Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners

Blai Bonet
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Héctor Palacios
Universidad Simón Bolívar
Caracas, Venezuela
hlp@ldc.usb.ve

Héctor Geffner
ICREA & Univ. Pompeu Fabra
Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

Finite-state and memoryless controllers are simple action selection mechanisms widely used in domains such as videogames and mobile robotics. Memoryless controllers stand for functions that map observations into actions, while finite-state controllers generalize memoryless ones with a finite amount of memory. In contrast to the policies obtained from MDPs and POMDPs, finite-state controllers have two advantages: they are often extremely compact, involving a small number of controller states or none at all, and they are general, applying to many problems and not just one. A limitation of finite-state controllers is that they must be written by hand. In this work, we address this limitation, and develop a method for deriving finite-state controllers automatically from models. These models represent a class of contingent problems where actions are deterministic and some fluents are observable. The problem of deriving a controller from such models is converted into a conformant planning problem that is solved using classical planners, taking advantage of a complete translation introduced recently. The controllers derived in this way are 'general' in the sense that they do not solve the original problem only, but many variations as well, including changes in the size of the problem or in the uncertainty of the initial situation and action effects. Experiments illustrating the derivation of such controllers are presented.

Introduction

Figure 1(a) illustrates a simple 1×5 grid where a robot, initially at one of the two leftmost positions, must visit the rightmost position, marked B, and get back then to A. Assuming that the robot can observe the marks A and B when in the cell, and that the actions *Left* and *Right* deterministically move the robot one unit left and right respectively, the problem can be solved by a contingent planner or a POMDP solver, resulting in the first case in a contingent tree, and a function mapping beliefs into actions in the second (Levesque 1996; Kaelbling, Littman, and Cassandra 1999). A solution to the problem, however, can also be expressed in a simpler manner as the finite-state controller shown in Fig. 1(b). Starting in the controller state q_0, this controller selects the action *Right*, whether mark A or no mark is observed ('$-$'), until observing mark B. Then the controller switches to state q_1 where it selects *Left* as long as no mark is observed.[1]

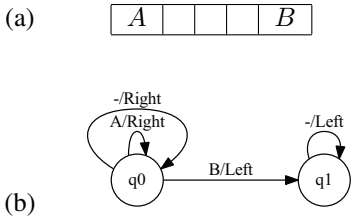

Figure 1: (a) A contingent problem where an agent initially in one of the two leftmost positions has to get to B and then back to A. These two marks are observable. (b) A 2-state controller that solves this problem and many variations. The circles are the controller states, and an edge $q \rightarrow q'$ labeled o/a says to do a when the observation is o in state q, switching then to q'.

The finite-state controller displayed in the figure has two features that make it more appealing than contingent plans and POMDP policies: it is very compact (it involves two states only), and it is very general. Indeed, the problem can be changed in a number of ways and the controller would still work. For example, the *size of the grid* can be changed from 1×5 to $1 \times n$, the agent can be placed *initially* anywhere in the grid (except at B), and the actions can be made *non-deterministic* by the addition of 'noise'. This generality is well beyond the power of contingent plans or exact POMDP policies that are tied to a particular state space. For these reasons, finite-state controllers are widely used in practice, from controlling non-playing characters in videogames (Buckland 2004) to mobile robots (Murphy 2000; Mataric 2007). Memoryless controllers or policies (Littman 1994) are widely used as well, and they are nothing but finite-state controllers with a single state. The additional states provide finite-state controllers with memory that allows different actions to be taken given the same observation.

The benefits of finite-state controllers, however, come at a price: unlike contingent trees and POMDP policies, they are usually not derived automatically from a model but are written by hand; a task that is not trivial even in the simplest cases. There have been attempts for deriving finite-state controllers for POMDPs with a given number of

[1] The problem is a variation of one in (Meuleau et al. 1999).

states (Meuleau et al. 1999; Poupart and Boutilier 2003; Amato, Bernstein, and Zilberstein 2007), but the problem can be solved approximately only with no correctness guarantees.

In this work, we develop a model-based method for deriving finite-state controllers automatically. The models represent a class of contingent problems where actions are deterministic and some fluents are observable. The task of deriving a controller for such models is converted into a conformant planning problem that is solved by a classical planner, taking advantage of a complete transformation reported recently (Palacios and Geffner 2007). A conformant problem is a contingent problem with no sensing where solutions are action sequences like in classical planning.

We will show that the controllers derived automatically in this way are 'general' in the sense that they not only solve the original problem (i.e., for which they are formally correct) but many variations as well, including changes in the size of the problem and in the uncertainty of the initial situation or action effects. We illustrate the nature and the derivation of such controllers through a number of examples.

The paper is organized as follows. We define the problems (Sect. 2), the finite-state controllers (Sect. 3), and establish a correspondence between the controllers that solve a given problem P and the conformant plans that solve a problem P_N (Sect. 4). We then show how to solve this transformed problem by solving a classical planning problem using either an heuristic search planner or a SAT planner (Sect. 5) . We finally present a number of examples (Sect. 6) and end with a brief discussion (Sect. 7).

Model and Language

The model from which we will derive finite-state controllers stands for a class of contingent planning problems. For problems in this class, we make three assumptions. First, that actions are *deterministic* and thus all uncertainty results from incomplete information in the initial situation. Second, that actions may have conditional effects but *no preconditions*, and thus are always executable. And third, that *sensing is passive* meaning that the set of observable fluents is fixed and does not depend on the action taken. The contingent planning problem remains challenging even under these assumptions, and in particular, the controllers that we obtain from these models often work when the assumption of determinism is dropped.

Under these assumptions, we adopt a syntax for expressing contingent problems similar to the one used for defining classical problems $P = \langle F, I, A, G\rangle$, where F is the set of fluents, I and G stand for the initial and goal situations, and A is the set of actions, except for the following features.

Actions a in A have all empty preconditions but may have a number of conditional effects $C \rightarrow C'$, where C is a set of fluent literals and C' is either one such set or one such literal. We sometimes write them as $a : C \rightarrow C'$ when we want to indicate the action name.

The initial situation I is given by a set of clauses over the fluents in F so the possible initial states of P are the truth valuations over F that satisfy I.

A set D of *axioms* or *ramification rules* $r \Leftarrow C$ are used to define a set of 'non-primitive fluents' $r \in R$, where C is a set of literals defined over the 'primitive fluents' in F (Thiébaux, Hoffmann, and Nebel 2005). The sets R and F are disjoint, and we allow non-primitive fluents in the body C of conditional effects $C \rightarrow C'$ and goals, but we do not allow them in the heads C'.

A state s is a truth valuation over the primitive fluents that defines the truth value of the non-primitive fluents $r \in R$ through the axioms: r is true in s iff there is an axiom $r \Leftarrow C$ such that C is true in s.

The *observable fluents* or simply the *observables* refer to a set $O \subseteq R$ of non-primitive fluents whose truth value in a given state s is known to the agent. An observation o represents the conjunction of O-literals that are true in a given state. We refer to the observation that corresponds to the state s as $o(s)$ and to the set of all possible observations as O^*. Clearly, the size of O^* is exponential in the number of observable fluents. The observables along with the set of axioms defining them constitute what is called the *sensor model*, that relates the true but hidden state s with the information $o(s)$ available to the agent.

These elements define the class of contingent problems considered, that we call Partially Observable Deterministic Control Problems (PODCPs), abbreviated simply as *control problems:*

Definition 1. *A control problem is a tuple* $P = \langle F, I, A, G, R, O, D\rangle$ *where*

- F *is a set of (primitive) fluents,*
- I *is a set of* F*-clauses representing the initial situation,*
- A *is a set of deterministic actions with conditional effects but no preconditions,*
- G *is a set of literals representing the goal situation,*
- R *is a set of non-primitive fluents,*
- O *is the set of* observable *fluents,* $O \subseteq R$*, and*
- D *is the set of axioms defining the fluents in* R*.*

As an illustration, the control problem expressed by the problem depicted in Fig. 1(a), can be expressed by means of (primitive) fluents $p_i, i = 1, \ldots, 5$, denoting the possible positions of the agent starting from the left, the non-primitive fluent $visitedB$ defined through the axiom $visitedB \Leftarrow p_5$, and initial and goal situations $I = \{oneof(p_1, p_2)\}$ and $G = \{p_1, visitedB\}$. The observables are the non-primitive fluents A and B defined by means of the axioms $A \Leftarrow p_1$ and $B \Leftarrow p_5$, and the actions are *Left* and *Right* with conditional effects $p_i \rightarrow p_{i+1} \wedge \neg p_i$ and $p_{i+1} \rightarrow p_i \wedge \neg p_{i+1}$ respectively, for each $i = 1, \ldots, 4$.

In PODCPs, the initial situation defines a set S_0 of possible initial states s_0, and from any such state, a sequence of actions $\langle a_0, \ldots, a_n\rangle$ defines a unique trajectory $\langle s_0, a_0, \ldots, a_n, s_{n+1}\rangle$ of state-action pairs, and a unique trajectory $\langle o_0, a_0, \ldots, a_n, o_{n+1}\rangle$ of observation-action pairs, that can be obtained by replacing s_i by $o(s_i)$.

Solutions Forms and Finite-State Controllers

PODCPs are a subclass of contingent planning problems and 'qualitative' POMDPs where the uncertainty is repre-

sented by sets of states rather than by probability distributions. Their solutions can be expressed as contingent trees or functions mapping beliefs (sets of states) into actions.

For example, for a version of the control problem shown in Fig. 1 where the agent starts in one of the two leftmost positions but the goal is set to reach the rightmost position, a possible contingent solution can be expressed by the *contingent plan*

$$\langle Right, Right, Right, \textbf{ if } \neg B \textbf{ then } Right \rangle$$

where $Right$ is the action to go right, and B is the observations that corresponds to the rightmost position.

A POMDP solution to the problem, on the other hand, can be given by the partial policy $\pi(b) = Right$, with b ranging over the belief states where the agent is certain not to be at the rightmost position p_5.

An iterative plan of the form

$$\textbf{while } - \text{ or } A \text{ is observed } \textbf{do } Right$$

also solves the problem but no domain-independent planner considers such plans (Sardina et al. 2006).

Finite-state controllers (FSCs) provide yet another solution form. Formally, a finite-state controller for a PODCP $P = \langle F, I, A, G, R, O, D \rangle$ is a tuple $\mathcal{C} = \langle Q, A, O^*, \delta, q_0 \rangle$ with a non-empty and finite set Q of *controller states*, sets A and O^* of actions and observations, a (partial) transition function δ that maps *observation and controller state pairs* $\langle o, q \rangle \in O^* \times Q$ into *action and controller state pairs* $\langle a, q' \rangle \in A \times Q$, and an initial controller state $q_0 \in Q$.[2]

Controller states serve as controller memory allowing the selection of different actions given the same observation. A FSC with a single state represents a memoryless controller.

For example, the FSC $\mathcal{C}$ that solves the problem above contains the single state q_0, and a partial transition function δ that maps the observations A and '$-$' in the state q_0 into the action $Right$ and the same state q_0; namely $\delta(\langle A, q_0 \rangle) = \langle Right, q_0 \rangle$ and $\delta(\langle -, q_0 \rangle) = \langle Right, q_0 \rangle$.

We depict controllers graphically using circles to represent controller states q and directed edges with labels to represent the transition from one controller state into another (or the same) controller state. An edge from q to q' with label o/a states that the transition function of the controller maps the pair $\langle o, q \rangle$ into the pair $\langle a, q' \rangle$. A 2-state controller for the problem is shown in Fig. 1(b). Memoryless controllers are depicted instead as tables mapping observations into actions, as they contain a single state only.

We use rules of the form $\langle o, q \rangle \rightarrow \langle a, q' \rangle$ to express that the transition function for $\mathcal{C}$ is defined on the pair $\langle o, q \rangle$ and maps it into the pair $\langle a, q' \rangle$. It is also convenient to understand a controller $\mathcal{C}$ as a set of such rules, that we express as tuples. Thus we write $t \in \mathcal{C}$ for a tuple $t = \langle o, q, a, q' \rangle$ if $\mathcal{C}$ is defined over the pair $\langle o, q \rangle$ and maps it into the pair $\langle a, q' \rangle$. Note, however, that if $t = \langle o, q, a, q' \rangle$ is a tuple in $\mathcal{C}$, $\mathcal{C}$ cannot contain a tuple $t' \neq t$ with the same $\langle o, q \rangle$ pair, as δ is a transition *function* over such pairs.

[2]Our FSCs are finite automata with output of Mealy-machine type. Moore machines provide an alternative; yet both representations are equally expressive, with Mealy machines typically more succinct (Hopcroft and Ullman 1979).

A finite-state controller $\mathcal{C}$ provides, like contingent trees and POMDP policies, an specification of the action a_{i+1} to do next after a given observation-action sequence $\langle o_0, a_0, \ldots, a_i, o_{i+1} \rangle$. The action to do at time $i = 0$ is $a_0 = a$ if $t = \langle o_0, q_0, a, q' \rangle$ is in $\mathcal{C}$ and $o(s_0) = o_0$, and the controller state that results at time $i = 1$ is q'. Similarly, the action to do at time $i > 0$ is a if o and q are the observation and state at time i, and $t = \langle o, q, a, q' \rangle$ is a tuple in $\mathcal{C}$, and the controller state that results at time $i + 1$ is then q'.

A controller $\mathcal{C}$ and an initial state s_0 determine a unique trajectory $\langle s_0, a_0, \ldots, a_i, s_{i+1}, \ldots \rangle$ of state-action pairs, and a unique trajectory $\langle o_0, a_0, \ldots, a_i, o_{i+1}, \ldots \rangle$ of observation-action pairs. These trajectories *terminate* at time $i + 1$ if the controller state at time $i + 1$ is q and there is no tuple $t = \langle o_{i+1}, q, a, q' \rangle$ in $\mathcal{C}$ defined over the pair $\langle o_{i+1}, q \rangle$. If this is not the case, the observation-action trajectory resulting from s_0 and $\mathcal{C}$ is *infinite*.

We will say that a controller $\mathcal{C}$ *solves* a control problem P if all the state-action pair trajectories that it produces, starting in a possible initial state $s_0 \in I$, reach a goal state. This is a weak form of solution as it does not demand all state trajectories to *terminate* in a goal state. The difference does not matter when the goals are observable but is relevant otherwise. In this paper, we will stick to this weak solution form, and leave stronger forms of solutions for future work. We will show nonetheless that many of the controllers that we obtain are terminating, and thus work, even if the goals are not observable.

Finite-State Controllers as Conformant Plans

The benefits of finite-state controllers over contingent trees and POMDP policies, namely, compact representations that generalize to other problems, come at a prize: while there are well known algorithms for computing solution trees and policies from suitable models, there are no effective methods for computing finite-state controllers. Instead, finite-state controllers are usually written by humans. Recently, heuristic methods for computing finite-state controllers for POMDPs have been considered (Hansen 1998; Meuleau et al. 1999; Poupart and Boutilier 2003) but such methods do not provide any guarantee on the quality or correctness of the result. Below we develop a method for computing finite-state controllers for PODCPs that is formally correct and whose effectiveness will be tested over a number of experiments. For this, the problem of deriving a finite-state controller $\mathcal{C}_N$ with N controller states for a control problem P is mapped into the problem of solving a *conformant problem* P_N that is then tackled using known methods.

Definition 2. *For $P = \langle F, I, A, G, R, O, D \rangle$ and $N \geq 1$, we define the* conformant problem *$P_N = \langle F', I', A', G' \rangle$ as the tuple*

- $F' = F \cup (R \setminus O) \cup O^* \cup Q$,
- $I' = I \cup \{q_0\} \cup \{\neg o \mid o \in O^*\} \cup \{\neg q \mid q \in Q \setminus \{q_0\}\}$,
- $G' = G$,
- $A' = \{b(t) \mid t \in O^* \times Q \times A \times Q\} \cup \{b^\circ\}$

where $Q = \{q_0, \ldots, q_{N-1}\}$. For $t = \langle o, q, a, q' \rangle$, the action

$b(t)$ has conditional effects

$$q,\, o \;\rightarrow\; q',\, \neg o,\, \neg q \qquad \textit{if } q \neq q', \textit{ or}$$
$$q,\, o \;\rightarrow\; \neg o \qquad \textit{if } q = q',$$

and conditional effects $q, o, C \;\rightarrow\; L$ for each conditional effect $a : C \rightarrow L$ in P. The action b°, on the other hand, is a unique (ramification) action *with conditional effects $C \rightarrow r$ for each non-primitive fluent $r \in R \setminus O$ and axiom $r \Leftarrow C$, and conditional effects*

$$C_1,\, C_2,\, \ldots,\, C_n \;\rightarrow\; o$$

for each $o \in O^*$ and each set of axioms $\{p_1 \Leftarrow C_1, p_2 \Leftarrow C_2, \ldots, p_n \Leftarrow C_n\}$ such that p_i, $i = 1, \ldots, n$ are the observables made true by o.

The ramification action b° takes care of updating the fluents in P_N that result from the non-primitive fluents in P: this includes the non-primitive fluents $r \in R \setminus O$ that are non-observable, and the observations $o \in O^*$ that result from the truth valuations over the observable fluents. The number of observations is exponential in the number of observables, and moreover, the number of conditional effects for keeping the value of the observations updated, can be exponential in the number of axioms. Normally, however, the number of observables and the number of axioms for defining them, which constitute the *sensor model* for the problem, can be kept bounded even if the size of the problem grows.

A direct and key consequence of this construction is that the observation fluents $o \in O^*$ are all pairwise mutex in P_N. This information is used below for proving the correctness of the translation and for solving the problem P_N that will yield the controller for P.

The translation of the control problem P into the conformant problem P_N does several things:

1. it translates the observations $o \in O^*$ and the controller states $q \in Q$ into fluents in P_N,
2. it sets the fluent q_0 true in the initial situation, and all other controller states and all observations o to false,
3. it makes the effects of the actions a in P conditional on each observation $o \in O^*$ and each controller state $q \in Q$ so that the 'controller action' $b(t)$ represents the action a, conditioned on $\langle o, q\rangle$, that results in the controller state q' when $t = \langle o, q, a, q'\rangle$, and
4. it captures the effects of actions on the observations and the non-observable non-primitive fluents of P by means of the 'ramification action' b°.

The problem P_N is conformant as the uncertainty in the initial situation I of P is transferred into the uncertainty about the initial situation I' of P_N, and P_N involves no observability at all. In P_N, the observations have been compiled away into the conditional effects of the actions.

Recall that the solution of a conformant problem is an action sequence π that maps the initial belief state into a goal belief state. In the present setting, where actions are deterministic, this can be simplified further, and indeed, an action sequence π solves P_N if π maps each possible initial state of P_N into a goal state.

We establish now the correspondence between the finite-state controllers $\mathcal{C}_N$ that solve a control problem P using N controller states and a particular class of conformant plans π that solve P_N. First of all, it will be convenient to consider two types of conformant plans, sequential and parallel, the former being a special case of the latter.

Definition 3. *A* parallel plan *in P_N is a finite sequence of sets of actions A_i each containing either 'controller actions' $b(t)$ only, or the 'ramification action' b° only, but not both. A* sequential plan *is a parallel plan where all the sets of actions A_i are singletons.*

In addition, we are interested in parallel or sequential plans in P_N that are consistent with a controller. We call these the *functional* plans.

Definition 4. *A sequential or parallel plan is* functional *iff for any pair of actions $b(t)$ and $b(t')$ appearing in the plan for $t = \langle o, q, a, q'\rangle$ and $t' = \langle o, q, a', q''\rangle$, $\langle a, q'\rangle = \langle a', q''\rangle$.*

Functional plans are the ones whose behavior can be explained by a finite-state controller. We show below how to constrain the problem P_N so that the resulting plans are all functional in this sense. Actions $b(t)$ and $b(t')$ are allowed to be done in parallel in functional plans, as at most one of these actions will have an effect on any state s in any belief state over P_N. Indeed, if $t = \langle o, q, a, q'\rangle$, then $b(t)$ has an effect on s only if $s \models o \wedge q$, but then if $b(t') \neq b(t)$ for $t' = \langle o', q'', a', q'''\rangle$ either o' or q'' will be false in s as both, different controller states, and different observations, are mutually exclusive in P_N.

The correspondence between the controllers $\mathcal{C}_N$ that solve the problem P with N controller states and a class of functional plans that solve the conformant problem P_N is established by means of the following theorems:

Theorem 5. *If the finite-state controller $\mathcal{C}_N$ solves the control problem P in at most k steps, the* functional parallel plan *$\pi(\mathcal{C}_N) = \langle A_0, B_0, A_1, \ldots, B_k, A_{k+1}\rangle$, where $A_i = \{b^\circ\}$ and $B_i = \{b(t) \mid t \in \mathcal{C}_N\}$ for $i = 0, \ldots, k+1$, solves the conformant problem P_N.*

Theorem 6. *Let π be a* functional *conformant plan that solves P_N, whether sequential or parallel. Then the controller $\mathcal{C}_N(\pi) = \{t \mid b(t) \in \pi\}$ solves the problem P.*

The proof of the first theorem mimics somewhat the proof in (Hoffmann and Brafman 2005) that the actions in a contingent tree that solves the delete-relaxation of a contingent planning problem with no preconditions, form a parallel conformant plan. The idea is that in a contingent problem with no preconditions and no deletes, all the branches in the plan can be applied in parallel, without interfering with each other, mapping all the possible initial states into goal states. If we consider the contingent tree that results from applying the controller $\mathcal{C}_N$ to P, and the actions a along the branch associated with a possible initial state s, we obtain that those actions map s into a goal state, and that the same occurs if the actions a produced by the tuple $t = \langle o, q, a, q'\rangle$ in $\mathcal{C}_N$ in the branch are replaced by $b(t)$ in the problem P_N. Moreover, in P_N, the actions along all branches can be applied in parallel, thus mapping all the possible initial states into goal states. The reason is that two different actions $b(t)$ and $b(t')$

done at the same time in different branches cannot interfere, very much as actions in the delete relaxation.

Solving the Conformant Problem

In order to solve the conformant problem P_N, we take advantage of a sound and complete translation into classical planning (Palacios and Geffner 2007), which is used either with a sequential suboptimal heuristic-search planner, or with an optimal parallel SAT-based planner. In the first case, we enforce solutions to be *functional* in the sense of Definition 4 by modifying the resulting classical problem P_N with the addition of new fluents. In the second case, we modify the CNF encoding by changing the mutex clauses.

For a conformant planning P_N, Palacios and Geffner define a family of translations. We will make use of the translation K_{S0} that maps P_N into the classical planning problem $K_{S0}(P_N)$. This translation is sound and complete, meaning that the conformant plans for P_N correspond to the classical plans for $K_{S0}(P_N)$, provided that certain dummy actions created in the translation (merges) are dropped. The translation is polynomial in the number of initial states of P and is thus effective when this number is not large.

The Controller from a Sequential Planner

Here we are interested in *functional* plans for the classical problem $K_{S0}(P_N)$, where P_N is the conformant problem associated with the control problem P. We obtain these plans by transforming $K_{S0}(P_N)$ slightly into a new classical planning problem $K'_{S0}(P_N)$ such that the classical plans for $K'_{S0}(P_N)$ are the functional plans for $K_{S0}(P_N)$, and hence, the functional conformant plans for P_N. Recall that a plan π, sequential or parallel, is functional if for every pair $\langle o, q\rangle$ of observation o and controller state q there is at most one operator $b(t)$ in π with $t = \langle o, q, a, q'\rangle$.

Non-functional plans can be excluded by adding two types of fluents $unused(o, q)$ and $mapped(t)$ for all $o \in O^*$, $q \in Q$, and $t = \langle o, q, a, q'\rangle$, and adding actions $map(t)$. The action $map(t)$ for $t = \langle o, q, a, q'\rangle$ has precondition $unused(o, q)$ and effects $mapped(t)$ and $\neg unused(o, q)$, while each literal $mapped(t)$ is added as a precondition of the action $b(t)$. By making the new fluents initially false, the functional plans for $K_{S0}(P_N)$ become simply the plans for $K'_{S0}(P_N)$:

Theorem 7. *Let π be a plan computed for $K'_{S0}(P_N)$ by a* classical planner. *Then, the finite-state controller $\mathcal{C}_N(\pi) = \{t \mid b(t) \in \pi\}$ solves the control problem P.*

The Controller from a SAT Parallel Planner

It is possible and often convenient, however, to compute a functional *parallel* plan that solves $K_{S0}(P_N)$. We have found it useful to use a SAT planner for computing such plans (Kautz and Selman 1996; Hoffmann et al. 2007). In this case, the restrictions ensuring that the resulting parallel plans are functional can be expressed more conveniently by means of additional literals and clauses.

Let $F[K_{S0}(P_N)]$ be the standard CNF encoding of the classical planning problem $K_{S0}(P_N)$ with conditional effects, for some planning horizon, expressing the parallelism captured in Definition 3. That is, in this encoding we only disallow parallelism between 'controller actions' $b(t)$ and the ramification action b°, but add instead other exclusions. We refer to the resulting formula as $F'[K_{S0}(P_N)]$. In this formula, we add first a mutex constraint at all time points between the pairs of literals Ko/s and Ko'/s, for different observations $o, o' \in O^*$ and every possible initial state $s \in S_0$. Likewise, we add a mutex constraint at all time points between pairs of different literals Kq/s and Kq'/s, and do the same for the empty tag in place of s. The literals KL/s are added by the translation K_{S0}, and these constraints over them capture implicit mutex constraints among different observations and among different controller states. Last, we add in $F'[K_{S0}(P_N)]$ the constraints that ensure that the parallel plans encoded in the models of this formula are functional. These constraints make any pair of actions $b(t)$ and $b(t')$ exclusive *at any pair of time points* if $t = \langle q, o, a, q'\rangle$, $t' = \langle q, o, a', q''\rangle$, and either $a \neq a'$ or $q' \neq q''$.

Domain	Inst.	N	Planner	Time	Fig	Works for
Hall-A	1×4	2	Both	0.0	1	$1 \times n$
	4×4	4	Satplan	5,730.5	2	$n \times m$
Hall-R	1×4	1	Both	0.0		$1 \times n$
	4×4	1	Both	0.0	2	$n \times m$
Prize-A	4×4	1	LAMA	0.0	3	$4 \times n, 3 \times n$
Corner-A	4×4	1	Both	0.1	3	$n \times m$
Prize-R	3×3	2	LAMA	0.1		$4 \times n, n \times 4$
	5×5	3	LAMA	2.7	3	$6 \times n, 5 \times n$
Corner-R	2×2	1	Both	0.0		$n \times m$
Prize-T	3×3	1	Both	0.1	3	$n \times m$
Blocks	6	2	Both	0.8	5	n blocks
Visual-M	$(8, 5)$	2	Satplan	1,289.5	5	(n, m)
Gripper	$(3, 5)$	2	Satplan	4,996.1	6	(n, m)

Table 1: Summary of results for some selected controllers: domain, instance, number of states, planner that solved the instance, the time in seconds taken by the fastest planner, figure where controller is shown, and instances for which the controller works.

Theorem 8. *Let M be a model of the CNF formula $F'[K_{S0}(P_N)]$ obtained by a* SAT solver. *Then, the controller $\mathcal{C}_N(M) = \{t \mid b(t)$ is true in M at some time point$\}$ is a finite-state controller that solves the control problem P.*

In this approach, the controller $\mathcal{C}_N$ is obtained by invoking a SAT solver over the CNF formula $F'[K_{S0}(P_N)]$ defined for a plan horizon that is increased from 0 until a model is found. The properties of the CNF encoding of classical plans and the completeness of the above translations ensure that this procedure terminates if there is one such controller.

Experiments

We computed finite-state controllers for control problems over a number of domains: navigation in halls and grids, trash collection, blocks, and gripper. For each problem P and a given number of controller states N, the conformant translation P_N is first obtained, and then the problem $K'_{S0}(P_N)$ is solved by a classical planner or by a SAT-based

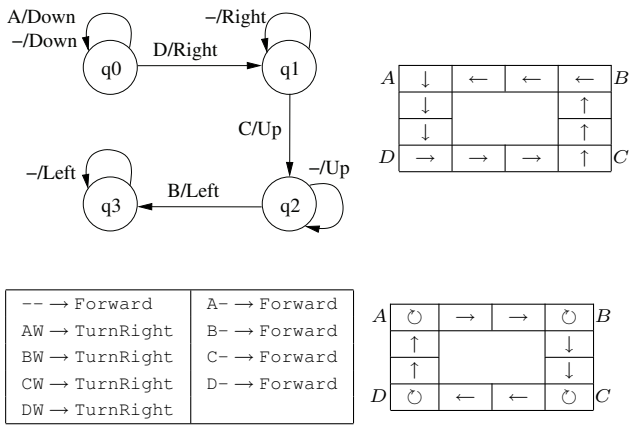

Figure 2: *Top:* 4-state controller obtained for the instance of Hall-A shown on right, with resulting execution. *Bottom:* memoryless controller obtained for the instance of Hall-R shown on right, with resulting execution. In Hall-A agent moves in each of the four directions, in Hall-R it only moves forward but rotates. Both controllers generalize to Halls of any size and work in the presence of non-deterministic effects.

planner using the CNF formula $F'[K_{S0}(P_N)]$.[3] In the first case, LAMA (Richter, Helmert, and Westphal 2008) is used, in the second case, the SAT-based parallel planner reported in (Hoffmann et al. 2007), that handles conditional effects, is used. We tried other planners and solvers, but these ended up being the best choices.[4] Experiments were performed on Xeon processors running at 1.86GHz in a 2Gb RAM Linux machine.

Table 1 provides a summary of most of the experimental results. It details the domains, the instance size used to derive the controller, the number of states in the controller, the planner used, the time taken, the figure where the controller is shown, and the family of instances for which the resulting controller applies.

Halls

The problem in the introduction is the version 1×5 of the Halls domain. The $n \times n$ version, includes four $1 \times n$ halls arranged in a square, and observable marks A, B, C, D at the four corners. Starting in A, the robot has to visit all the marked positions and return to A. As in the problems involving grids below, we consider two representations for the problem: in Hall-A, there are four actions that move the agent in each of the four possible directions, in Hall-R, there is the action to move forward, and two actions for turning 90° left or right. Thus one representation involves absolute coordinates (A), the other coordinates that are relative to the agent (R). In the second representation, the presence of a wall in front of the agent can be detected (W).

[3]The number of controller states N is set by hand in the experiments, although one could also search for N by looking for the minimum integer N for which a solution to the conformant planning problem P_N can be found starting with $N = 1$.

[4]Silvia Richter made an adjustment in LAMA for us, as LAMA could not extract the multivalued variables involved.

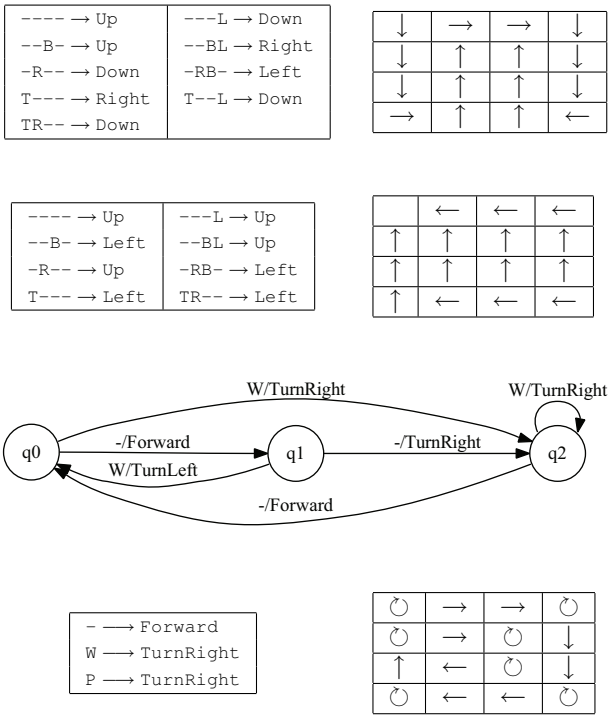

Figure 3: *Top:* memoryless controller obtained for the instance of Prize-A shown on right, with resulting execution. *Second Row:* memoryless controller obtained for the instance of Corner-A shown on right, with resulting execution. *Third Row:* 3-state controller for the 5×5 instance of Prize-R. *Bottom:* Memoryless controller for the 4×4 instance of Prize-T shown on right.

A 4-state controller obtained for the 4×4 instance of Hall-A, and a memoryless controller obtained for a 4×4 instance of Hall-R are shown in Fig. 2. The arrows in the cells show the execution that results when the controller is applied to the initial state where the agent is at A. Both controllers generalize to Halls of any dimension, and work also in the presence of noise in both the initial situation and in the action effects. This generalization is achieved in spite of having inferred the controller from a fixed initial state, and results from the change of representation: sequential plans do not generalize as they do not represent finite-state controllers, unless we associate controller states with time indices. By removing the dependence on time indices, the generalization is achieved. Another way to look at the controllers, is as contingent plans in a language where looping constructs are allowed (Levesque 2005).

Navigation in Grids

The domains Prize and Corner involve rectangular grids. In Prize, the agent has to scan the grid completely for a hidden prize; in Corner, the agent has to get to the top-left corner. The domain options 'A' and 'R' represent the two possible encodings as in Hall, except for the observations: in 'A', the agent observes the presence of walls in the four adjacent cells; in 'R', the agent observes the presence of a wall in front. In the Corner instances, the agent starts in an unknown

FNU → WanderForTrash	AAU → Grab
FAU → WanderForTrash	AFU → WanderForTrashcan
FFU → WanderForTrash	AFU → WanderForTrashcan
NAU → MoveToTrash	FNH → MoveToTrashcan
NNU → MoveToTrash	FAH → Drop
NFU → MoveToTrash	FAH → Drop
ANU → Grab	

Figure 4: Memoryless Controller for Trash Collecting: first position in observation vector refers to how far is the trash (Far, Near, At), the second to how far is the trash can (Far, Near, At), and the third, to whether trash is being held.

location; in Prize, the agent starts at the top-left corner.

Some of the controllers obtained for these domains are shown in Fig. 3. Surprisingly, the memoryless controller obtained for the 4×4 instance of Prize-A (Top), does not generalize to all grids, but only to grids of any 'height'. Probably, the solution for the 5×5 would generalize to any grid size, but we could not solve it (more about this below). On the other hand, the controller obtained for Corner-A for a 4×4 instance (Fig. 3, Second Row), generalizes to any grid size. In this controller, the agent moves along the walls to get to the target corner, and on all other locations, it moves toward one particular wall. This policy is actually very robust, and works in the presence of noisy actions or changes in the initial situation. A 3-state controller for the 5×5 instance of Prize-R is also shown (Fig. 3, Third Row). As in the 4×4 instance of Prize-A, the controller generalizes along one dimension but not along the other. We also tried a variation of Prize-R, called Prize-T, in which the agent is allowed to drop 'pebbles' as it moves around the grid like in the story of Hansen and Gretel. The agent can observe both whether it is in front of a wall ('W') or in front of a cell with a pebble ('P'). Figure 3 (Bottom) shows the memoryless controller obtained from a 3×3 instance that solves the problem Prize-T for any grid size, by making the agent scan the grid by following an spiral.

Trash Collecting

Figure 4 shows a controller for a trash-can collecting robot (Connell 1990; Murphy 2000; Mataric 2007), that can wander for a target object until the object is near, can move to an object if the object is near, can grab an object if located right where the object is, and can drop an object if the object is being held. The task is to move around, wandering for trash, and when a piece of trash is being held, wander for a trash can to drop it. In the encoding, the observable fluents are trash-held, far-from-X, near-X and at-X where X is trash or trashcan. The observations correspond to vectors ABC where A refers to how far is the trash (Far, Near, At), B refers to how far is the trash can (Far, Near, At), and C refers to whether the trash is being held (Held, Unheld). This problem is solved in milliseconds.

Blocks

The first blocksworld problem (Blocks) represents a tower with n blocks, and the goal is to have a green block 'collected'. We encode the domain with three actions, $Unstack$, $Drop$, and $Collect$, that take no arguments and have conditional effects. The first, if block x is clear and on y, unstacks x, clears y, and puts x in the gripper; the second discards the block being held, and the third, collects it. If this block is green, the goal is achieved, otherwise a dead end results. The observable fluents are whether the top block in the tower is green (G), and whether a block is being held (H). Thus the condition that the block being held is green is not observable, and a memoryless controller would not solve the problem. A 2-state controller that generalizes to any number of blocks is shown in Fig. 5 (left).

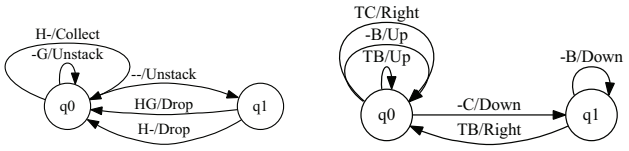

Figure 5: *Left.* Blocks: 2-state controller for collecting a green block in a tower, by observing whether the top block is green and whether an object is being held. *Right.* Visual-M: 2-state controller for placing a visual marker on top of a green block.

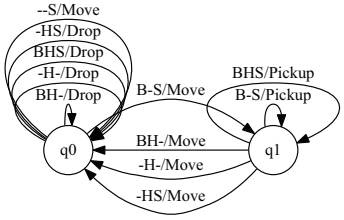

Figure 6: Gripper: 2-state controller for the instance $(3, 5)$ that consists of a robot with 3 grippers and an uncertain number of balls, from 1 to 5. The controller generalizes for problems with an arbitray number of balls and grippers.

The second blocksworld problem (Visual-M) is about placing a visual marker on top of a green block in a blocksworld scene, and is inspired by the use of deictic representations in (Chapman 1989; Ballard et al. 1997). The visual marker, initially at the lower left corner, can be moved along the four compass directions between the cells of the scene, one cell at a time. The observations are whether the cell beneath the marker is empty ('C'), is a non-green block ('B'), or is a green block ('G'), and whether it is on the table ('T') or not ('-'). We obtained controllers for different instances yet none generalized over arbitrary configurations. An instance (n, m) contains m horizontal cells and n blocks in some diposition. By restricting the left/right movements of the visual marker to the level of the table (i.e., when 'T' is observed true), however, we obtained a controller that works for *any* number of blocks and cells. It is shown in the right side of Fig. 5, and it basically searches for a tower with a green block from left to right, going all the way up to the top in each tower, then going all the way down to the table, and iterating in this manner until the visual marker appears on top of a block that is green.

Gripper

This is a version of the familiar Gripper domain, where a robot with a number of grippers must carry balls from room

		LAMA			SATPLAN		
Problem	N	Largest	Len	Time	Largest	Len/MkSp	Time
Hall-A	4	3×2	36	1.3	4×4	24/24	5,730.5
Hall-R	1	4×4	41	0.0	4×4	32/32	68.1
Prize-A	1	4×4	41	0.0	4×4	33/32	253.0
Corner-A	1	4×4	37	0.4	4×4	25/14	1.3
Prize-R	4	4×4	198	0.3	3×2	20/20	106.1
Corner-R	1	5×5	22	1.6	5×5	21/20	6.2
Prize-T	1	5×5	71	0.3	5×4	56/57	4,586.0
Blocks	2	20	185	34.8	13	68/58	3,964.8
Visual-M	2	$(7, 5)$	26	0.3	$(8, 5)$	39/39	1,289.5
Gripper	2	$(2, 2)$	–	–	$(3, 5)$	41/29	4,996.1

Table 2: Results that illustrate the scalability of LAMA and SATplan. LAMA fails on memory, and SATPLAN on either memory or time.

B to room A. The robot can move between rooms, and it can pick up and drop balls using its grippers. The observations consist of whether there are balls left in B ('B'), whether there is space left in the grippers ('S'), and whether the robot is holding some ball ('H'). The robot cannot directly observe its location yet it is initially at room A with certainty. The instance (n, m) refers to a problem with n grippers and an uncertain number of balls in room B, that could range from 1 to m. Fig. 6 shows the controller obtained for the instance $(3, 5)$ in 4,996 seconds by Satplan which also works for problems (n, m) for arbitrary n and m. The robot goes first to B, and picks up balls until no space is left in the grippers, then it moves to A, where it drops all the balls, one by one, repeating the cycle, until no balls are left in B and the robot is not holding any balls. In this case, the goal is observable and is true when neither B nor H are true.

Discussion

In this work, we have developed a method for deriving finite-state controllers automatically from models in a class of contingent problems. We have converted the problem of deriving a controller into a conformant planning problem that is solved using a recently proposed transformation and classical planners. The controllers derived in this way are often general in the sense that they do not solve the original problem only, but many variations too, including changes in the size of the problem or in the uncertainty of the initial situation and action effects. It is not clear, however, from looking at an instance whether the resulting controller will generalize along some, all, or none of these dimensions. This remains an interesting open issue. The other challenge is scalability. The classical problems that we obtain are not easy for the current classical planners. Table 2 illustrates the current capabilities in terms of the instances that the planners LAMA and SATplan can solve. Finally, while the resulting controllers solve the given control problem, the notion of solution used in the paper is rather weak: it requires the goal to be reached but it does not require the controller to halt the execution. This, however, is necessary when the achievement of the goal is not observable. The approach, however, can be generalized to yield such terminating controllers and we will work out the details elsewhere.

Acknowledgements We thank Silvia Richter and Joerg Hoffmann for help with LAMA and SATplan, and Malte Helmert and Patrik Haslum for trying their planners on some instances too. H. Geffner is partially supported by grant TIN2006-15387-C03-03 from MEC/Spain.

References

Amato, C.; Bernstein, D.; and Zilberstein, S. 2007. Optimizing memory-bounded controllers for decentralized pomdps. In *Proc. UAI*.

Ballard, D.; Hayhoe, M.; Pook, P.; and Rao, R. 1997. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences* 20(04):723–742.

Buckland, M. 2004. *Programming Game AI by Example*. Wordware Publishing, Inc.

Chapman, D. 1989. Penguins can make cake. *AI magazine* 10(4):45–50.

Connell, J. H. 1990. *Minimalist Mobile Robotics*. Morgan Kaufmann.

Hansen, E. 1998. Solving POMDPs by searching in policy space. In *Proc. UAI*, 211–219.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, 71–80.

Hoffmann, J.; Gomes, C.; Selman, B.; and Kautz, H. A. 2007. SAT encodings of state-space reachability problems in numeric domains. In *Proc. IJCAI*, 1918–1923.

Hopcroft, J., and Ullman, J. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Kaelbling, L. P.; Littman, M.; and Cassandra, A. R. 1999. Planning and acting in partially observable stochastic domains. *Artif. Intell.* 101:99–134.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI*, 1194–1201.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proc. AAAI*, 1139–1146.

Levesque, H. 2005. Planning with loops. In *Proc. IJCAI*, 509–515.

Littman, M. L. 1994. Memoryless policies: Theoretical limitations and practical results. In Cliff, D., ed., *From Animals to Animats 3*. MIT Press.

Mataric, M. J. 2007. *The Robotics Primer*. MIT Press.

Meuleau, N.; Peshkin, L.; Kim, K.; and Kaelbling, L. P. 1999. Learning finite-state controllers for partially observable environments. In *Proc. UAI*, 427–436.

Murphy, R. R. 2000. *An Introduction to AI Robotics*. MIT Press.

Palacios, H., and Geffner, H. 2007. From conformant into classical planning: Efficient translations that may be efficient too. In *Proc. ICAPS*, 264–271.

Poupart, P., and Boutilier, C. 2003. Bounded finite state controllers. In *Proc. NIPS*, 823–830.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI*, 975–982.

Sardina, S.; Giacomo, G. D.; Lesperance, Y.; and Levesque, H. 2006. On the limits of planning over belief states. In *Proc. KR*, 463–471.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.* 168(1–2):38–69.