Synthesis of Reactive Plans for Multi-Path Environments*

F. Kabanza
Université de Liège
Institut Montéfiore B28
4000 Liège Sart-Tilman; Belgium
U519026@BLIULG11.bitnet

Abstract

We describe a planner that works on the description of a multi-path environment and generates a conditional plan. The resulting plan is guaranteed to fulfill its goal whatever path of the description the environment follows during the plan execution.

Introduction

Classical planning systems [Fikes *et al.*, 1971; Cohen and Feigenbaum, 1982; Wilkins, 1984] assume that the planning agent operates in a static environment (world). That is, at every moment of time, the world is in a given state and that state remains unchanged until an action is performed by the agent.

Yet most interesting environments in which an intelligent agent is expected to operate are not static. Most realistic worlds are changing and nondeterministic. The assumption that the world remains static between successive actions of the agent no longer holds. Furthermore, the agent usually has only nondeterministic information on how the world is going to evolve.

Recent work has considered the problem of planning in such environments. Allen and Koomen [Allen and Koomen, 1983] use an interval temporal logic to model the past, the present, and the future knowledge of the agent. The changes of the environment and the nondeterminism of its behavior are conveyed by formulas of the logic. McDermott [McDermott, 1982] uses a branching structure (tree of chronsets) to represent a nondeterministic future. Pednault [Pednault, 1987] uses a STRIP-like representation of actions to model some changing worlds, especially in motion problems for which one can give a prediction of the state of the world after an action. Lansky [Lansky, 1987] uses a point-based logic and an event-based representation of states to express synchronization properties between agents in a multiagent domain. Dean [Dean, 1987] describes a framework for scheduling tasks with imposed deadline constraints. In that approach, the planner can use statistical information to predict the future.

In this paper, we also consider the problem of planning in a multi-path environment. We introduce a planning method that is inspired by recent developments in program synthesis from temporal logic specifications [Abadi *et al.*, 1989; Pnueli and Rosner, 1989a; Pnueli and Rosner, 1989b]. The multi-path environment is viewed as a tree of states, which we call a *world-automaton*. Each path in the tree represents a possible behavior of the environment. We call *environment* knowledge the knowledge represented by the world-automaton. We assume that the planner has no control over this tree and that he cannot predict which path the world is actually going to follow. The *operational* knowledge of the plan is given by a set of actions. The planning problem can now be stated as follows: given an environment characterized by a world-automaton, an operational knowledge, an initial state, and a goal, find a (conditional) plan that is guaranteed to achieve the goal whatever path of the world-automaton the environment actually follows.

To handle this problem, one first needs a formalism for describing the environment knowledge. One could give the set of world states and the transition relation between these states, but such a description would be tedious and hence prone to error in complex environments. Instead, we use propositional branching time temporal logic (CTL [Emerson and Clarke, 1982]). For instance, let us assume that the world-automaton is specified by the CTL formula "$\exists\Box$guard $\wedge$ $\forall\Diamond$dark" stating that it might be the case that there is always a guard present and that it will definitely end up being dark. Consider the goal "**rob**(\$20 million) from Bank" and an initial state where it is not dark. Then the plan that is generated should work whether a guard is present or not, though it can wait for darkness to appear since this is guaranteed to happen.

The problem of generating the plan is related to that of synthesizing a reactive module as described in [Rosenschein, 1989; Pnueli and Rosner, 1989a; Pnueli and Rosner, 1989b]. A reactive program is one that continuously interacts with its environment dur-

*The following text presents research results of the Belgian National incentive-program for fundamental research in artificial intelligence initiated by the Belgian State - Prime Minister's Office - Science Policy Programming. The scientific responsibility is supported by its author.

ing execution. The environment controls some variables, the program others. The synthesis problem consists in building a program that satisfies a given specification (expressed in temporal logic) for all possible behaviors of the environment (also expressed in temporal logic). This is done by first building a formula expressing the statement : "for any execution of the environment, there is an execution of the program that satisfies the specification". One then uses a decision procedure to generate a model for this formula. One could adapt the algorithms of [Pnueli and Rosner, 1989a; Pnueli and Rosner, 1989b] to the synthesis of plans. Unfortunately these methods are of very high computational complexity (double exponential), which makes them of limited use in practice.

The approach we follow proceeds in three steps. First, we generate a description of the world-automaton from the CTL formula describing the environment. This can be done with the algorithms described in [Emerson and Clarke, 1982; Emerson and Halpern, 1985; Manna and Wolper, 1984] and summarized in [Wolper, 1989]. Then, from this world-automaton, the operational knowledge, and the goal, we build a graph representing a synchronization of the executions of the agent and of the environment. A node of this graph represents an action that has to be executed by the agent. A transition between two nodes is labeled by an event of the world-automaton, that is, of the environment. Finally, we extract the desired plan from this graph. This approach allows us to exploit classical heuristic methods (as in SIPE [Wilkins, 1984]) to only explore a fragment of the search space of possible action sequences.

In the next section, we start with a formal description of world-automata and plans. We then discuss the specification language and the algorithm that automatically generates the world-automaton from a CTL specification. The following section describes the planner. The last section is devoted to other features of the planner and to possible extensions.

Describing Environment Knowledge and Plans

Actions, Events, Goals, and Strategies

Actions are described, as in classical systems such as STRIPS or SIPE [Fikes *et al.*, 1971; Wilkins, 1984], essentially by their precondition and their effect. However, we also allow an exclusive disjunction as the effect of an action to represent an effect that depends on the state in which the action is applied. We will call actions with disjunctive effects *disjunctive actions*. The action $NULL$ means "do nothing".

Goals are also described, as in classical systems, by a set of propositions. But in addition, we allow *conditional goals* of the form $p \rightarrow a$, where p is a proposition and a is an action. Such a goal means "do a every time p is true".

A *strategy* is a set of linearly ordered goals. The order must be safe, that is, if a goal g_1 precedes a goal g_2, then all actions leading to g_1 have to be performed before any action for g_2 during execution. Therefore a goal is a special strategy. Strategies are more or less similar to Chapman's "cognitive cliches" [Chapman and Agre, 1987].

Environment Knowledge and Plans

A world-automaton is a tuple $W = (Q, E, \gamma, next, q)$, where

$Q = \{q_1, \ldots q_n\}$ is a set of states;

$E = \{NOTH, e_1, \ldots e_m\}$ is a set of events (the event $NOTH$ is a special event indicating that nothing (interesting) happens in the environment);

$\gamma : Q \rightarrow E$ is a function mapping states to events;

$next : Q \rightarrow 2^Q$ is the transition relation between states;

$q \in Q$ is the initial state.

The fact that there is only one initial state is not restrictive. If an automaton has many initial states we can make them the successors of an initial state with event $NOTH$. An execution of a world-automaton is defined as in classical automata theory. We start in the initial state q. The first event to appear is $\gamma(q)$. Being in state q_i means that the event $\gamma(q_i)$ is the last one to have occurred.

The *environment knowledge* is described by a world-automaton. The possible executions of this automaton constitute an infinite tree. Each path of this tree describes a possible behavior of the environment. Henceforth, we will use the terms "execution", "behavior", and "path" of the environment interchangeably.

The *plans* we generate are also represented by world-automata. An automaton representing a plan contains both the actions (events) of the environment and those of the planning agent. To execute a plan, an agent performs the actions in the plan that are his own and waits for the environment to perform all other actions.

Events can be generic, that is, have many possible instances. But, at any moment, only one instance of an event may happen. For example, the event $on(a, x)$ has $on(a, table)$ and $on(a, b)$ among its instances. In the world-automaton, a transition happens when any instance of the corresponding event happens.

Example 1 Suppose we have to build a reactive plan for robot *Agent2* stacking on table $T2$ parts produced on a separate table $T1$ by another robot *Agent1*. Agent1 produces parts by groups of four: blocks a, b, c and d. It stacks each block on a separate rack appropriate for its type. When a rack is occupied, the agent waits until it becomes clear. The block b is always produced just after the block a, and d is always produced just after c. In figures and in formulas, we abbreviate "rack i" by r_i.

The environment knowledge for Agent2 is thus the production activity of Agent1. It is described by the world-automaton of Figure 1. A part of the plan that our method generates for Agent2 is given in Figure 2. In this figure, all actions that have to be performed by Agent2 are prefixed by *ad*.

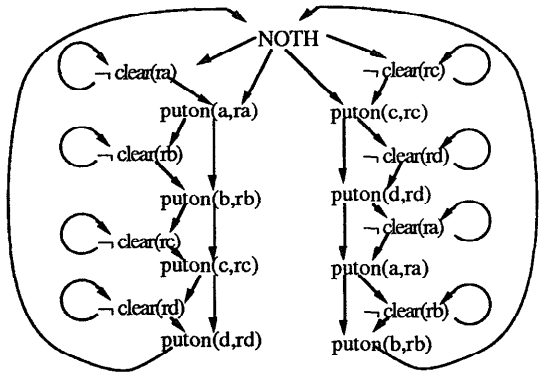

Figure 1: Environment for Agent2

∎

Specifying the World-Automaton

We use propositional branching time temporal logic (CTL) to describe the environment. This logic is formally defined in [Emerson and Clarke, 1982; Wolper, 1989]. We only introduce it informally here.

CTL formulas are built from atomic propositions, boolean connectives, and eight temporal operators. A temporal operator is a path quantifier directly followed by a modality operator. The path quantifiers are $\forall$ meaning "for all execution paths", and $\exists$ meaning "there is an execution path". The modality operators are $\Diamond$ ("eventually"), $\Box$ ("always"), $\bigcirc$ ("next") and U "until". They are all unary, except U which is binary. Proposional formulas (i.e. without temporal operator) are the basic CTL formulas. Any formula built by prefixing a CTL formula with a temporal operator is also a CTL formula. Examples of CTL formulas are : $\exists\Diamond accepted \wedge \forall(\neg presented\, U\, accepted)$.

Formulas are interpreted over infinite trees of propositions. They can be naturally interpreted over world-automata.

- A formula is satisfied by an interpretation if it is satisfied by the initial state of the interpretation.
- A proposition is satisfied by a state if it is true (in the classical sense) of the event of that state.
- $\exists\bigcirc f$ means "*possibly next f*". Its dual $\forall\bigcirc f$ means "*surely next f*".
- $\exists\Diamond f$ means "*possibly f will happen*". Its dual $\forall\Diamond f$ means "*surely eventually f*".
- $\exists\Box f$ means "*possibly it is the case that always f*", whereas $\forall\Box f$ means "*surely always f*".
- $\exists(fUg)$ means "*possibly it is the case that f holds until g*", and $\forall(fUg)$ means "*surely f holds until g*"

For instance, the world-automaton of Figure 2 satisfies the formula $\exists\Diamond puton(d, rd)$.

Generating the World-Automaton

We use the algorithm of [Emerson and Clarke, 1982] to generate the world-automaton from a CTL specification. The worst case complexity of the algorithm is exponential in the size of the formula. However, with an implementation that uses all possible optimizations, it often gives good results.

Example 2 The following formulas are the bulk of the specification from which the world-automaton for *Agent2* (Figure 1) has been generated.

1. A block is produced if its rack is empty
   $\forall\Box(puton(a, r_a) \supset clear(r_a))$
   (Similar axioms for b, c and d.)
2. Every time r_a is empty, a will eventually be produced
   $\forall\Box(clear(r_a) \supset \forall\Diamond puton(a, r_a))$
   (A similar axiom is given for r_c.)
3. The order of block production: when a is produced, the following block must be b
   $\forall\Box(puton(a, r_a) \supset$
   $\forall(\neg(puton(a, r_a) \vee puton(c, r_c) \vee puton(d, r_d))$
   $U\, puton(b, r_b)))$
   (Similarly for c and d.)
4. At each cycle we do not know the first block to be produced
   $\exists\Diamond puton(a, r_a) \wedge \exists\Diamond puton(c, r_c)$

∎

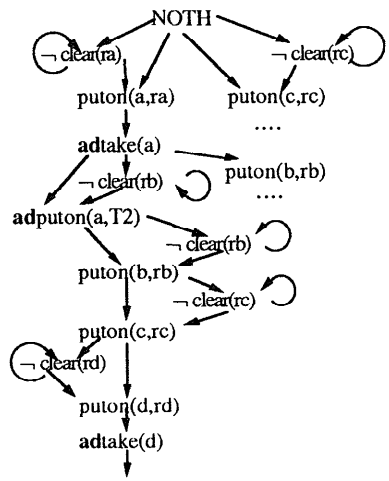

Figure 2: Plan for Agent2

The Planner

The General Idea

We start with the following data: a world-automaton, an operational knowledge, and a strategy (set of goals). From this, we first build an intermediate structure. This structure is a directed graph whose nodes are labeled with actions of the operational knowledge and whose transitions are labeled with events of the world-automaton. This graph represents a synchronization of the agent trying to complete the goals with the environment: at a node of the graph, the agent executes the associated action; it then waits for an event labeling a transition from that node. When the event happens, the agent goes to the node indicated by the transition labeled by the event, executes the action corresponding to that node, and waits again. Actions of the graph are chosen by the planner so that they lead to the goals.

It then remains to extract a plan from the constructed graph. As mentioned in the knowledge description section, the plan has the form of a world-automaton. Hence, it can be used as input for another planning agent.

Building the graph is thus the core of the algorithm. We associate the following additional information to each node of the graph: a state of the world-automaton, a goal to be completed within the node, and a stack of strategies to be performed by descendants of the node (the part of the initial strategy that still has to be completed). Recall that the strategy is the goal of the agent.

We build the graph starting from a state in which the associated node of the world-automaton is the initial state of that automaton, the strategy is the overall goal, and the action is *NULL*. The graph is then built incrementally as follows.

- A graph node with no successor and with a nonempty strategy is chosen.
- An action whose precondition is satisfied and that reduces the distance from the goal is chosen (nondeterministically).
- Successor nodes are constructed for each possible action: the action chosen for the agent and all possible environment actions.

Of course, backtracking is used to implement the nondeterministic choices. Also, goal-reduction is used: when the precondition of the chosen action cannot be satisfied, we attempt to apply goal reduction to the action. If this is impossible, we look ahead in the world-automaton to see if more promising states are coming up and we wait for them; if this is not the case, we backtrack.

The Algorithm

In this paper, we sketch the planning algorithm with the following restrictions: there are no conditional goals, no generic events, and the operational knowledge does not contain disjunctive actions.

The planning algorithm operates in one of two modes: wait and reduce. In the wait mode, it is searching the world-automaton for an event to happen. In the reduce mode, it is trying to find an action that fulfills a goal. We use the following convention for variables: s and s' for nodes of the graph, e for events of the world-automaton. S is the set of nodes of the graph and E is the set of events of the environment. The data-structures used by the planning algorithm are:

- The world-automaton $(Q, E, \gamma, next, q_0)$;
- The intermediate structure (graph) in which:

  S is a set of nodes of the graph,
  $succ : (S \times E) \rightarrow S$ is the graph transition function.
  The information associated with each node s consists of: an action $action(s)$, a world-automaton state $was(s)$, a goal $G(s)$, and a strategy $stack(s)$;
- The variable $Mode$ that takes values W (wait) and R (reduce);
- The event waited upon when in mode W is *waited-event*.

A sketch of construction of the intermediate graph is given in Figure 3.

The *goal-reduction* is any of the heuristics available in AI [Wilkins, 1984], used to decompose a goal into primitive subgoals. Any of these heuristics can be applied since we have the same problem: find a sequence of actions to complete a goal $G(s)$ from an initial state q_0, given a set of actions (the operational knowledge).

Nodes can be fused when they are defined by the same elements (action, stack,...). Hence loops might be introduced. The planning algorithm has to check that the loops that are introduced are safe. This is done with techniques similar to those used to check that *eventuality formulas* are satisfied when building models of temporal logic formulas [Emerson and Clarke, 1982; Manna and Wolper, 1984].

The plan is extracted from the intermediate structure as follows. First build a new structure from the intermediate structure, by introducing between two successive nodes a new node labeled with the transition label between those nodes. The plan is induced from that last structure by skipping *NULL* actions and *NOTH* events.

Example 3 Consider the environment knowledge of Example 1. Suppose we are given the following operational knowledge:

Actions

$take(x)$	precond: $on(x,z) \wedge clear(x)$
	add-list: $have(x) \wedge clear(z)$
$puton(x,y)$	precond: $have(x) \wedge clear(y)$
	add-list: $on(x,y)$.

The goal of Agent2 is to arrange blocks on a different table $T2$, as follows:

$$on(a, T2) \wedge on(d, a) \wedge on(b, d) \wedge on(c, b).$$

Construct-Intermed-Struct(strategy)
Create an initial node s with $G(s) = NULL$, $stack(s)$= strategy, and $was(s) = q_0$.
Set $Mode$ to R.
Repeat
Select a node s with $stack(s)$ not empty and with no successors. If there is no such node, break from the loop.
If $Mode$=W **then:**

Check if $was(s)$ =*waited-event*. If so, set $Mode$ to R. If not, set $action(s)$ to $NULL$; for each event e successor of $was(s)$, create the node $succ(s,e)$, set $G(succ(s,e))$ to $G(s)$, and $stack(succ(s,e))$ to $stack(s)$.

Else:

-If $G(s)$ is not an atomic action reduce it to a sequence of atomic actions by applying goal-reduction. If such a sequence can be found then update $G(s)$ and $stack(s)$ and continue.

-From now, we can assume $G(s)$ is atomic. If the precondition of $G(s)$ is not satisfied use the verification algorithm of [Clarke *et al.*, 1986] to see if the precondition of $G(s)$ can be satisfied by further states of the world-automaton; if it can, set $Mode$ to W, and set *waited-event* to the precondition of $G(s)$; else backtrack.

-If the precondition of $G(s)$ is satisfied, set $action(s)$ to $G(s)$; then determine the strategy and the stack for successors nodes : for each event successor of $was(s)$, create a new node s' of the graph and a transition, labeled by the event, from the current node to the new node s'; set $G(s')$ to the first element of $stack(s)$ and $stack(s')$ to the rest of $stack(s)$.

End of Repeat
End of Construct-Intermed-Struct

Figure 3: Constructing the intermediate structure

It is reduced to the strategy:
$take(a) \rightarrow puton(a,T2) \rightarrow take(d) \rightarrow puton(d,a) \rightarrow take(b) \rightarrow puton(b,d) \rightarrow take(c) \rightarrow puton(c,b)$.

The initial part of the plan generated is shown in Figure 2. ∎

Additional Features and Possible Extensions

The following features have also been developed: the use of past temporal formulas to express conditions of conditional goals. We keep a linear structure recording the past activity. We can then use a polynomial algorithm to verify if past preconditions of actions are satisfied on that structure. The verification algorithm is similar to that of [Clarke *et al.*, 1986]. We can also use temporal formulas to describe the effects of an action. This is a partial response to the commitment problem [Cohen and Levesque, 1987]. When an action with a temporal effect is selected at a state in the intermediate structure, we replace the sub-automaton rooted at the corresponding state in the world-automaton by a cross-poduct of the sub-automaton with an automaton representing the effects formula.

If there are paths of the world-automaton on which it is impossible for the agent to achieve its goal, there is no solution to the planning problem. However, it seems reasonable for the agent to move ahead anyway making the assumption that the path for which he cannot plan is unlikely. This implies that it would be useful to have probabilistic information for each path of the world automaton. The planner would then use that information to estimate the probability that the plan will work. Another interesting approach is for the planner to procrastinate when, for some goals, the future is very uncertain. He would delay planning for those goals until information about the future becomes more precise.

Automatic generation of environment specifications is a problem we intend to investigate, possibly using techniques similar to those in [Kautz and Allen, 1986; Kautz, 1987] to construct a prediction module.

Planning is also expected to be interleaved with the execution. During execution some information could confirm or disconfirm the current structure of the world-automaton. The planner can then determine whether or not the plan could go awry and, if necessary, replan with respect to the new environment structure.

Conclusions

We have shown how temporal logic can be fruitfully used for generating reactive plans. The problem we considered is that of planning for agents evolving in a changing and nondeterministic world. The generation algorithm is in fact a blend of goal-reduction and environment enabling, and it can be viewed in the tread of recent work done by [Bresina and Drummond, 1990].

The performance of this planner depends on the available strategies. The lower level the strategies are, the faster the plan generation is going to be. We believe the method viable if refined strategies are available. Such refined strategies are analogous to those used in human behavior. If you are told to go from *Liège* to *Brussels*, you immediately decide to take a bus to the station, where you hope to catch a train. If you know there might not be a bus, you immediately plan to take a cab. That is, you act as if you had strategies triggered by goals you have to commit to. You don't think "to be in *Brussels* I must have been in a train; to be in a train I must have been to the station, and so on." Such reasoning only appears in rather unusual situations (lost in a town, ...).

Clearly the method, as it stands now, has some limitations. The world-automaton has to be correct for the

plan to be correct too. Thus there is need for some way of dealing with unpredictable events. Another limitation is the seemingly great complexity of the algorithm. With all that in mind, we expect this work will serve as a good guideline for future investigations towards a tractable planner.

Acknowledgments

I would like to thank two anonymous referees for helpful suggestions for a better presentation of this paper. I am also grateful to Marianne Baudinet, Jean-Marc Stevenne, and Pierre Wolper for careful reading of this paper and thoughtful comments. Philippe Simar helped in the typesetting of this paper. This work has been made possible only through the enthusiastic supervision of Professor Pierre Wolper, to whom I extend my thanks.

References

Abadi, M.; Lamport, L.; and Wolper, P. 1989. Realizable and unrealizable concurrent program specifications. In *Proc. 10th Int. Colloquium on Automata, Languages and Programming*. LNCS, Vol. 372, Springer-Verlag. 1–17.

Allen, J.F. and Koomen, J.F. 1983. Planning using a temporal world wodel. In Bundy, A., editor 1983, *8th IJCAI*. 741–747.

Bresina, J. and Drummond, M. 1990. Integrating planning and reaction. In *AAAI workshop on Planning in Uncertain, Unpredictable, or Changing Environments*, Stanford Univ.

Chapman, D. and Agre, P. E. 1987. Abstract reasoning as emergent from concrete activity. In Georgeff, M. P. and Lansky, A., editors 1987, *Reasonning about Actions and Plans, Proceedings of the 1986 Workshop, Timberline, Oregon*. Morgan Kaufmann. 411–424.

Clarke, E.M.; Emerson, E.A.; and Sistla, A.P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244–263.

Cohen, P. R. and Feigenbaum, E. A. 1982. *Handbook of Artificial Intelligence*. Pitman, London.

Cohen, P. R. and Levesque, H. J. 1987. Persistence, intention, and commitment. In Georgeff, M. P. and Lansky, A., editors 1987, *Reasonning about Actions and Plans, Proceedings of the 1986 Workshop, Timberline, Oregon*. Morgan Kaufmann. 297–340.

Dean, Thomas L. 1987. Intractability and time-dependent planning. In Georgeff, M. P. and Lansky, Amy, editors 1987, *Reasonning about Actions and Plans, Proceedings of the 1986 Workshop, Timberline, Oregon*. Morgan Kaufmann. 245–266.

Emerson, E.A. and Clarke, E.M. 1982. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming* 2:241–266.

Emerson, E.A. and Halpern, J.Y. 1985. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences* 30:1–24.

Fikes, R.; Hart, P. E.; and Nilsson, N.J. 1971. STRIPS: A new approach to the application of theorem proving. *Artificial Intelligence* 2:189–208.

Kautz, H. A. and Allen, J. F. 1986. Generalized plan recognition. In *5th NCAI*. AAAI, Morgan Kaufmann. 32–37.

Kautz, H. A. 1987. A formal theory of plan recognition. Technical Report 215, Univ. of Rochester, NY.

Lansky, A. 1987. A representation of parallel activity based on events, structure, and causality. In Georgeff, M. P. and Lansky, A., editors 1987, *Reasonning about Actions and Plans, Proceedings of the 1986 Workshop, Timberline, Oregon*. Morgan Kaufmann. 123–159.

Manna, Z. and Wolper, P. 1984. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 6(1):68–93.

McDermott, D. 1982. A temporal logic for reasoning about processes and plans. *Cognitive Science* 6(2):101–155.

Pednault, E. P.D. 1987. Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff, M. P. and Lansky, A., editors 1987, *Reasonning about Actions and Plans, Proceedings of the 1986 Workshop, Timberline, Oregon*. Morgan Kaufmann. 47–82.

Pnueli, A. and Rosner, R. 1989a. On the synthesis of a reactive module. In *16th Annual ACM Symposium on POPL*, Austin. 179–189.

Pnueli, A. and Rosner, R. 1989b. On the synthesis of an asynchronous reactive module. In *ICALP*. LNCS, Vol 372, Springer-Verlag. 652–671.

Rosenschein, S. J. 1989. Synthesizing information-tracking automata from environment descriptions. In Brachman, R. J.; Levesque, H. J.; and Reiter, R., editors 1989, *First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto. Morgan Kaufmann. 386–393.

Wilkins, D. E. 1984. Domain-independant planning: Representation and plan generation. *Artificial Intelligence* 22(3):269–301.

Wolper, P. 1989. On the relation of programs and computations to models of temporal logic. In Banieqbal, B.; Barringer, H.; and Pnueli, A., editors 1989, *Proc. Temporal Logic in Specification*. LNCS, Vol. 398, Springer-Verlag. 75–123.