ITC irst

CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

Planning with Extended Goals and Partial Observability

Bertoli ., Pistore .

December 2003

Technical Report # T03−12−18

Planning with Extended Goals and Partial Observability*

Piergiorgio Bertoli
ITC-irst - Trento - Italy
bertoli@irst.itc.it

Marco Pistore
University of Trento - Italy
pistore@dit.unitn.it

Abstract

Planning in nondeterministic domains with temporally extended goals under partial observability is one of the most challenging problems in planning. Simpler subsets of this problem have been already addressed in the literature, but the general combination of extended goals and partial observability is, to the best of our knowledge, still an open problem. In this paper we present a first attempt to solve the problem, namely, we define an algorithm that builds plans in the general setting of planning with extended goals and partial observability. The algorithm builds on the top of the techniques developed in the planning with model checking framework for the restricted problems of extended goals and of partial observability.

Introduction

In the last years increasing interest has been devoted to planning in nondeterministic domains, and different research lines have been developed. On one side, planning algorithms for tackling temporally extended goals have been proposed in (Kabanza, Barbeau, & St-Denis 1997; Pistore & Traverso 2001; Dal Lago, Pistore, & Traverso 2002), motivated by the fact that many real-life problems require temporal operators for expressing complex requirements. This research line is carried out under the assumption that the planning domain is fully observable. On the other side, in (Bertoli *et al.* 2001; Weld, Anderson, & Smith 1998; Bonet & Geffner 2000; Rintanen 1999) the hypothesis of full observability is relaxed in order to deal with realistic situations, where the plan executor cannot access the whole status of the domain. The key difficulty is in dealing with the uncertainty arising from the inability to determine precisely at run-time what is the current status of the domain. These approaches are however limited to the case of simple reachability goals.

Tackling the problem of planning for temporally extended goals under the assumption of partial observability is not trivial. In (Bertoli *et al.* 2003), a framework for plan validation under these assumptions was introduced; however, no plan generation algorithm was provided. In this paper we complete the framework of (Bertoli *et al.* 2003) with a planning algorithm. The algorithm builds of the top of techniques for planning with extended goals (Pistore & Traverso 2001) and for planning under partial observability (Bertoli *et al.* 2001). The algorithm in (Pistore & Traverso 2001) is based on goal progression, that is, it traces the evolution of goals along the execution of the plan. The algorithm of (Bertoli *et al.* 2001) exploits belief states, which describe the set of possible states of the domain that are compatible with the actions and the observations performed during plan execution. The main structure of the planning algorithm described in this paper are belief-desires. A belief-desire extends a belief by associating subgoals to its states. Our planning algorithm is based on progressing subgoals of belief-desires in a forward-chaining setting. However, belief-desires alone are not sufficient, as an ordering of active subgoals must be kept in order to guarantee progression in the satisfaction of goals. For this reason, we also rely on a notion of intention, meant as the next active subgoal that has to be achieved.

The paper is structured as follows. We first recap the framework proposed in (Bertoli *et al.* 2003); then, we introduce the key notions of the algorithm, i.e., desires and intentions, and describe the algorithm; finally, we discuss related and propose some concluding remarks.

The framework

In the framework for planning proposed in (Bertoli *et al.* 2003), a domain is a model of a generic system, such as a power plant or an aircraft, with its own dynamics. The plan can control the evolutions of the domain by triggering *actions*. At execution time, the state of the domain is only partially visible to the plan, via an *observation* of the state. In essence, planning is building a suitable plan that can guide the evolutions of the domain in order to achieve the specified goals. Further details and examples on the framework can be found in (Bertoli *et al.* 2003).

Planning domains

A planning domain is defined in terms of its *states*, of the *actions* it accepts, and of the possible *observations* that the domain can exhibit. Some of the states are marked as valid *initial states* for the domain. A *transition function* describes how (the execution of) an action leads from one state to possibly many different states. Finally, an *observation function* defines what observations are associated to each state of the domain.

Definition 1 (planning domain) *A* nondeterministic planning domain with partial observability *is a tuple* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$*, where:*

- $\mathcal{S}$ *is the set of* states*.*
- $\mathcal{A}$ *is the set of* actions*.*
- $\mathcal{O}$ *is the set of* observations*.*

*This research has been partly supported by the ASI Project I/R/271/02 (DOVES).

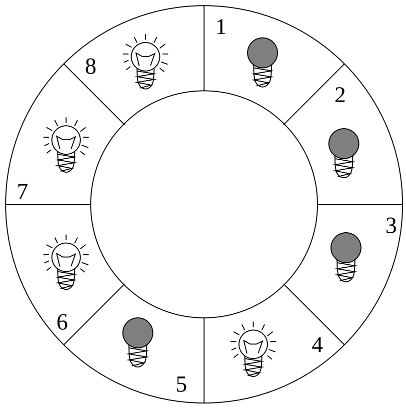

Figure 1: A simple domain.

- $\mathcal{I} \subseteq \mathcal{S}$ *is the set of* initial states*; we require* $\mathcal{I} \neq \emptyset$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}}$ *is the* transition function*; it associates to each current state* $s \in \mathcal{S}$ *and to each action* $a \in \mathcal{A}$ *the set* $\mathcal{T}(s, a) \subseteq \mathcal{S}$ *of next states. We require that* $\mathcal{T}(s, a) \neq \emptyset$ *for all* $s \in \mathcal{S}$ *and* $a \in \mathcal{A}$*, i.e., every action is executable in every state.*[1]
- $\mathcal{X} : \mathcal{S} \to 2^{\mathcal{O}}$ *is the* observation function*; it associates to each state* s *the set of possible observations* $\mathcal{X}(s) \subseteq \mathcal{O}$*. We require that some observation is associated to each state* $s \in \mathcal{S}$*, that is,* $\mathcal{X}(s) \neq \emptyset$.

Technically, a domain is described as a nondeterministic Moore machine, whose outputs (i.e., the observations) depend only on the current state of the machine, not on the input action. Uncertainty is allowed in the initial state and in the outcome of action execution. The mechanism of observations allowed by this model is very general. It can model *no observability* and *full observability* as special cases; moreover, since the observation associated to a given state is not unique, it is possible to represent noisy sensing and lack of information.

Following a common praxis in planning, in the following example, that will be used throughout the paper, we use *fluents* and *observation variables* to describe states and observations of the domain.

Example 1 *Consider the domain represented in Figure 1. It consists of a ring of* N *rooms. Each room contains a light that can be on or off, and a button that, when pressed, switches the status of the light. A robot may move between adjacent rooms (actions* go-right *and* go-left*) and switch the lights (action* switch-light*). Uncertainty in the domain is due to an unknown initial room and initial status of the lights. Moreover, the lights in the rooms not occupied by the robot may be nondeterministically switched on without the direct intervention of the robot (if a light is already on, instead, it can be turned off only by the robot). The domain is only partially observable: the rooms are indistinguishable and the robot can sense only the status of the light in its current room.*

A state of the domain is defined in terms of fluent room*, that ranges from* 1 *to* N *and describes in which room the robot is currently in, and of boolean fluents* on$[i]$*, with* $i \in \{1, \ldots, N\}$*, that describe whether the light in room* i *is on. Any state is a possible initial state.*
The actions are go-left*,* go-right*,* switch-light*, and* wait*. Action* wait *corresponds to the robot doing nothing during a transition (the state of the domain may change only due to the lights that may be turned on without the intervention of the robot). The effects of the other actions have been already described.*
The observation is defined in terms of observation variable light*. It is true if and only if the light is on in the current room.*

[1]This requirement allows us to simplify the planning algorithm. It is easy to allow for actions that are executable only in some states, at the cost of some additional complexity in the planning algorithm.

Plans

Now we present a general definition of plan, that encodes sequential, conditional and iterative behaviors, and is expressive enough for dealing with partial observability and with extended goals. In particular, we need plans where the selection of the action to be executed depends on the observations, and on an "internal state" of the executor, that can take into account, e.g., the knowledge gathered during the previous execution steps. A plan is defined in terms of an *action function* that, given an observation and a *context* encoding the internal state of the executor, specifies the action to be executed, and in terms of a *context function* that evolves the context.

Definition 2 (plan) *A* plan *for planning domain* $\mathcal{D}$ *is a tuple* $P = \langle C, c_0, \mathit{act}, \mathit{evolve}\rangle$*, where:*

- C *is the set of* plan contexts.
- $c_0 \in C$ *is the* initial context.
- *act* : $C \times \mathcal{O} \rightharpoonup \mathcal{A}$ *is the* action function*; it associates to a plan context* c *and an observation* o *an action* $a = \mathit{act}(c, o)$ *to be executed.*
- *evolve* : $C \times \mathcal{O} \rightharpoonup C$ *is the* context evolutions function*; it associates to a plan context* c *and an observation* o *a new plan context* $c' = \mathit{evolve}(c, o)$.

Technically, a plan is described as a Mealy machine, whose outputs (the action) depends in general on the inputs (the current observation). Functions *act* and *evolve* are deterministic (we do not consider nondeterministic plans), and can be partial, since a plan may be undefined on context-observation pairs that are never reached during execution.

Example 2 *We consider a plan for the domain of Figure 1. This plan causes the robot to move cyclically through the rooms, turning off the lights whenever they are on. The plan is cyclic, that is, it never ends. The plan has two contexts* E *and* L*, corresponding, respectively, to the robot having just entered a room, and the robot being about to leave the room after switching the light. The initial context is* E*. Functions act and evolve are defined by the following table:*

c	o	$\mathit{act}(c,o)$	$\mathit{evolve}(c,o)$
E	light $= \top$	switch-light	L
E	light $= \bot$	go-right	E
L	any	go-right	E

Plan execution

Now we discuss plan execution, that is, the effects of running a plan on the corresponding planning domain. Since both the plan and the domain are finite state machines, we can use the standard techniques for synchronous compositions defined in model checking. That is, we can describe the execution of a plan over a domain in terms of transitions between configurations that describe the state of the domain and of the plan. This idea is formalized in the following definition.

Definition 3 (configuration) *A* configuration *for domain $\mathcal{D}$ and plan P is a tuple (s, o, c, a) such that:*

- $s \in \mathcal{S}$,
- $o \in \mathcal{X}(s)$,
- $c \in C$, *and*
- $a = act(c, o)$.

Configuration (s, o, c, a) may evolve into configuration (s', o', c', a'), written $(s, o, c, a) \rightarrow (s', o', c', a')$, if $s' \in \mathcal{T}(s, a)$, $o' \in \mathcal{X}(s')$, $c' = evolve(c, o)$, and $a' = act(c', o')$. Configuration (s, o, c, a) is initial *if $s \in \mathcal{I}$ and $c = c_0$. The* reachable configurations *for domain $\mathcal{D}$ and plan P are defined by the following inductive rules:*

- *if (s, o, c, a) is initial, then it is reachable;*
- *if (s, o, c, a) is reachable and $(s, o, c, a) \rightarrow (s', o', c', a')$, then (s', o', c', a') is also reachable.*

Notice that we include the observations and the actions in the configurations. In this way, not only the current states of the two finite states machines, but also the information exchanged by these machines are explicitly represented. In the case of the observations, this explicit representation is necessary since more than one observation may correspond to the same state.

We are interested in plans that define an action to be executed for each reachable configuration. These plans are called *executable*.

Definition 4 (executable plan) *Plan P is* executable *on domain $\mathcal{D}$ if:*

1. if $s \in \mathcal{I}$ and $o \in \mathcal{X}(s)$ then $act(c_0, o)$ is defined;

and if for all the reachable configurations (s, o, c, a):

2. $evolve(c, o)$ is defined;

3. if $s' \in \mathcal{T}(s, a)$, $o' \in \mathcal{X}(s')$, and $c' = evolve(c, o)$, then $act(c', o')$ is defined.

Condition 1 guarantees that the plan defines an action for all the initial states (and observations) of the domain. The other conditions guarantee that, during plan execution, a configuration is never reached where the execution cannot proceed. More precisely, condition 2 guarantees that the plan defines a next context for each reachable configuration. Condition 3 is similar to condition 1 and guarantees that the plan defines an action for all the states and observations of the domain that can be reached from the current configuration.

An execution path of the plan is basically a sequence of configurations $(s_0, o_0, c_0, a_0) \rightarrow (s_1, o_1, c_1, a_1) \rightarrow (s_2, o_2, c_2, a_2) \rightarrow \cdots$. Due to the nondeterminism in the domain, we may have an infinite number of different executions of a plan. We provide a finite presentation of these executions with an *execution structure*, i.e, a Kripke Structure (Emerson 1990) whose set of states is the set of reachable configurations of the plan, and whose transition relation corresponds to the transitions between configurations.

Definition 5 (execution structure) *The* execution structure *corresponding to domain $\mathcal{D}$ and plan P is the Kripke structure $K = \langle Q, Q_0, R \rangle$, where:*

- *Q is the set of reachable configurations;*
- *$Q_0 = \{(s, o, c_0, a) \in Q : s \in \mathcal{I} \wedge o \in \mathcal{X}(s) \wedge a = act(c_0, o)\}$ are the initial configurations;*
- $R = \big\{\big((s, o, c, a), (s', o', c', a')\big) \in Q \times Q : (s, o, c, a) \rightarrow (s', o', c', a')\big\}$.

Temporally extended goals: CTL

Extended goals are expressed with temporal logic formulas. In most of the works on planning with extended goals (see, e.g., (Kabanza, Barbeau, & St-Denis 1997; de Giacomo & Vardi 1999; Bacchus & Kabanza 2000)), Linear Time Logic (LTL) is used as goal language. LTL provides temporal operators that allow one to define complex conditions on the sequences of states that are possible outcomes of plan execution. Following (Pistore & Traverso 2001), we use Computational Tree Logic (CTL) instead. CTL provides the same temporal operators of LTL, but extends them with universal and existential path quantifiers that provide the ability to take into account the non-determinism of the domain.

We assume that a set $\mathcal{B}$ of basic propositions is defined on domain $\mathcal{D}$. Moreover, we assume that for each $b \in \mathcal{B}$ and $s \in \mathcal{S}$, predicate $s \models_0 b$ holds if and only if basic proposition b is true on state s. In the case of the domain of Figure 1, possible basic propositions are $\mathsf{on}[i]$, that is true in those states where the light is on in room i, or $\mathsf{room}{=}i$, that is true if the robot is in room i.

Definition 6 (CTL) *The goal language* CTL *is defined by the following grammar, where b is a basic proposition:*

$$g ::= \mathrm{A}(g \,\mathrm{U}\, g) \mid \mathrm{E}(g \,\mathrm{U}\, g) \mid \mathrm{A}(g \,\mathrm{W}\, g) \mid \mathrm{E}(g \,\mathrm{W}\, g) \mid \\ g \wedge g \mid g \vee g \mid b \mid \neg b$$

We denote with $\mathrm{cl}(g)$ the set of the sub-formulas of g (including g itself). We denote with $\mathrm{cl}_U(g)$ the subset of $\mathrm{cl}(g)$ consisting of the strong until sub-formulas $\mathrm{A}(\_ \,\mathrm{U}\, \_)$ and $\mathrm{E}(\_ \,\mathrm{U}\, \_)$.

CTL combines temporal operators and path quantifiers. "U" and "W" are the "(strong) until" and "weak until" temporal operators, respectively. "A" and "E" are the universal and existential path quantifiers, where a path is an infinite sequence of states. They allow us to specify requirements that take into account nondeterminism. Intuitively, the formula $\mathrm{A}(g_1 \,\mathrm{U}\, g_2)$ means that for every path there exists an initial prefix of the path such that g_2 holds at the last state of the prefix and g_1 holds at all the other states along the prefix. The formula $\mathrm{E}(g_1 \,\mathrm{U}\, g_2)$ expresses the same condition, but only on some of the paths. The formulas $\mathrm{A}(g_1 \,\mathrm{W}\, g_2)$ and $\mathrm{E}(g_1 \,\mathrm{W}\, g_2)$ are similar to $\mathrm{A}(g_1 \,\mathrm{U}\, g_2)$ and $\mathrm{E}(g_1 \,\mathrm{U}\, g_2)$, but allow for paths where g_1 holds in all the states and g_2 never holds. Formula $\mathrm{AF}\, g$ ($\mathrm{EF}\, g$) is an abbreviation of $\mathrm{A}(\top \,\mathrm{U}\, g)$ (resp. $\mathrm{E}(\top \,\mathrm{U}\, g)$). It means that goal g holds in some future state of every path (resp. some paths). Formula $\mathrm{AG}\, g$ ($\mathrm{EG}\, g$) is an abbreviation of $\mathrm{A}(g \,\mathrm{W}\, \bot)$ (resp. $\mathrm{E}(g \,\mathrm{W}\, \bot)$).

It means that goal g holds in all states of every path (resp. some path).[2]

A remark is in order. Even if negation $\neg$ is allowed only in front of basic propositions, it is easy to define $\neg g$ for a generic CTL formula g, by "pushing down" the negations: for instance $\neg \mathrm{A}(g_1 \, \mathrm{W} \, g_2) \equiv \mathrm{E}(\neg g_2 \, \mathrm{U}(\neg g_1 \wedge \neg g_2))$.

Goals expressed as CTL formulas allow specifying different classes of requirements on plans, e.g., reachability goals, safety or maintainability goals, and arbitrary combinations of them. Examples of CTL goals can be found in (Pistore & Traverso 2001; Bertoli *et al.* 2003).

We now define when CTL formula g is true in configuration (s, o, c, a) of execution structure K, written $K, (s, o, c, a) \models g$. We use the standard semantics for CTL formulas over Kripke Structures (Emerson 1990).

Definition 7 (semantics of CTL) *Let K be an execution structure. We define $K, q \models g$ as follows:*

- $K, q \models \mathrm{A}(g \, \mathrm{U} \, g')$ *if for all $q = q_0 \to q_1 \to q_2 \to \cdots$ there is some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.*
- $K, q \models \mathrm{E}(g \, \mathrm{U} \, g')$ *if there is some $q = q_0 \to q_1 \to q_2 \to \cdots$ and some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.*
- $K, q \models \mathrm{A}(g \, \mathrm{W} \, g')$ *if for all $q = q_0 \to q_1 \to q_2 \to \cdots$, either $K, q_j \models g$ for all $j \geq 0$, or there is some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.*
- $K, q \models \mathrm{E}(g \, \mathrm{W} \, g')$ *if there is some $q = q_0 \to q_1 \to q_2 \to \cdots$ such that either $K, q_j \models g$ for all $j \geq 0$, or there is some $i \geq 0$ such that $K, q_i \models g'$ and $K, q_j \models g$ for all $0 \leq j < i$.*
- $K, q \models g \wedge g'$ *if $K, q \models g$ and $K, q \models g'$.*
- $K, q \models g \vee g'$ *if $K, q \models g$ or $K, q \models g'$.*
- $K, q \models b$ *if $q = (s, o, c, a)$ and $s \models_0 b$.*
- $K, q \models \neg b$ *if $q = (s, o, c, a)$ and $s \not\models_0 b$.*

We define $K \models g$ if $K, q_0 \models g$ for all the initial configurations $q_0 \in Q_0$ of K.

Plan validation

The definition of when a plan satisfies a goal follows.

Definition 8 (plan validation for CTL goals) *Plan P satisfies CTL goal g on domain $\mathcal{D}$, written $P \models_{\mathcal{D}} g$, if $K \models g$, where K is the execution structure for $\mathcal{D}$ and P.*

In the case of CTL goals, the *plan validation* task amounts to CTL model checking. Given a domain $\mathcal{D}$ and a plan P, the corresponding execution structure K is built as described in Definition 5 and standard model checking algorithms are run on K in order to check whether it satisfies goal g.

We describe now some goals for the domain of Figure 1. We recall that the initial room of the robot is uncertain, and that light can be turned on (but not off) without the intervention of the robot.

[2] CTL also includes temporal operators AX and EX that allow to express conditions on the next states. These operators are not very useful for defining planning goals. Moreover, the planning algorithm requires some extensions in order to work with AX and EX operators. For these reasons, we leave the management of these operators for an extended version of this paper.

Example 3 *The first goal we consider is*

$$\mathrm{AF}\,(\neg \mathit{on}[3]),$$

which requires that the light of room 3 is eventually off. The plan of Example 2 satisfies this goal: eventually, the robot will be in room 3 and will turn off the light if it is on.
There is no plan that satisfies to following goal:

$$\mathrm{AF}\,\mathrm{AG}\,(\neg \mathit{on}[3]),$$

which requires that the light in room 3 is turned off and stays then off forever. This can be only guaranteed if the robot stays in room 3 forever, and it is impossible to guarantee this condition in this domain: due to the partial observability of the domain, the robot does never know it is in room 3.
The plan of Example 2 satisfies the following goal

$$\mathrm{AG} \bigwedge_{i \in 1,\ldots,N} \mathrm{AF}\,(\neg \mathit{on}[i]),$$

which requires that the light in every room is turned off infinitely often. On the other hand, it does not satisfy the following goal

$$\mathrm{AG}\,\mathrm{AF} \bigwedge_{i \in 1,\ldots,N} (\neg \mathit{on}[i]),$$

which requires that the lights in all the rooms are off at the same time infinitely often. Indeed, the nondeterminism in the domain may cause lights to turn on at any time.

The planning algorithm

This section describes a planning algorithm for domains with partial observability and CTL goals. The algorithm is based on a forward-chaining approach, that incrementally adds new possible contexts to the plan. Intuitively, contexts are associated to belief-desire structures; a belief-desire relates sets of states compatible with past actions and observations to subgoals that have to be achieved in such states. The plan search is based on progressing a belief and the associated goals; for explanatory purposes, we introduce goal progression by considering single states first, and introduce belief-desires later on.

Progressing goals

Let us assume that we want to satisfy goal g in a state q. Goal g defines conditions on the current state and on the next states to be reached. Intuitively, if g must hold in q, then some conditions must be projected to the next states. The algorithm extracts the information on the conditions on the next states by "progressing" the goal g. For instance, if g is $\mathrm{EF}\, g'$, then g holds in q if either g' holds in q or $\mathrm{EF}\, g'$ hold in some next state. If g is $\mathrm{A}(g' \, \mathrm{U} \, g'')$, then either g'' holds in q, or g' holds in q and $\mathrm{A}(g' \, \mathrm{U} \, g'')$ hold in every next state. In the next example, the progression of goals is shown on a more complex case.

Example 4 *Consider the goal*

$$g = \mathrm{AG}\,(\mathit{on[1]} \wedge \mathrm{EF}\,\neg \mathit{on[2]} \wedge \mathrm{EF}\,\mathit{on[3]}):$$

- *if in room 1 the light is off, then there is no way of progressing the goal, since the goal is unsatisfiable;*
- *if in rooms 1 and 2 the light is on, and in room 3 it is off, g must hold in all next states and both $\mathrm{EF}\,\neg \mathit{on}[2]$ and $\mathrm{EF}\,\mathit{on}[3]$ must hold in some next state;*

- *if in room 1 the light is on and in rooms 2 and 3 it is off, then g must hold in all next states and* $\mathrm{EF}\,\textsf{on}[3]$ *must hold in some next state;*
- *if in rooms 1, 2, and 3 the light is on, then g must hold in all next states and* $\mathrm{EF}\,\neg\textsf{on}[2]$ *must hold in some next state;*
- *if in rooms 1, 3 the light is on and in room 2 it is off, then g must hold in all next states.*

The progression of goals is described by function *progr*, that takes as arguments a state and a goal and rewrites the goal in terms of conditions to be satisfied on the next states. In general, $progr(q, g)$ describes a disjunction of possible alternative ways for projecting a given goal into the next states. Each disjunct describes a goal assignment for the next states, as a conjunction of conditions. In each disjunct, we distinguish the conditions that must hold in all the next states (A_i) from those that must hold in some of the next states (E_i). Thus we represent $progr(q, g)$ as a set of pairs, each pair containing the A_i and the E_i parts of a disjunct:

$$progr(q, g) = \{(A_i, E_i) \mid i \in I\}.$$

We remark that $progr(q, g) = \emptyset$ means that goal g is unsatisfiable in q, since there is no possibility to progress it successfully, while $progr(q, g) = (\{(\emptyset, \emptyset)\})$ means that goal g is always true in q, i.e., no conditions need to be progressed to the next states.

Definition 9 (goal progress) *Let q be a state and g be a goal. Then $progr(q, g)$ is defined by induction on the structure of g, as follows:*

- $progr(q, b) =$ *if* $b \in q$ *then* $\{(\emptyset, \emptyset)\}$ *else* $\emptyset$;
- $progr(q, \neg b) =$ *if* $b \in q$ *then* $\emptyset$ *else* $\{(\emptyset, \emptyset)\}$;
- $progr(q, g_1 \wedge g_2) = \{(A_1 \cup A_2, E_1 \cup E_2) : (A_1, E_1) \in progr(q, g_1)$ *and* $(A_2, E_2) \in progr(q, g_2)\}$;
- $progr(q, g_1 \vee g_2) = progr(q, g_1) \cup progr(q, g_2)$;
- $progr(q, \mathrm{A}(g \,\mathrm{U}\, g')) = \{(A \cup \{\mathrm{A}(g \,\mathrm{U}\, g')\}, E) : (A, E) \in progr(q, g)\} \cup progr(q, g')$;
- $progr(q, \mathrm{E}(g \,\mathrm{U}\, g')) = \{(A, E \cup \{\mathrm{E}(g \,\mathrm{U}\, g')\}) : (A, E) \in progr(q, g)\} \cup progr(q, g')$;
- $progr(q, \mathrm{A}(g \,\mathrm{W}\, g')) = \{(A \cup \{\mathrm{A}(g \,\mathrm{W}\, g')\}, E) : (A, E) \in progr(q, g)\} \cup progr(q, g')$;
- $progr(q, \mathrm{E}(g \,\mathrm{W}\, g')) = \{(A, E \cup \{\mathrm{E}(g \,\mathrm{W}\, g')\}) : (A, E) \in progr(q, g)\} \cup progr(q, g')$.

Let G be a set of goals. Then $progr(q, G) = progr(q, \bigwedge_{g \in G} g)$.

We remark that untils and weak untils progress in the same way. In fact, the difference between these two operators can only been defined considering infinite behaviors.

Given a disjunct $(A, E) \in progr(q, g)$, we can define a function that assigns goals to be satisfied to the next states. We denote with $assign((A, E), Q)$ the set of all the possible assignments $a : Q \to 2^{A \cup E}$ such that each universally quantified goal is assigned to all the next states (i.e., if $f \in A$ then $f \in a(q)$ for all $q \in Q$) and each existentially quantified goal is assigned to one of the next states (i.e., if $h \in E$ and $h \notin A$ then $f \in a(q)$ for one particular $q \in Q$).

Definition 10 (goal assignment) *Let $(A, E) \in progr(q, g)$ and Q be a set of states. Then $assign((A, E), Q)$ is the set of all possible assignments $a : Q \to 2^{A \cup E}$ such that:*

- *if $h \in A$ then $h \in a(q)$ for all $q \in Q$; and*
- *if $h \in E - A$, then $h \in a(q)$ for a unique $q \in Q$.*

With slight abuse of notation, we define:

$$assign(progr(q, g), Q) = \bigcup_{(A,E) \in progr(q,g)} assign((A, E), Q).$$

Example 5 *Consider again the goal* $g = \mathrm{AG}\,(\textsf{on[1]} \wedge \mathrm{EF}\,\neg\textsf{on[2]} \wedge \mathrm{EF}\,\textsf{on[3]})$, *and let the current state be* $\textsf{room}{=}1, \textsf{on}[1], \textsf{on}[2], \neg\textsf{on}[3]$. *In this case we have* $progr(g, q) = (\{g\}, \{\mathrm{EF}\,\neg\textsf{on}[2], \mathrm{EF}\,\textsf{on}[3]\})$.
If we execute action go-right*, then the next states are* $s_1 = (\textsf{room}{=}2, \textsf{on}[1], \textsf{on}[2], \neg\textsf{on}[3])$ *and* $s_2 = (\textsf{room}{=}2, \textsf{on}[1], \textsf{on}[2], \textsf{on}[3])$. *Then g must hold in s_1* and *in s_2, while* $\mathrm{EF}\,\neg\textsf{on}[2]$ *and* $\mathrm{EF}\,\textsf{on}[3]$ *must hold* either *in s_1* or *in s_2. We have therefore four possible state-formulas assignments $a_1, \ldots, a_4$ to be considered:*

$$a_1(s_1) = \{g, \mathrm{EF}\,\neg\textsf{on}[2], \mathrm{EF}\,\textsf{on}[3]\},\ a_1(s_2) = \{g\}$$
$$a_2(s_1) = \{g, \mathrm{EF}\,\neg\textsf{on}[2]\},\ a_2(s_2) = \{g, \mathrm{EF}\,\textsf{on}[3]\}$$
$$a_3(s_1) = \{g, \mathrm{EF}\,\textsf{on}[3]\},\ a_3(s_2) = \{g, \mathrm{EF}\,\neg\textsf{on}[2]\}$$
$$a_4(s_1) = \{g\},\ a_4(s_2) = \{g, \mathrm{EF}\,\neg\textsf{on}[2], \mathrm{EF}\,\textsf{on}[3]\}$$

Belief-desires

In this section we extend goal progression taking into account that in planning with partial observability we have only a partial knowledge of the current state, described in terms of a belief-state.

Let us assume that initially we have goal g_0. This goal is associated to all states of the initial belief. The progress of a goal depends on the current state, therefore, from now on the goal progresses independently for each of the possible states. This leads to an association of different goals to the different states of the beliefs. A *belief-desire* describes this association between states and goals, as in the following definition.

Definition 11 (belief-desire) *A* belief-desire *for domain D and goal g_0 is a set $bd \in \mathcal{S} \times \mathrm{cl}(g_0)$. We represent with* $states(bd) = \{q : (q, g) \in bd\}$ *the states of belief-desire bd.*

The belief-desire can be used in the planning algorithm to define the context of the plan that is being built, by associating a belief-desire to each context of the plan. Functions *progr* and *assign* are used to describe the valid evolutions of the belief-desire along plan execution.

Indeed, let us assume that the bd is the current belief-desire. If we perform action a then the belief-desire evolves in order to reflect the progress of the goals associated to the states. More precisely, the new belief-desires have to respect the following condition. Let $bd'_{(q,g)}$ be a belief-desire representing a goal progression for $(q, g) \in bd$, namely

$$bd'_{(q,g)} \in assign(progr(q, g), \mathrm{Exec}(q, a)).$$

Then every belief-desire bd' that collects such elementary goal progressions, namely $bd' = \bigcup_{(q,g) \in bd} bd'_{(q,g)}$, is a valid new belief-desire.

Intentions

The belief-desire carries most of the information that is needed to describe the contexts in the planning algorithm. However, the next example shows that there are cases where the belief-desire does not carry enough information to decide the next action to execute.

Example 6 *Consider a ring of 8 rooms and the goal* $g_0 = \mathrm{AG}\,(\neg(\textit{room}{=}6) \wedge \mathrm{AF}\,\neg\textit{on}[1] \wedge \mathrm{AF}\,\neg\textit{on}[5])$ *and, for simplicity, let us assume that the robot knows its current room (e.g., since the initial room is 1). It is easy to check that the execution of any plan that satisfies the goal should traverse infinitely often room 3, going back and forth between room 1 and room 5. Whenever the execution is in room 3, the active belief-desire is the following, independently on whether we are coming from room 1 or from room 5:*

$\textit{room}{=}3, \neg\textit{on}[1], \neg\textit{on}[5] \mapsto g_0$
$\textit{room}{=}3, \textit{on}[1], \neg\textit{on}[5] \mapsto g_0, \mathrm{AF}\,\neg\textit{on}[1]$
$\textit{room}{=}3, \neg\textit{on}[1], \textit{on}[5] \mapsto g_0, \mathrm{AF}\,\neg\textit{on}[5]$
$\textit{room}{=}3, \textit{on}[1], \textit{on}[5] \mapsto g_0, \mathrm{AF}\,\neg\textit{on}[1], \mathrm{AF}\,\neg\textit{on}[5]$

However, the action to be executed in room 3 depends on whether we are moving towards room 1 or towards room 5. In order to allow for these different moves, it is necessary to have different contexts associated to the same belief-desire.

The previous example shows that an ingredient is still missing, namely we need to distinguish contexts not only according to the belief-desire that they represent, but also according to a specific goal that we intend to fulfill first. This goal can be seen as the current *intention* of the plan. For instance, in the scenario described in the previous example we could have two possible intentions, namely turning out the light in room 1 (i.e., $\mathrm{AF}\,\neg\mathsf{on}[1]$), or turning out the light in room 5 (i.e., $\mathrm{AF}\,\neg\mathsf{on}[5]$). In the first case, we would move towards room 1, in the second towards room 5.

Intentions correspond to the strong until formulas in $\mathrm{cl}_U(g_0)$. Indeed, these are the goals describing properties that need to be eventually achieved. An extra intention $\star$ is used to describe those cases where no strong until goal is active.

In the planning algorithm, the active intention is associated with a set of states that describes what are the states from which the active intention has to be achieved. We call *belief-intention* an intention together with its associated set of states.

Definition 12 (belief-intention) *A* belief-intention *for planning domain* $\mathcal{D}$ *and goal* g_0 *is a pair* $bi = (i, b_i) \in (\mathrm{cl}_U(g_0) \cup \{\star\}) \times 2^Q$*. We require that, if* $i = \star$ *then* $b_i = \emptyset$ *(i.e.,* $\star$ *is associated to an empty set of states).*

The belief-intention evolves along with the belief-desire during the execution of the plan. More precisely, let us assume that the current belief-desire is bd and that the current belief-intention is $bi = (i, b_i)$. Assume also that the belief-desire evolves according to the family $bd'_{(q,g)}$ of elementary goal progressions (hence $bd' = \bigcup_{(c,g)\in bd} bd'_{(q,g)}$). Then the new set of states associated to intention i is defines as follows: $b' = \{q' : (q', i) \in bd'_{(q,i)}$ for some $q \in b_i\}$.

If $b' = \emptyset$, then intention i has been fulfilled, and a new strong until can become active. In the following we assume that the next strong until is chosen according to a fixed arbitary ordering $g_1, g_2, \ldots, g_k$ of the formulas in $\mathrm{cl}_U(g_0)$. More precisely, in order to find the new intention, we exploit function *next*, defined as $\textit{next}(g_h) = g_{(h \bmod k)+1}$ and $\textit{next}(\star) = g_1$. The function is repeatedely applied starting from i, until we find an intention i' that is associated to some state in bd'. If there is no such intention, then $\star$ is used as the new intention.

Definition 13 (intention update) *Let* $bi = (i, b_i)$ *be a belief intention,* $\{bd'_{(q,g)}\}_{(q,g)\in bd}$ *a family of elementary progressions for* bd*, and* $bd' = \bigcup_{(c,g)\in bd} bd'_{(q,g)}$*. The updated belief intention* $bi' = \textit{update}(bi, \{bd'_{(q,g)}\}_{(q,g)\in bd})$ *is defined as* $bi' = (i', b'_i)$ *where:*

- *if* $b' \neq \emptyset$ *then* $i' = i$ *and* $b'_i = b'$*;*
- *else if there is some* $i'' \in \mathrm{cl}_U(g_0)$*,* $i'' \neq i$ *such that* $(q', i) \in bd'$ *for some state* q'*, then:*
  - $i' = next^n(i)$*, where* $n = \min\{m{>}0 \mid \exists q' \in \mathcal{S}.(q', next^m(i)) \in bd'\}$
  - $b'_i = \{q' \in \mathcal{S} \mid (q', i) \in bd'\}$
- *else* $i' = \star$ *and* $b'_i = \emptyset$*.*

Belief-desires and belief-intentions play complementary roles in the planning algorithm. Belief-desires make it sure that the plan respects all "maintainability" requirements (expressed by the weak-untils) and that it does not prevent the fulfillment of the "reachability" requirements (expressed by strong untils). Belief-intentions add the guarantee that the "reachability" requirements are eventually achieved. Indeed, strong and weak untils are managed in the same way in goal progression (see Definition 9), and the additional requirement of strong untils $\mathrm{A}(g\,\mathrm{U}\,g')$ and $\mathrm{E}(g\,\mathrm{U}\,g')$ of eventually achieving g' is captured by the intentions.

The role of intentions is also reflected in the management of loops in plans. A loop corresponds to reaching a belief-desire and a belief-intention that have been already encountered. This loop is acceptable only if all strong until requirements are fulfilled along the loop. More precisely, if the same intention i is active along the loop, and $i \neq \star$, then the loop is invalid.

The algorithm

Now we describe a planning algorithm *build-plan* that, given a domain $\mathcal{D}$ and a goal g_0, returns either a valid plan or a failure. The algorithm is reported in Figure 2. We assume that the domain and the goal are global variables.

The algorithm performs a depth-first forward-chaining search. Starting from the initial belief-desire and belief-intention, it considers every possible observation, and for every observation, it picks up an action and progresses the belief-desire and belief-intention to successor states. The iteration goes on until a successful plan is found for every possible observation, or the goal cannot be satisfied for some observation. The contexts $c = (bd, bi)$ of the plan are associated to the belief-desires bd and the belief-intentions bi which are considered at each turn of the iteration.

The main function of the algorithm is the recursive function *build-plan-aux*$(c, \textit{pl}, \textit{open})$, which builds the plan for context c. If a plan is found, then it is returned by the function. Otherwise, $\perp$ is returned. Argument *pl* is the plan built so far by the algorithm. Argument *open* is the list of contexts corresponding to the currently open problems: if $c \in$ *open* then we are currently trying to build a plan for context c either in the current instance of function *build-plan-aux* or in some of the invoking instances. Whenever function *build-plan-aux* is called with a context c already in *open*, then we have a loop in the plan. In this case (lines 9-11), function *is-good-loop*$(c, \textit{open})$ is called to check whether the loop is valid or not. If the loop is good, plan *pl* is returned, otherwise function *build-plan-aux* fails.

```
1  function build-plan : Plan
2    $bd_0 = \{(q_0, g_0) : q_0 \in \mathcal{I}\}$
3    if $g_0 \in \mathrm{cl}_U(g_0)$ then $bi_0 = (g_0, \mathcal{I})$
4                       else $bi_0 = (\star, \emptyset)$
5    $pl.c_0 := (bd_0, bi_0)$; $pl.C := \{pl.c_0\}$
6    $pl.act := \emptyset$; $pl.evolve := \emptyset$
7    return build-plan-aux($pl.c_0, pl, [\,]$)

8  function build-plan-aux($c = (bd, (i, b_i))$, pl, open) : Plan
9    if $c$ in open then
10     if is-good-loop($c$, open) then return pl
11                                else return $\bot$
12   foreach $o \in \mathcal{O}$ do
13     if defined $pl.act[c, o]$ then next $o$
14     $bd_o := \{(q, g) \in bd : q \in states(bd) \wedge o \in \mathcal{X}(q)\}$
15     $bi_o := (i, \{q \in b_i : q \in states(bd) \wedge o \in \mathcal{X}(q)\})$
16     foreach $a$ in $\mathcal{A}$ do
17       foreach $\{bd'_{(q,g)}\}_{(q,g) \in bd_o}$ with
18           $bd'_{(q,g)}$ in assign(progr($q, g$), Exec($q, a$)) do
19         $bd' := \bigcup_{(q,g) \in bd_o} bd'_{(q,g)}$
20         $bi' := update(bi_o, \{bd'_{(q,g)}\}_{(q,g) \in bd_o})$
21         $c' := (bd', bi')$
22         $pl' := pl$; $pl'.C := pl'.C \cup \{c'\}$
23         $pl'.act[c, o] := a$; $pl'.evolve[c, o] := c'$
24         $open' :=$ conc($c'$, open)
25         $pl' :=$ build-plan-aux($c', pl', open'$)
26         if ($pl' \neq \bot$) then $pl := pl'$; next $o$
27     return $\bot$
28   return pl

29 function is-good-loop($c = (bd, (i, b_i))$, open) : boolean
30   if $i = \star$ then return true
31   while $c \neq$ head(open) do
32     $(bd', (i', b'_i)) :=$ head(open)
33     if $i \neq i'$ then return true
34     open := tail(open)
35   return false
```

Figure 2: The planning algorithm.

Function *is-good-loop* checks if the loop is acceptable, namely, it checks whether the current intention is $\star$ (line 30) or the intention changes somewhere inside the loop (lines 31-34). In these cases, the loop is good, since all strong until goals are eventually fulfilled. Otherwise (line 35) the loop is bad and has to be discarded.

If context c is not in *open*, then we need to build a plan from this context. We consider all possible observations $o \in \mathcal{O}$ (line 12) and try to build a plan for each of them. If *pl* is already defined on context c and observation o (i.e., (c, o) is in the range of function *act* and hence condition "defined $pl.act[c, o]$" is true), then a plan for the pair has already been found in another branch of the search, and we move on to the next observation (line 13).

If the pair (c, o) is not already in the plan, then the belief-desire and the belief-intention associated to the context are restricted to those states q that are compatible with observation o, namely, $o \in \mathcal{X}(q)$ (lines 14-15). Then the algorithm considers in turn all actions a (line 16) and all new belief-desires bd' corresponding to valid goal progressions (lines 17-19); updates the belief-intention bi' according to the goal chosen progression (line 20); and takes $c' = (bd', bi')$ as the new context. Function *build-plan-aux* is called recursively on c'. In the recursive call, argument *pl* is updated to take into account that action a has been selected for context c and observation o, and that c' is the evolved context associated to c and o (lines 22-23). Moreover, the context c' is added in front of list *open* (line 24).

If the recursive call *build-plan-aux* is successful, then we move on to the next observation (line 26). Once all observations have been handled, function *build-plan-aux* ends with success and returns plan *pl* (line 28). The different goal progressions and the different actions are tried until, for an observation, a plan is found; if no plan is found, then no plan can be built for the current context c and observation o and $\bot$ is returned (line 27).

Function *build-plan* is defined on the top of function *build-plan-aux*, which is invoked using as initial context (bd_0, bi_0). Belief-desire bd_0 associates goal g_0 to all initial states of the domain. If goal g_0 is a strong until formula, then bi_0 associates all initial states to intention g_0, otherwise the special intention $\star$ is used as initial intention.

The following theorem expresses correctness results on the algorithm.

Theorem 1 *Let $\mathcal{D}$ be a domain and g_0 a goal for $\mathcal{D}$. Then build-plan(g_0) terminates. Moreover, if build-plan$(g_0) = P \neq \bot$, then $P \models_{\mathcal{D}} g_0$. If build-plan$(g_0) = \bot$, then there is no plan P such that $P \models_{\mathcal{D}} g_0$.*

Implementation

In order to test the algorithm, we designed and realized a prototype implementation. The implementation contains some simple optimizations. First of all, some heuristics is used to choose the most suitable action and goal progression, namely, those choices are preferred that lead to smaller belief-desires or belief-intentions. Moreover, loop detection is improved so that a loop is recognized also when the current context is a subset of one of the contexts in list *open*, i.e., it has smaller belief-desires and belief-intentions.

Despite these optimizations, the implemented algorithm has some severe limits. The first limit is the quality of plans. The generated plan for the ring domain with 2 rooms and for goal $\mathrm{AG}\,(\mathrm{AF}\,\neg\mathsf{on}[1] \wedge \mathrm{AF}\,\neg\mathsf{on}[2])$ is the following:

c	o	$act(c, o)$	$evolve(c, o)$
c_0	light $= \bot$	go-left	c_1
c_0	light $= \top$	go-left	c_{10}
c_1	light $= \bot$	go-left	c_0
c_1	light $= \top$	go-left	c_2
c_2	light $= \bot$	go-left	c_3
c_2	light $= \top$	switch	c_9
c_3	*any*	switch	c_4
c_4	*any*	wait	c_5
c_5	*any*	switch	c_6
c_6	*any*	go-left	c_7
c_7	*any*	switch	c_8
c_8	*any*	go-left	c_1
c_9	*any*	go-left	c_3
c_{10}	light $= \bot$	switch	c_9
c_{10}	light $= \top$	switch	c_{11}
c_{11}	*any*	left	c_1

The comparison between the generated plan and the hand-written plan of Example 2 is dramatic. The generated plan is correct, but it is much larger and contains several unnecessary moves. They are due in part to the progression of goals and in part to the depth-first search. A second limit is the efficiency of the search. Indeed, if we consider goal $\mathrm{AG} \bigwedge_{i=1,\ldots,N} \mathrm{AF} \neg \mathsf{on}[i]$ and we increase the number N of rooms, we get the following results (on a PC Pentium 4, 1.8Mhz):

N	#states	search time	#contexts
1	2	0.01 sec	3
2	8	0.22 sec	12
3	24	2.10 sec	50
4	64	35.11 sec	215
5	160	1128.07 sec	1011
6	384	$\gg$ 3600.00 sec	??????

We see that already with $N = 5$ the construction of the plan requires a considerable time and generates a huge plan.

In order to overcome these limitations, and to be able to tackle complex requirements and large-scale domains, we intend to integrate the algorithm with heuristic techniques such as those proposed in (Bertoli, Cimatti, & Roveri 2001; Bertoli *et al.* 2001), and with symbolic techniques such as those in (Pistore, Bettin, & Traverso 2001; Dal Lago, Pistore, & Traverso 2002). Both techniques are based on (extensions of) symbolic model checking, and their successful exploitation would allow for a practical implementation of the planning algorithm.

Concluding remarks

This paper presents a novel planning algorithm that allows generating plans for temporally extended goals under the hypothesis of partial observability. To the best of our knowledge, no other approach has tackled so far this complex combination of requirements.

The simpler problem of planning for temporally extended goals, within the simplified assumption of full observability, has been dealt with in previous works. However, most of the works in this direction restrict to deterministic domains, see for instance (de Giacomo & Vardi 1999; Bacchus & Kabanza 2000). A work that considers extended goals in nondeterministic domains is described in (Kabanza, Barbeau, & St-Denis 1997). Extended goals make the planning problem close to that of automatic synthesis of controllers (see, e.g., (Kupferman, Vardi, & Wolper 1997)). However, most of the work in this area focuses on the theoretical foundations, without providing practical implementations. Moreover, it is based on rather different technical assumptions on actions and on the interaction with the environment.

On the other side, partially observable domains has been tackled either using a probabilistic Markov-based approach (see (Bonet & Geffner 2000)), or within a framework of possible-world semantics (see, e.g., (Bertoli *et al.* 2001; Weld, Anderson, & Smith 1998; Rintanen 1999)). These works are limited to expressing only simple reachability goals. An exception is (Karlsson 2001), where a linear-time temporal logics with a knowledge operator is used to define search control strategies in a progressive probabilistic planner. The usage of a linear-time temporal logics and of a progressive planning algorithm makes the approach of (Karlsson 2001) quite different in aims and techniques from the one discussed in this paper.

In (Bertoli *et al.* 2003) an extension of CTL with knowledge goals $\mathbf{K}\, b$ has been defined. In brief, $\mathbf{K}\, b$ means the executor *knows* that all the possible current states of the domain satisfy condition b. The extension of the algorithm described in this paper to the case of K-CTL goals is simple and does not require the addition of further complexity to the algorithm. Indeed, the evaluation of knowledge goals can be performed on the existing belief-desire structures.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logic to Express Search Control Knowledge for Planning. *Artificial Intelligence* 116(1-2):123–191.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. IJCAI'01*.

Bertoli, P.; Cimatti, A.; Pistore, M.; and Traverso, P. 2003. A Framework for Planning with Extended Goals under Partial Observability. In *Proc. ICAPS'03*, 215–224.

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic Search + Symbolic Model Checking = Efficient Conformant Planning. In *Proc. IJCAI'01*.

Bonet, B., and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proc. AIPS 2000*.

Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. AAAI'02*.

de Giacomo, G., and Vardi, M. 1999. Automata-theoretic approach to Planning with Temporally Extended Goals. In *Proc. ECP'99*.

Emerson, E. A. 1990. Temporal and modal logic. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning Control Rules for Reactive Agents. *Artificial Intelligence* 95(1):67–113.

Karlsson, L. 2001. Conditional progressive planning under uncertainty. In *Proc. IJCAI'01*.

Kupferman, O.; Vardi, M.; and Wolper, P. 1997. Synthesis with incomplete information. In *Proc. ICTL'97*.

Pistore, M., and Traverso, P. 2001. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. IJCAI'01*.

Pistore, M.; Bettin, R.; and Traverso, P. 2001. Symbolic techniques for planning with extended goals in non-deterministic domains. In *Proc. ECP'01*.

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intellegence Research* 10:323–352.

Weld, D.; Anderson, C.; and Smith, D. 1998. Extending Graphplan to Handle Uncertainty and Sensing Actions. In *Proc. AAAI'98*.