {is};
    push(is, openlist);
    loop
        if there are no more reachable states in the openlist then
          we are done
          break;
        else
          let s = choose a reachable state from openlist;
              openlist := openlist − {s};
              oneof
                  split-state :
                      choose a proposition p and split s into |val(p)| states;
                      remove s from N and insert the new states;
                      add the new states to the open list;
                  assign-action :
                      choose an action (or no-op) that is applicable for s;
                  fail
```

Figure 5: The DAP planning algorithm.

to split. We may not be able to split the state productively even if the state is only partially specified. No further splitting will be productive if we can determine that some bad transition *must* occur in the state, that the state is reachable, and that there are no available actions with which to preempt the bad transition.

The structure of the NFA being constructed guides us in backtracking. When we fail to successfully handle a state, we backjump to the earliest solved state (we keep these on a closed list) that has an edge into the failed state. Because the state is reachable, there must be a state with an edge into it, unless the state is the starting state. If we fail on the starting state, the search as a whole has failed.

Note that we do not backtrack over state refinements. Backtracking over these refinements is never necessary: for every plan that can be found at a low level of detail, there is a corresponding plan at every higher level. Our experience suggests that the cost of "coarsening" an NFA (and the additional bookkeeping necessary to provide this option) is not worth the small savings in graph size.

Through additional backtracking, we provide a simple anytime behavior. The AIS caches plans as they are produced (recall that all plans are safety-preserving). Through backtracking, the AIS can generate plans that satisfy more of the goal propositions. Thus once a first safety-producing plan is generated, the AIS may at will invest more time into generating better plans.

There are two aspects to the heuristic control of the search: the search should be directed to achieve safety and to move the system towards states that satisfy as many goal propositions as possible. To make the search for goal propositions most efficient, the first action the DAP planner takes is to split the initial state according to the goal propositions. The heuristic we use for directing the choice of actions and refinements is a modified version of McDermott's heuristic estimator for state-based ADL planning (McDermott 1996). When choosing how to handle a state, the planner constructs an operator-proposition graph connecting the current state description to the goal state description. This is a layered graph, with alternating layers containing nodes that represent propositions to be achieved and operators that can establish those propositions. Despite using full lookahead, this approach is heuristic and efficient because it ignores details such as interactions between operators.

Our version differs from McDermott's because our actions are simple STRIPS operators; his approach covers schemas as well and must consider variable binding. Another difference is that McDermott's is a more traditional state-space planner, so state descriptions are complete and the only way to establish a proposition is to apply an operator with the appropriate postconditions. Our state descriptions are partial, and one way for the DAP planner to establish a proposition is to refine a partial state description to include that propo-

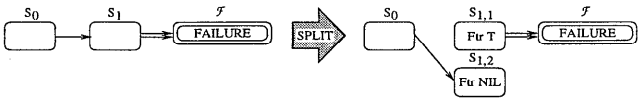

Figure 6: Using refinement to isolate a failure.

sition. Note that this operation is similar to the kind of conditional planning done by CNLP (Peot & Smith 1992) and Plinth (Goldman & Boddy 1994): when the planner cannot determine *a priori* the value of a proposition, it plans for both alternatives.

The planner combines information about the context of a state with the heuristic information provided by the operator-proposition graph. For example, when choosing between several interesting propositions on which to refine a state, the planner will prefer those that are established by some transition leading into the state.

As we mentioned earlier, the planner must concern itself with safety as well as goal achievement. One place where this difference becomes significant is when backtracking from a bad state (a state is bad if it has an unpreemptable path to the failure state). In this case, the planner will work to avoid the failure. There are two ways to do this: either avoid actions that lead to the bad state or refine the bad abstract state, to demonstrate that the sub-states in which the bad transition(s) occur are not, in fact, reachable (for example, see Figure 6). Safety concerns also intrude when none of the goal-directed actions available at a state are fast enough to preempt a transition that would lead to failure. Safety is always the paramount consideration, causing the planner to choose an action not preferred by the heuristics in this case.

Implementation Status & Preliminary Results

The prototype DAP planner is implemented and running on a selection of example domains that were used in the original CIRCA research. The DAP planner reasons about safety preserving goals of avoidance and optional goals of achievement in much the same way as the original CIRCA planner, except that it does not yet consider the detailed temporal model necessary to ensure failure preemption in all cases. Manual inspection of the prototype's output plans shows that they are very similar to the original planner's; the new planner chooses the same actions for the same states, but does not yet correctly derive the timing requirements on all of those actions.

Given these limitations, comparisons between the two planners are still only approximate. However, initial results are dramatic. Figure 7 shows several representative cases, some with nearly an order of magnitude reduction in search space using DAP. In the

	Enumerated States		Runtime (sec)	
Domain Name	Original Planner	DAP Planner	Original Planner	DAP Planner
Puma 1	826	89	222	1.13
Xdemo 2	28	9	0.43	0.08
Puma 3	76	16	8.59	0.09
Puma 4	330	71	68.3	0.59
BT 6	7	7	0.08	0.04
Puma 9	212	41	58.8	0.33

Figure 7: The DAP planner dramatically reduces the search space and time.

Puma 1 domain, which is one of the largest problems to which CIRCA has been applied, the DAP planner is able to find significant structure in the domain that the original CIRCA planner cannot exploit. For example, the DAP plan is able to describe all of the conditions in which to take the `push-emergency-button` action as a disjunction of just three abstract state descriptions, while the original CIRCA planner selects that action for 54 different fully-described states.

The BT 6 domain is a small, hand-crafted problem designed to force the original CIRCA planner to backtrack through several decisions, thus exercising the backtracking and worst-case state space enumeration of the planner. The domain has only one state feature, so the DAP planner can find no suitable abstraction and it makes the same backtracking moves as the original planner, yielding the same search-space performance. To date, this is the only domain in which the DAP technique has not yielded any performance improvement. Other simple domains, such as the Xdemo 2 domain (which has only 5 state features), still contain enough hidden structure that the DAP technique is able to find and exploit feasible abstractions.

Related Work

Many classical planning systems have used abstraction methods to increase the efficiency of searching for plans (see (Kambhampati 1994) for a brief survey). However, these abstractions are typically used only as guides in searching for a plan; the system may not know that its goals will actually be achieved by an abstract plan, and it will not be able to execute the abstracted operators directly. Instead, traditional abstraction planners must eventually expand their current plans down to the lowest level of detail, removing the abstraction to produce a final executable plan.

In the DAP approach, which involves abstraction only of state descriptions, abstract plans are executable, because the operators are always completely specified. This has two main advantages. First, the

planning process can supply initial plans that preserve safety but might, on further refinement, do a better job of goal achievement. Second, the planning process can terminate with an executable abstract plan, which our results have shown may be much smaller than the corresponding plan expanded to precisely-defined states.

Dearden and Boutilier (1997) have developed an abstract planning algorithm for decision-theoretic planning modeled as a Markov decision process (MDP). Their method is similar to the DAP approach in that it involves aggregating states, but there are some differences. First, their method is not dynamic: aggregation is performed using a predefined set of "relevant" propositions, which is determined using Knoblock's approach (Knoblock 1994). Second, their method is uniform: the same propositions are relevant everywhere. The underlying model is also significantly different from CIRCA's: it does not model exogenous events or the timing required for real-time guarantees.

Kabanza *et al.* (Kabanza, Barbeau, & St-Dcnis 1997) have developed a planning method for reactive agents that is similar to the original CIRCA. Their architecture differs in emphasis, however. The NFAs it constructs are "clocked:" they make transitions at times that are the least common denominator of all possible transitions. This scheme will suffer a state space explosion in domains where there is a wide range of possible transition delays, like those to which CIRCA has been applied. Kabanza's group has concentrated on developing a more flexible notation for goals than those used by CIRCA, but they do not make the same distinction between safety and goal achievement. In previous work, Godefroid and Kabanza (Godefroid & Kabanza 1991) developed an abstraction technique based on partial orders. Their results allow a system to examine only a single ordering of independent actions, rather than enumerating all possible orderings. Unfortunately, these results are not immediately applicable to CIRCA, because their world model does not include exogenous events. The more recent work by Kabanza *et al.* (Kabanza, Barbeau, & St-Denis 1997) *does* include exogenous events, but they do not seem to have carried over the earlier abstraction concepts.

Future Directions

In this paper, we have presented Dynamic Abstraction Planning (DAP), an abstraction technique that we use to generate real-time control plans in the CIRCA system. This abstraction technique is significantly different from others in preserving safety guarantees and in performing abstraction locally and dynamically. In our experience, by automatically selecting the appropriate level of abstraction at each step during the planning process, DAP significantly reduces the size of the search space.

The main next step in developing the DAP methodology is to fully integrate the detailed temporal reasoning that the current prototype omits. This will bring the new planner onto equal footing with the original CIRCA planner, and will allow more accurate comparisons of the efficiency improvements gained by using the dynamic abstraction method.

Acknowledgments This work was supported by the Defense Advanced Research Projects Agency under contract DAAK60-94-C-0040-P0006. We thank the reviewers for their helpful comments.

References

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1–2):219–283.

Godefroid, P., and Kabanza, F. 1991. An efficient reactive planner for synthesizing reactive plans. In *Proc. Nat'l Conf. on Artificial Intelligence*, 640–645.

Goldman, R. P., and Boddy, M. S. 1994. Conditional linear planning. In *Proc. Second Int'l Conf. on Artificial Intelligence Planning Systems*, 80–85.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning control rules for reactive agents. Technical Report 197, Comp. Sci. Dept., Univ. of Sherbrooke.

Kambhampati, S. 1994. Refinement search as a unifying framework for analyzing planning algorithms. In *Proc. Fourth Int'l Conf. on Principles of Knowledge Representation and Reasoning.*

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243–302.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int'l Conf. on Artificial Intelligence Planning Systems*, 142–149.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Trans. Systems, Man, and Cybernctics* 23(6):1561–1574.

Musliner, D. J.; Durfee, E. II.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proc. First Int'l Conf. on Artificial Intelligence Planning Systems*, 189–197.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5(2):115–135.