A Correctness Result for Synthesizing Plans With Loops in Stochastic Domains

Citation for published version:
Treszkai, L & Belle, V 2020, 'A Correctness Result for Synthesizing Plans With Loops in Stochastic Domains', *International Journal of Approximate Reasoning*, vol. 119, pp. 92-107. https://doi.org/10.1016/j.ijar.2019.12.005

Digital Object Identifier (DOI):
https://doi.org/10.1016/j.ijar.2019.12.005

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
International Journal of Approximate Reasoning

A Correctness Result for Synthesizing Plans With Loops in Stochastic Domains

Laszlo Treszkai[a], Vaishak Belle[a,b,*]

[a] *School of Informatics, University of Edinburgh, Edinburgh, UK.*
[b] *Alan Turing Institute, London, UK.*

Abstract

Finite-state controllers (FSCs), such as plans with loops, are powerful and compact representations of action selection widely used in robotics, video games and logistics. There has been steady progress on synthesizing FSCs in deterministic environments, but the algorithmic machinery needed for lifting such techniques to stochastic environments is not yet fully understood. While the derivation of FSCs has received some attention in the context of discounted expected reward measures, they are often solved approximately and/or without correctness guarantees. In essence, that makes it difficult to analyze fundamental concerns such as: do all paths terminate, and do the majority of paths reach a goal state?

In this paper, we present new theoretical results on a generic technique for synthesizing FSCs in stochastic environments, allowing for highly granular specifications on termination and goal satisfaction.

Keywords: Plan and program synthesis, Stochastic domains, Loops in plans and programs, Stochastic algorithms, Planning in robotics

1. Introduction

Finite-state controllers (FSCs), such as plans with loops, are powerful and compact representations of action selection widely used in robotics, video games and logistics. In AI, FSCs are much sought after for automated planning paradigms such as *generalized planning*, as in Figure 1, where one attempts to synthesize a controller that works in multiple initial states. Such controllers are usually hand-written by domain experts, which is problematic when expert knowledge is either unavailable or unreliable. To that end, the automated synthesis of FSCs has received considerable attention in recent years, e.g., [19, 7, 32, 30, 12, 33, 16]. Of course, FSCs synthesis is closely related to program synthesis [19], and FSCs are frequently seen as program-like plans [20], and recent synthesis literature involves an exciting exchange of technical insights between the two fields [32]; representative examples include the use of program synthesis to infer high-level action types [31], and the use of partial order planning for imperative program synthesis [15].

Naturally, from an algorithmic perspective, the two most immediate questions are: in which sense are controllers *correct*, and how do we *synthesize* controllers that are *provably correct*? In classical deterministic settings, plan paths can only be extended uniquely, so it suffices to show that there is a terminating path that reaches the goal state. Ideally, then, what we seek is a procedure that is both *sound* (i.e., all synthesized controllers are correct) and *complete* (i.e., if there is a plan, then the procedure finds it).

The compact nature of FSCs makes them particularly attractive for mobile robots [24], among other domains where there is inherent stochasticity and actions are noisy. To a first approximation, in the presence of non-probabilistic non-determinism, it is common practice to make assumptions, such as disallowing repeated configurations of (state, action) pairs [27, 28]. However, in stochastic environments, that is almost always an unreasonable assumption. Consider a robot attempting to grip an object: the first and second attempt may fail, but perhaps the third succeeds. Structurally, the first two failures are identical: while a domain expert might find a way to distinguish the two states for the planner,

[*]Corresponding author. Vaishak Belle was supported by a Royal Society University Research Fellowship.
Email addresses: `laszlo.treszkai@gmail.com` (Laszlo Treszkai), `vaishak@ed.ac.uk` (Vaishak Belle)

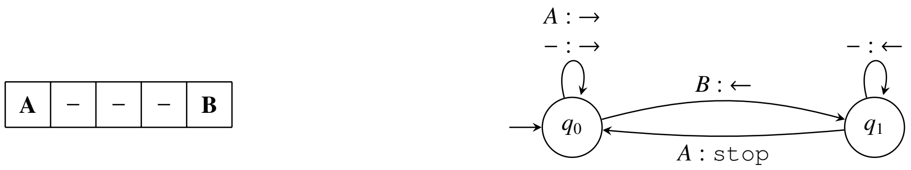

Figure 1: *Left*: A planning problem where the agent is initially in cell A, and the goal is to visit cell B and go back to cell A. At each state, possible observations are A, B, or $-$. *Right*: A correct finite state controller for this problem. The circles are controller states, and an edge $q \xrightarrow{o:a} q'$ means "do a when the observation is o in controller state q, and then switch to controller state q'." The reader may observe that this controller works for any number of states between A and B in the domain, e.g., for $(A, -, B)$ as well as $(A, -, -, -, -, B)$.

from a robustness viewpoint, of course, it is more desirable when algorithms operate without such assumptions. In this regard, the algorithmic machinery needed for lifting FSC synthesis techniques to stochastic environments is not yet fully understood.

More generally, we identify the following desiderata:

D1. The planner should cope with plan paths that do not terminate in a goal state;

D2. The planner should correctly account for how a looping history affects goal probabilities, distinguishing loops that never terminate and loops that can be extended into goal histories; and

D3. The planner should recognize when a combination of loops never terminates, even if the loops by themselves appear to be possibly terminating.

Implicit in these desiderata is the idea that the planner should leverage the likelihood of action outcomes, because *(a)* these likelihoods are informative about which action outcomes are more likely than others, and *(b)* in the presence of repeating configurations, probabilities allow for a natural tapering of the likelihood of paths.

In this paper, we present new theoretical results on a generic technique for synthesizing FSCs in stochastic environments, by means of a probabilistic extension of AND-OR search. We provide a careful analysis of how to maintain upper and lower bounds of the likelihoods of paths, so that one can naturally deal with tapering probabilities, arising from repeating configurations. In particular, it allows us to plan for highly granular specifications, such as: generate a FSC under the requirement that >80% of the paths terminate, and >60% of the paths reach the goal state. Most significantly, we prove that our algorithm is both *sound* and *complete*.

2. Problem formalisation

Our contributions do not depend on the details of the formal language (e.g., [14]), and so we consider an abstract framework [4].

Definition 1. An *environment* $\mathcal{E}$ is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, O, \Delta, \Omega \rangle$, whose elements are the following: $\mathcal{S}$, $\mathcal{A}$, O are finite sets of states, actions, and observations; $\Delta : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is a stochastic state transition function, where $\Pi(\mathcal{S})$ denotes the set of probability distributions over $\mathcal{S}$; and $\Omega : \mathcal{S} \to O$ is an observation function.

A planning problem is defined as an environment, an initial state, and a set of goal states:

Definition 2. A *planning problem* $\mathcal{P}$ is a triple $\langle \mathcal{E}, s_0, \mathcal{G} \rangle$, where $\mathcal{E}$ is an environment with state space $\mathcal{S}$, $s_0 \in \mathcal{S}$ is the *initial state*, and $\mathcal{G} \subset \mathcal{S}$ is the *set of goal states*.

We represent loopy plans as follows [25]:

Definition 3. A *finite state controller* (FSC) C is defined by a tuple $\langle Q, q_0, O, \mathcal{A}, \gamma, \delta \rangle$, where: $Q = \{q_0, q_1, \dots, q_{N-1}\}$ is a finite set of *controller states*; $q_0 \in Q$ is the *initial state* of the controller; O & $\mathcal{A}$ are the sets of possible observations & actions; $\gamma : Q \times O \to (\mathcal{A} \cup \{\texttt{stop}\})$ is a partial function called the *labeling function*; $\delta : Q \times O \to Q$ is a partial function called the *transition function*.

An FSC forms part of a system $\langle \mathcal{E}, C\rangle$, usually for a planning problem $\langle \mathcal{E}, s_0, \mathcal{G}\rangle$. Initially, at step $t = 0$, the environment is in state $s^{(0)} = s_0$, and the FSC C is in controller state $q^{(0)} = q_0$, where the superscript denotes the current step t. The controller makes an observation $o^{(0)} = \Omega(s_0)$, executes action $a^{(0)} = \gamma(q^{(0)}, o^{(0)})$, and transitions to controller state $q^{(1)} = \delta(q^{(0)}, o^{(0)})$. The environment transitions to state $s^{(1)} \sim \Delta(s^{(1)}|s^{(0)}, a^{(0)})$. This process is repeated until the special action `stop` is executed, when the state→observation→action→next-state cycle stops.

We call a pair of controller and environment state $\langle q, s\rangle$ a *combined state* of the system.

Notation. We denote the value of any $x \in \{s, q, o, a, p\}$ at step t during the execution of a system by $x^{(t)}$. (The meaning of $p^{(t)}$ is introduced in Definition 4.) A sequence is written as $\langle x^{(t)}\rangle_{t=i}^{j} := \langle x^{(i)}, x^{(i+1)}, \ldots, x^{(j)}\rangle$. The subsequence of $h \equiv \langle x^{(t)}\rangle_{0\le t\le n}$ between indices i and j is denoted by $h^{(i:j)} := \langle x^{(i)}, x^{(i+1)}, \ldots, x^{(j)}\rangle$, and the subsequence from an index until the beginning or end of the sequence are denoted by $h^{(i:)} := h^{(i:n)}$ and $h^{(:j)} := h^{(0:j)}$, respectively. $end(h)$ refers to the last element of h. The concatenation of two *compatible* sequences is denoted by the $\cdot$ operator, e.g., $h^{(i:j)} \cdot h^{(j:k)} := h^{(i:k)}$. (Sequences h_1 and h_2 are compatible if $end(h_1) = h_2^{(0)}$.)

A *history* of a system from a given combined state is one possible sequence of states that it follows, not necessarily until termination.

Definition 4. Let $C = \langle Q, q_0, O, \mathcal{A}, \gamma, \delta\rangle$ be a finite state controller, and $\mathcal{E}$ an environment. For any integer $T \ge 0$, a *history* $h = \langle\, \langle q^{(t)}, s^{(t)}\rangle\, \rangle_{t=0}^{T} \in (Q \times \mathcal{S})^T$ of a system $\langle \mathcal{E}, C\rangle$ from the combined state $\langle q^{(0)}, s^{(0)}\rangle$ is a finite sequence of combined states such that $p^{(t+1)} = \Delta(s^{(t+1)} \mid s^{(t)}, a^{(t)}) > 0$, where $a^{(t)} := \gamma(q^{(t)}, \Omega(s^{(t)}))$, and $q^{(t+1)} = \delta(q^{(t)}, \Omega(s^{(t)}))$, for each $0 \le t < T$. A history $h^{(0:T)}$ for a planning problem $\langle \mathcal{E}, s_0, \mathcal{G}\rangle$ is *terminating* if $a^{(T)} =$ `stop`. A terminating history for a planning problem $\langle \mathcal{E}, s_0, \mathcal{G}\rangle$ is a *goal history* if C terminates in a goal state: $a^{(T)} =$ `stop` and $s^{(T)} \in \mathcal{G}$. Unless otherwise noted, the first element of a history is $\langle q_0, s_0\rangle$.

Although the action $a^{(t)}$ is not included explicitly in the history, it can be obtained from $\langle q^{(t)}, s^{(t)}\rangle$.

The *likelihood* ℓ of a history h is the probability that at each step t, the environment responds to the controller's action $a^{(t)}$ with the next state $s^{(t+1)}$. The likelihood can be defined inductively based on the length of the history:

$$\begin{aligned}
&\ell(h^{(:0)}) := 1\\
&\ell(h^{(:t+1)}) := \ell(h^{(:t)}) \cdot \Delta(s^{(t+1)} \mid s^{(t)}, a^{(t)}).
\end{aligned}$$

The most immediate question here is this: in which sense would we say that a controller is *adequate* for a planning problem? In the absence of noise/nondeterminism, it is easy to show that the transition of combined states is deterministic; put differently, histories can be extended uniquely [10, 12]. So it suffices to argue that there is a *terminating history* and that it is a *goal history*. Of course, in the presence of nondeterminism, the extension of histories is no longer unique (because of nondeterministic action outcomes), and in the presence of probabilities, it is also useful to consider the likelihood of these extensions. We follow [2], where the notion of correctness from [14, 12] is extended for noise, and define:

Definition 5. The *total likelihood of termination* of the system $\langle \mathcal{E}, C\rangle$ on a planning problem $\mathcal{P}$ is denoted by **LTER**:

$$\mathbf{LTER} := \sum_{\{h \,\mid\, h \text{ is a terminating history}\}} \ell(h) \tag{1}$$

Analogously, for goals, we define:

Definition 6. The *total likelihood of goal termination* of the system $\langle \mathcal{E}, C\rangle$ on a planning problem $\mathcal{P}$ is denoted by **LGT**:

$$\mathbf{LGT} := \sum_{\{h \,\mid\, h \text{ is a goal history}\}} \ell(h) \tag{2}$$

Finally, we state the search problems we want to solve.

Problem 1. Given a planning problem $\mathcal{P}$, an integer N, $LGT^\star \in (0, 1)$, find a finite-state controller with at most N states such that $\mathbf{LGT} \ge LGT^\star$ for $\mathcal{P}$.

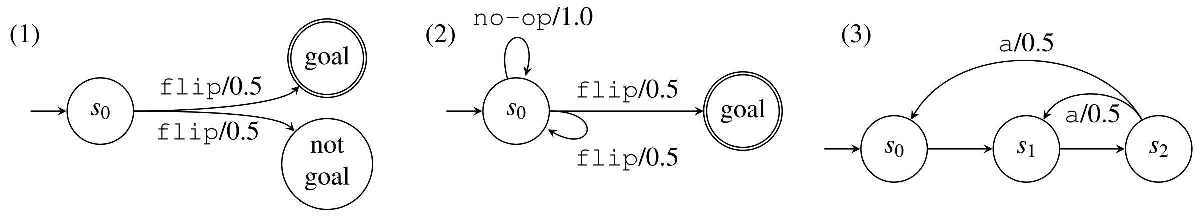

Figure 2: Test environments for probabilistic planning with loops. (1) No controller with **LGT** = 1 exists. (2) Difference between decaying and non-decaying loops. (3) Decaying loops whose combination never terminates.

A more fine-grained version is where a minimum bound on **LTER** is also possible, defined below:

Problem 2. Given a planning problem $\mathcal{P}$, an integer N, $LTER^\star \in (0,1)$, $LGT^\star \in (0,1)$, find a finite-state controller with at most N states that is **LTER** $\geq LTER^\star$ and **LGT** $\geq LGT^\star$ for $\mathcal{P}$.

We restrict our attention to solutions for Problem 1 for the most part, and turn to Problem 2 in the penultimate technical section.

3. Synthesizing classical controllers

Existing strategies for synthesizing FSCs include the compilation of generalized planning problems to classical ones [7], and the generalization of a sequential plan by abstraction [32]. A review of generalized planning is given by Celorrio et al. [8].

From the perspective of an algorithmic schema, the generic technique of Hu and De Giacomo [12] is perhaps the simplest to analyze, based on AND-OR search. Here, an environment virtually identical to ours is assumed, and the transition relation is also nondeterministic (but non-probabilistic) via a state transition *relation* $\Delta \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. The pseudocode for the planner is in Algorithm 1. Initially, the algorithm starts with the empty controller C_ε, at the initial controller state $q^{(0)}$, with next states in $\mathcal{S}_0$, the initial states of a generalized planning problem. The AND-STEP function enumerates the outcomes of an action from a given combined state and history, and calls OR-STEP to synthesize a controller that is correct for every outcome. The OR-STEP function enumerates the extensions of a controller for the current controller state and observation, and thus selects a next action for the current observation, and then calls AND-STEP to test for correctness recursively on the outcomes of the chosen action.[1] The algorithm is essentially a blind search in controller space, reverting to the last non-deterministic choice point when a branch fails. The search space is trimmed in two ways. First, when a controller C is found to be not correct, every extension of C is dropped as well. Second, if $C \prec C'$ (meaning that every controller transition defined by C is the same in C'), then the histories of C that were already explored are not tested again for C'. Most significantly, this exhaustive search results in the algorithm being *sound* and *complete* [12].

4. Problems with loops in a noisy environment

We identified three desiderata in the introduction, which we justify below. For **D1** (handling termination out of a goal state), it is clear that there exist planning problems where even the optimal controller might not terminate on every run or end up in a goal state on every terminating run. (Consider a problem with an unavoidable dead end state, for example one where one outcome of a `coin_flip` action ends up in the goal, another outcome results in a dead end state. See Fig. 2.1.)

D2 (handling looping histories) is the result of assigning probabilities to the different outcomes of an action. If a history repeats a combined state at steps n and m, and $\ell(h^{(0:n)}) = \ell(h^{(0:m)})$, then the system will repeat the loop $h^{(n:m)}$ indefinitely, and never terminate. On the other hand, if $\ell(h^{(0:n)}) > \ell(h^{(0:m)})$, and there exists a history h' from $h^{(n)}$ such

[1]We provide the pseudocode of the algorithm to easily contrast it with our algorithm, but some details from [12] are omitted for the sake of exposition.

Algorithm 1 The AND-OR search algorithm for bounded finite state controllers [12, Fig. 4].

```
Require: $\overline{\mathcal{P}} = \langle \mathcal{E}, S_0, \mathcal{G} \rangle$, a generalized planning problem;
         $N$, a bound on the number of controller states.
1:  function ANDOR-SYNTH($\overline{\mathcal{P}}, N$)
2:      return AND-STEP$_{\overline{\mathcal{P}},N}(C_\varepsilon, 0, S_0, \langle\rangle)$
3:  end function
4:  function AND-STEP$_{\overline{\mathcal{P}},N}(C, q, S', h)$
5:      for all $s' \in S'$ do
6:          $C \leftarrow$ OR-STEP$_{\overline{\mathcal{P}},N}(C, q, s', h)$
7:      end for
8:      return $C$
9:  end function
10: function OR-STEP$_{\overline{\mathcal{P}},N}(C, q, s, h)$
11:     if $s \in \mathcal{G}$ then
12:         return $C$
13:     else if $\langle q, s \rangle \in h$ then
14:         fail
15:     else if $q \xrightarrow{\Omega(s)/a} q' \in C$ for some $q', a$ then
16:         $S' \leftarrow \{s' \mid \langle s, a, s' \rangle \in \Delta\}$
17:         return AND-STEP$_{\overline{\mathcal{P}},N}(C', q', S', h \cdot \langle q, s \rangle)$
18:     else
19:         non-det. branch $a \in \mathcal{A}$ and $q' \in \{0, \ldots, N-1\}$
20:         $C' \leftarrow C \cup \{q \xrightarrow{\Omega(s)/a} q'\}$
21:         $S' \leftarrow \{s' \mid \langle s, a, s' \rangle \in \Delta\}$
22:         return AND-STEP$_{\overline{\mathcal{P}},N}(C', q', S', h \cdot \langle q, s \rangle)$
23:     end if
24: end function
```

that $h^{(0:n)} \cdot h'$ terminates in $\mathcal{G}$, then $h^{(0:m)} \cdot h'$ will also terminate in $\mathcal{G}$. (See Fig. 2.2.) This means that in a stochastic environment, tracking the likelihood of histories is essential.

For **D3** (handling non-terminating loop combinations), in some environments no looping history has the property that $\ell(h^{(0:n)}) = \ell(h^{(0:m)})$, and yet the system has no terminating runs. Fig. 2.3 illustrates a simple example, where executing action $\mathtt{a}$ in s_2 brings the environment either to s_0 or s_1, and from s_0 the only possible action leads deterministically to s_1 and from there back to s_2.

It is not possible to analyze every controller synthesis framework in the literature to verify its adherence to these desiderata, but, in the very least, the case of [12] is illustrative. Their procedure is correct for the dynamic environment in which they operate, but as can be inferred from the above examples, it easily follows that the procedure fails to meet the first two of the desiderata in stochastic environments.

Theorem 1. *The algorithm by Hu and De Giacomo [12] returns with failure if every controller for the planning problem has at least one history that cannot be extended into a goal history.*

Theorem 2. *The algorithm by Hu and De Giacomo [12] returns with failure if every controller for the problem has at least one looping history.*

Theorem 2 is true because line 13–14 of Algorithm 1 disallows plans that result in looping histories. As the algorithm explores the whole AND-OR tree of the system, such plans need to be discarded, otherwise the algorithm would be stuck exploring the infinite subtree resulting from the loop. Without assigning probabilities to histories, the algorithm cannot tell how important any given loop would be.

More significantly, it is not possible to specify likelihood-based correctness criteria, which becomes essential for handling domains where actions fail, for example.

5. Algorithm for loopy planning

We propose a search algorithm that provably meets all three of the desiderata. Similar to the algorithm of Hu and De Giacomo [12], it also instantiates an AND-OR search in that it simulates the runs of a system, enumerates the possible controller extensions whenever no action is yet defined for the current combined state, and when an action has multiple outcomes, it does a depth-first search on the next states recursively. However, it fixes the shortcomings stated in the previous section: instead of only allowing controllers that are correct on every run, it synthesizes controllers whose correctness likelihood exceeds some likelihood given as input to the algorithm; and it is capable of handling looping histories. As it is a probabilistic variant of the AND-OR search, we name it PANDOR.

5.1. Allowing less than perfect controllers

The basic idea behind our planner is that it maintains an upper and lower bound for the **LGT** of the current controller, based on the histories simulated thus far. Whenever a failing run is encountered, the upper bound is decreased by the likelihood of this run; similarly, a goal run increases the lower bound on **LGT**. When the lower bound exceeds the desired correctness likelihood (hereafter denoted by $LGT^\star$), the current controller is guaranteed to be "good enough", and the algorithm returns with success. When the upper bound is lower than $LGT^\star$, none of the extensions of the controller is sufficiently good, and we revert the program state to the point of the last non-deterministic choice point. In the simplest variant of PANDOR, any run with repeated combined states is counted as a failed run, thus it meets **D1** but not **D2**. (It handles out-of-goal termination but is too conservative with looping histories.) This property leads to an underestimation of the lower and upper bounds on **LGT**, making the search sound but incomplete.

While the planner of [12] declared a controller and all of its extensions insufficiently good when it had a single failing run, we relax this condition. Now a controller is insufficiently good when the total likelihood of all of its failing runs exceeds $1 - LGT^\star$; this results in the same behavior as that of its predecessor when $LGT^\star = 1$.

Algorithm 2 The PANDOR algorithm, which synthesizes finite state controllers with looping histories.

```
Require: $\mathcal{P} = \langle \mathcal{E}, s_0, \mathcal{G} \rangle$, a planning problem;
         $N$, a bound on the number of controller states;
         $LGT^\star$: the desired minimum LGT.
 1: function PANDOR-SYNTH($\mathcal{P}, N$)
 2:     (global) $\alpha \leftarrow \langle\rangle$
 3:     return AND-STEP$_{\mathcal{P},N}(C_\varepsilon, 0, \{\langle s_0, 1.0\rangle\}, \langle\rangle)$
 4: end function
 5: function AND-STEP$_{\mathcal{P},N}(C, q, SP', h^{(0:n)})$
 6:     $\alpha_x^{(n+1)} \leftarrow 0, \quad$ for $x \in \{\text{goal}, \text{fail}, \text{noter}\}$
 7:     $\alpha_{\text{loop}}^{(n+1,:)} \leftarrow 0; \quad \alpha_{\text{loop}}^{(:,n+1)} \leftarrow 0$
 8:     for all $\langle s', p'\rangle \in SP'$ do
 9:         $C \leftarrow$ OR-STEP$_{\mathcal{P},N}(C, q, s', p', h)$
10:         $\lambda \leftarrow$ CALCLAMBDA$(h, \alpha)$
11:         if $\lambda_{\text{goal}} \geq LGT^\star$ then
12:             return $C$
13:         else if $1 - \lambda_{\text{fail}} - \lambda_{\text{noter}} < LGT^\star$ then
14:             fail this non-deterministic branch
15:         end if
16:     end for
17:     $\alpha \leftarrow$ CUMULATEALPHA$(h, \alpha)$
18:     return $C$
19: end function
20: function OR-STEP$_{\mathcal{P},N}(C, q, s, p, h^{(0:n)})$
21:     if $s = s_{\text{win}}$ then
22:         $\alpha_{\text{goal}}^{(n+1)} \leftarrow \alpha_{\text{goal}}^{(n+1)} + p$
23:         return $C$
24:     else if $s = s_{\text{fail}}$ then
25:         $\alpha_{\text{fail}}^{(n+1)} \leftarrow \alpha_{\text{fail}}^{(n+1)} + p$
26:         return $C$
27:     else if $q^{(k)} = q$ and $s^{(k)} = s$ for some $k$ then
28:         if $p = 1$ and $p^{(i)} = 1$ for all $k + 1 \leq i \leq n$ then
29:             $\alpha_{\text{noter}}^{(n+1)} \leftarrow 1$
30:             return $C$
31:         else
32:             $\alpha_{\text{loop}}^{(k,n)} \leftarrow \alpha_{\text{loop}}^{(k,n)} + p$
33:             return $C$
34:         end if
35:     else if $q \xrightarrow{\Omega(s)/a} q' \in C$ for some $a, q'$ then
36:         $SP' \leftarrow$ NEXTSTATES$_{\mathcal{P}}(s, a)$
37:         return AND-STEP$_{\mathcal{P},N}(C', q', SP', h \cdot \langle q, s, p\rangle)$
38:     else
39:         non-det. branch $a \in \mathcal{A}$ and $q' \in \{0, \ldots, N-1\}$:
40:         $C' \leftarrow C \cup \{q \xrightarrow{\Omega(s)/a} q'\}$
41:         $SP' \leftarrow$ NEXTSTATES$_{\mathcal{P}}(s, a)$
42:         return AND-STEP$_{\mathcal{P},N}(C', q', SP', h \cdot \langle q, s, p\rangle)$
43:     end if
44: end function
```

Algorithm 3 Helper functions used by PANDOR.

```
45: function CUMULATEALPHA($h^{(0:n)}, \alpha$)
46:     for all $x \in \{\text{goal}, \text{fail}, \text{noter}\}$ do
47:         $\alpha_x^{(n)} \leftarrow \alpha_x^{(n)} + p^{(n)}\alpha_x^{(n+1)}/(1 - \alpha_{\text{loop}}^{(n,n)})$
48:     end for
49:     for $k \leftarrow 0 \ldots n-1$ do
50:         $\alpha_{\text{loop}}^{(k,n-1)} \leftarrow \alpha_{\text{loop}}^{(k,n-1)} + p^{(n)}\alpha_{\text{loop}}^{(k,n)}/(1 - \alpha_{\text{loop}}^{(n,n)})$
51:         $\alpha_{\text{loop}}^{(k,n)} \leftarrow 0$
52:     end for
53:     $\alpha_{\text{loop}}^{(n,n)} \leftarrow 0$
54:     return $\alpha$
55: end function

56: function NEXTSTATES$_{\mathcal{P}}(s, a)$
57:     if $a = \texttt{stop}$ and $s \in \mathcal{G}$ then
58:         return $\{\langle s_{\text{win}}, 1.0\rangle\}$
59:     else if $a = \texttt{stop}$ and $s \notin \mathcal{G}$ then
60:         return $\{\langle s_{\text{fail}}, 1.0\rangle\}$
61:     else
62:         return $\{\langle s', p'\rangle \mid \Delta(s' \mid s, a) = p' > 0\}$
63:     end if
64: end function

65: function CALCLAMBDA($h^{(0:n)}, \alpha$)
66:     $\lambda_x \leftarrow \alpha_x^{(n+1)}$,    for $x \in \{\text{goal}, \text{fail}, \text{noter}\}$
67:     $\lambda_{\text{loop}}^{(0:n)} \leftarrow [0, 0, \ldots, 0]$
68:     for $k \leftarrow n \ldots 0$ do
69:         $\lambda_{\text{loop}}^{(k)} \leftarrow \alpha_{\text{loop}}^{(k,n)}$
70:         for $m \leftarrow n-1 \ldots k$ do
71:             $\lambda_{\text{loop}}^{(k)} \leftarrow \alpha_{\text{loop}}^{(k,m)} + p^{(m+1)}/(1 - \lambda_{\text{loop}}^{(m+1)}) \cdot \lambda_{\text{loop}}^{(k)}$
72:         end for
73:         if $\lambda_{\text{loop}}^{(k)} + \lambda_{\text{noter}} \approx 1$ then
74:             $\lambda_{\text{loop}}^{(k)} \leftarrow 0$
75:             $\alpha_{\text{loop}}^{(k:,k:)} \leftarrow 0$
76:             $\alpha_{\text{noter}}^{(k)} \leftarrow \alpha_{\text{noter}}^{(k)} + p^{(k)}$
77:             for all $x \in \{\text{goal}, \text{fail}, \text{noter}\}$ do
78:                 $\lambda_x \leftarrow \alpha_x^{(k)}$
79:             end for
80:         else
81:             for all $x \in \{\text{goal}, \text{fail}, \text{noter}\}$ do
82:                 $\lambda_x \leftarrow \alpha_x^{(k)} + p^{(k)}/(1 - \lambda_{\text{loop}}^{(k)}) \cdot \lambda_x$
83:             end for
84:         end if
85:     end for
86:     return $\langle \lambda_{\text{goal}}, \lambda_{\text{fail}}, \lambda_{\text{noter}}\rangle$
87: end function
```

5.2. Correctly counting looping histories

In order to account for looping histories, we draw on the following insight. Suppose that for a history $h^{(0:k)}$, there is a history h_{loop} from $h^{(k)}$ with $\mathit{end}(h_{\text{loop}}) = h^{(k)}$, and another history h_{goal} from $h^{(k)}$ that terminates in a goal state. A *system* with an FSC has the Markov property such that both the next action of the controller and the next state of the environment are defined by the current combined state. As a result, $h \cdot h_{\text{goal}}$, $h \cdot h_{\text{loop}} \cdot h_{\text{goal}}$, $h \cdot h_{\text{loop}} \cdot h_{\text{loop}} \cdot h_{\text{goal}}$ and so on are all valid goal histories, where the one with m repetitions of h_{loop} has likelihood $\ell(h)(\ell(h_{\text{loop}}))^m \ell(h_{\text{goal}})$. These likelihoods form a geometric progression, whose sum for all $m \geq 0$ is $\ell(h)\ell(h_{\text{goal}})/(1 - \ell(h_{\text{loop}}))$. (The existence of two distinct histories from $h^{(k)}$, namely h_{goal} and h_{loop}, guarantees that $\ell(h_{\text{loop}}) < 1$.) In the following, we describe how to utilize this argument.

We said that PANDOR enumerates the histories of FSCs; let $h_{\text{curr}}^{(0:n)}$ be the currently simulated history at some point during execution. Now we construct the set of all goal histories from pairwise disjoint sets of histories, one set for each $0 \leq k \leq n$.

Denote by H_{loop}^k the set of histories h_{loop} from $h_{\text{curr}}^{(k)}$ with the following properties:

L1. $h_{\text{loop}}^{(0)} = h_{\text{curr}}^{(k)}$

L2. $\mathit{end}(h_{\text{loop}}) = h_{\text{curr}}^{(k)}$

L3. apart from its first and last element, h_{loop} doesn't contain $h_{\text{curr}}^{(k)}$, and

L4. no element of h_{loop} is equal to $h_{\text{curr}}^{(i)}$ for any $i < k$.

Furthermore, let $H_{\text{goal+}}^k$ denote the set of histories h_{goal} from $h_{\text{curr}}^{(k-1)}$ with the following properties:

W1. $h_{\text{goal}}^{(0)} = h_{\text{curr}}^{(k-1)}$

W2. $s_{\text{end}} \in \mathcal{G}$ and $\gamma(q_{\text{end}}, \Omega(s_{\text{end}})) = \texttt{stop}$ (where $\langle s_{\text{end}}, q_{\text{end}} \rangle \equiv \mathit{end}(h_{\text{goal}})$)

W3. apart from its first element, h_{goal} doesn't contain $h_{\text{curr}}^{(k-1)}$

W4. no element of h_{goal} is equal to $h_{\text{curr}}^{(i)}$ for any $i < k - 1$.

Finally, H_{goal}^k consists of histories h_{goal} that fulfill **W1–W4** and also **W5**:

W5. $h_{\text{goal}}^{(1)} \neq h_{\text{curr}}^{(k)}$.

Condition **W2** states that h_{goal} is a goal history. The other conditions collectively ensure that if $h^\star$ is a goal history from $h^{(k-1)}$ such that $h^{\star(1:)}$ doesn't repeat any element of $h^{(:k-1)}$, then it is either **W1–W5** for k, or it can be uniquely pieced together by elements of H_{loop}^k and H_{goal}^{k+1}.

Lemma 1. *If $h^\star \in (H_{\text{goal+}}^k - H_{\text{goal}}^k)$, i.e., $h^\star$ satisfies* **W1–W4** *but not* **W5***, then there is a unique $m \geq 0$ such that for some $h_{\text{loop},i} \in H_{\text{loop}}^k$ for all $1 \leq i \leq m$ and some $h_{\text{goal}} \in H_{\text{goal+}}^{k+1}$, we have:*

$$h^\star = h_{\text{curr}}^{(k-1:k)} \cdot h_{\text{loop},1} \cdot \ldots \cdot h_{\text{loop},m} \cdot h_{\text{goal}}. \tag{3}$$

Let $\tilde{\alpha}_{\text{goal}}^{(k)}$ and $\tilde{\lambda}_{\text{loop}}^{(k)}$ denote the sum of the likelihood of the elements of $H_{\text{goal}}^{(k)}$ and $H_{\text{loop}}^{(k)}$.

$$\tilde{\alpha}_{\text{goal}}^{(k)} = \sum_{h_{\text{goal}} \in H_{\text{goal}}^{(k)}} \ell(h_{\text{goal}}) \tag{4}$$

$$\tilde{\lambda}_{\text{loop}}^{(k)} = \sum_{h_{\text{loop}} \in H_{\text{loop}}^{(k)}} \ell(h_{\text{loop}}) \tag{5}$$

Notation. $\tilde{\alpha}_{\text{loop}}^{(i,k)}$ and $\tilde{\alpha}_{\text{goal}}^{(k+1)}$ denote the probability of the system looping back to $h_{\text{curr}}^{(i)}$ or reaching a goal state from $h_{\text{curr}}^{(k)}$ without following the current history along $h_{\text{curr}}^{(k+1)}$. $\tilde{\lambda}_{\text{loop}}^{(k)}$ and $\tilde{\lambda}_{\text{goal}}^{(k+1)}$ denote the probability of the system looping back to $h_{\text{curr}}^{(i)}$ or reaching a goal state from $h_{\text{curr}}^{(k)}$, possibly while following the current history along $h_{\text{curr}}^{(k+1)}$, or $h_{\text{curr}}^{(k+1)}$ and $h_{\text{curr}}^{(k+2)}$, etc.

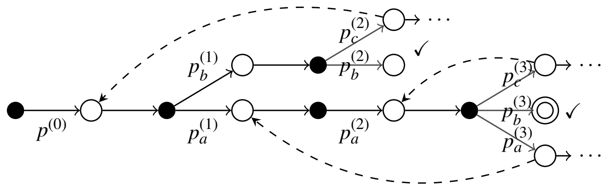

Figure 3: An example AND-OR tree corresponding to the execution of PANDOR, with the root at the left. Numbers on the edges are the transition probabilities; empty circles: OR nodes (a combined state of the system); filled black circles: AND nodes (some action made, with the outcome yet uncertain); checkmark: system terminates in a goal state; a dashed arrow indicates that the relevant states are equal in a looping history, with an infinitely repeating tree below. When all nodes are explored and the double circle is the current node in the OR-step (thus h_{curr} is a sequence of the three states before), the non-zero α values are $\alpha^{(1)}_{\text{goal}} = p^{(1)}_b p^{(2)}_b$, $\alpha^{(3)}_{\text{goal}} = p^{(3)}_b$, $\alpha^{(0,0)}_{\text{loop}} = p^{(1)}_b p^{(2)}_c$, $\alpha^{(1,2)}_{\text{loop}} = p^{(3)}_a$, $\alpha^{(2,2)}_{\text{loop}} = p^{(3)}_c$.

The likelihood of the system terminating in a goal state (s.t.i.g.) if started in the initial state $h^{(0)}_{\text{curr}}$ can be calculated inductively, as follows.

Lemma 2. *Let $\tilde{\lambda}^{(k)}_{\text{goal}}$ denote the probability of s.t.i.g. after the history $h^{(:k-1)}_{\text{curr}}$ without repeating any combined state of $h^{(:k-1)}_{\text{curr}}$. Then the following hold for any $0 \le k < n$:*

$$\tilde{\lambda}^{(n)}_{\text{goal}} = \tilde{\alpha}^{(n)}_{\text{goal}}, \tag{6}$$

$$\tilde{\lambda}^{(k)}_{\text{goal}} = p^{(k)} \tilde{\lambda}^{(k+1)}_{\text{goal}} / (1 - \tilde{\lambda}^{(k)}_{\text{loop}}) + \tilde{\alpha}^{(k)}_{\text{goal}}, \tag{7}$$

where $p^{(k)}$ is the probability of transitioning from $h^{(k-1)}_{\text{curr}}$ to $h^{(k)}_{\text{curr}}$, i.e.

$$p^{(k)} = \Delta(s^{(k)}_{\text{curr}} \mid s^{(k-1)}_{\text{curr}}, \delta(q^{(k-1)}_{\text{curr}}, \Omega(h^{(k-1)}_{\text{curr}}))). \tag{8}$$

This is because $\tilde{\lambda}^{(k)}_{\text{goal}}$ is equal to the probability of s.t.i.g. from $h^{(:k)}_{\text{curr}}$ without repeating $h^{(:k-1)}_{\text{curr}}$ plus the probability of s.t.i.g. from $h^{(:k-1)}_{\text{curr}}$ without repeating $h^{(:k-1)}_{\text{curr}}$ if the kth combined state is not equal to $h^{(k)}_{\text{curr}}$. The latter is simply $\tilde{\alpha}^{(k)}_{\text{goal}}$, and the former is the sum of an infinite geometric series whose ratio is $\tilde{\lambda}^{(k)}_{\text{loop}}$.

The probability of s.t.i.g. from $h^{(0)}_{\text{curr}}$ is **LGT**, and with the natural definition[2] of $h^{(:-1)} := \langle\rangle$, this probability is equal to $\tilde{\lambda}^{(0)}_{\text{goal}}$.

Lemma 3. **LGT** $= \tilde{\lambda}^{(0)}_{\text{goal}}$

5.3. Measuring the looping goal likelihood

While PANDOR simulates the histories of a controller, it maintains variables $\alpha^{(k)}_{\text{goal}}$ for each k, which is the sum of likelihoods of the elements $h \in H^k_{\text{goal}}$ such that $h^{(k-1)}_{\text{curr}} \cdot h$ has already been visited. (This is done by increasing the relevant $\alpha^{(k)}_{\text{goal}}$ in the OR-step when the controller terminates in a goal state.) The following inequality trivially holds:

Lemma 4. $\alpha^{(k)}_{\text{goal}} \le \tilde{\alpha}^{(k)}_{\text{goal}}$, *with equality when every non-looping goal history from $h^{(k-1)}_{\text{curr}}$ has been visited.*

Tracking the likelihoods of the encountered loops is done with a two-dimensional array α_{loop}, with initial values all zero. When a looping history is found in an OR-step, the relevant entry of α_{loop} is increased by the probability of the last state transition. Formally, after a history of $h^{(0:n)}_{\text{curr}}$, if $\langle q, s\rangle = h^{(k)}_{\text{curr}}$ for some $k \le n$, such that the likelihood of the loop is $\ell(h^{(k:n)}_{\text{curr}}) \cdot p = p_{\text{loop}} < 1$, then $\alpha^{(k,n)}_{\text{loop}}$ is increased by p, the probability of the last transition. (Note that in the OR-STEP function, h_{curr} doesn't yet include the current combined state.)

[2] $h^{(:-1)}_{\text{curr}}$ can be interpreted as the history before $h^{(0)}_{\text{curr}}$, which is an empty sequence.

For each $k < m \leq n$, the product $\alpha_{\text{loop}}^{(k,m)} \cdot \ell(h_{\text{curr}}^{(k:m)})$ is the sum of likelihoods of those elements $h \in H_{\text{loop}}^{k}$ where $h_{\text{curr}}^{(k)} \cdot h$ has already been visited, and h starts with $h_{\text{curr}}^{(k:m)}$ but not with $h_{\text{curr}}^{(k:m+1)}$. Thus, if the system starts in $h_{\text{curr}}^{(m)}$, the probability estimate that it will reach $h_{\text{curr}}^{(k)}$ before any other element of $h_{\text{curr}}^{(0:m)}$, provided the first step is not $h_{\text{curr}}^{(m+1)}$, is $\alpha_{\text{loop}}^{(k,m)}$.

Now we define $\lambda_{\text{loop}}^{(k)}$, which is a lower bound on $\tilde{\lambda}_{\text{loop}}^{(k)}$, based on the histories explored up to the current point of the execution. After a history of $h_{\text{curr}}^{(k+1:m)}$, the system can loop to $h_{\text{curr}}^{(k)}$ from any of $h_{\text{curr}}^{(i)}$ with $k \leq i \leq n$ (meaning the step after $h_{\text{curr}}^{(i)}$ is not $h_{\text{curr}}^{(i+1)}$), which for a given i has a probability of $\alpha_{\text{loop}}^{(k,i)} \prod_{j=k+1}^{i} p^{(j)}/(1 - \lambda_{\text{loop}}^{(j)})$.

$$
\begin{aligned}
\lambda_{\text{loop}}^{(k)} &= \alpha_{\text{loop}}^{(k,k)} + \\
&\quad + p^{(k+1)} \left(1 - \lambda_{\text{loop}}^{(k+1)}\right)^{-1} \Big(\alpha_{\text{loop}}^{(k,k+1)} + \\
&\qquad + p^{(k+2)} \left(1 - \lambda_{\text{loop}}^{(k+2)}\right)^{-1} \Big(\alpha_{\text{loop}}^{(k,k+2)} + \ldots \\
&\qquad\quad + p^{(n-1)} \left(1 - \lambda_{\text{loop}}^{(n-1)}\right)^{-1} \Big(\alpha_{\text{loop}}^{(k,n-1)} + \\
&\qquad\qquad + p^{(n)} \left(1 - \lambda_{\text{loop}}^{(n)}\right)^{-1} \alpha_{\text{loop}}^{(k,n)}\Big) \cdots \Big)\Big) \\
&= \sum_{i=k}^{n} \left(\alpha_{\text{loop}}^{(k,i)} \prod_{j=k+1}^{i} p^{(j)}/(1 - \lambda_{\text{loop}}^{(j)}) \right).
\end{aligned} \tag{9}
$$

This calculation is done in lines 69–72 of Alg. 3. The following result is proved easily by induction on k.

Lemma 5. $\lambda_{\text{loop}}^{(k)} \leq \tilde{\lambda}_{\text{loop}}^{(k)}$, *with equality when every once-looping history from* $h_{\text{curr}}^{(k)}$ *has been visited.*

We define $\lambda_{\text{goal}}^{(k)}$ inductively with the help of $\alpha_{\text{goal}}^{(k)}$ values, for all $0 \leq k < n$:

$$\lambda_{\text{goal}}^{(n)} := \alpha_{\text{goal}}^{(n)}, \tag{10}$$

$$\lambda_{\text{goal}}^{(k)} := p^{(k)} \lambda_{\text{goal}}^{(k+1)}/(1 - \lambda_{\text{loop}}^{(k)}) + \alpha_{\text{goal}}^{(k)}, \tag{11}$$

where $p^{(k)}$ is still the probability of transitioning from $h_{\text{curr}}^{(k-1)}$ to $h_{\text{curr}}^{(k)}$. Using Lemmas 4 and 5, it can be seen that $\lambda_{\text{goal}}^{(k)}$ is a lower bound on $\tilde{\lambda}_{\text{goal}}^{(k)}$:

Lemma 6. $\lambda_{\text{goal}}^{(k)} \leq \tilde{\lambda}_{\text{goal}}^{(k)}$, *with equality when every non-looping goal history from* $h_{\text{curr}}^{(k-1)}$ *has been visited.*

In Alg. 3, the CalcLambda function calculates $\lambda_{\text{goal}}^{(0)}$ based on $\boldsymbol{\alpha}_{\text{goal}}$, $\boldsymbol{\alpha}_{\text{loop}}$, and h_{curr}, and this $\lambda_{\text{goal}}^{(0)}$ serves as the basis for termination in the AND-step. ($\boldsymbol{\alpha}_{\text{goal}}$ denotes the collection of $\alpha_{\text{goal}}^{(i)}$ values for all i; similarly $\boldsymbol{\alpha}_{\text{loop}}$.)

5.4. Failing and non-terminating histories

We can treat *failing histories* (histories that terminate in a non-goal state) and histories that contain a non-decaying loop similarly to goal histories. We account for them via $\alpha_{\text{fail}}^{(k)}$ and $\alpha_{\text{noter}}^{(k)}$ values in a manner analogous to $\alpha_{\text{goal}}^{(k)}$; $\lambda_{\text{fail}}^{(k)}$ is defined similarly to $\lambda_{\text{goal}}^{(k)}$. We have similar results for the relevant λ values as before:

Lemma 7. $\lambda_{\text{fail}}^{(k)} \leq \tilde{\lambda}_{\text{fail}}^{(k)}$, *with equality when every non-looping failing history from* $h_{\text{curr}}^{(k-1)}$ *has been visited.*

Calculating $\lambda_{\text{noter}}^{(k)}$ is done as $\lambda_{\text{goal}}^{(k)}$, but it is peculiar in that multiple decaying loops can add up to a history that cannot be extended into a terminating history (Fig. 2.3). When no history from $h_{\text{curr}}^{(k)}$ terminates, $\tilde{\lambda}_{\text{loop}}^{(k+1)} + \tilde{\lambda}_{\text{noter}}^{(k)} = 1$ – for example, $h_{\text{curr}}^{(k)}$ has an extension with likelihood 0.1 with a non-decaying loop at the end, and another extension with likelihood 0.9 loops back to $h_{\text{curr}}^{(k)}$. When this happens, the values of $\alpha_{\text{loop}}^{(i,j)}$ are zeroed out for all $i \geq k$ and $j \geq k$, and $\lambda_{\text{noter}}^{(k)}$ is assigned $p^{(k)}$. (Lines 73–76.) With this caveat, the inequality result for λ_{noter} is the following:

Lemma 8. $\lambda_{\text{noter}}^{(k)} \leq \tilde{\lambda}_{\text{noter}}^{(k)}$, *with equality when every once-looping history from $h_{\text{curr}}^{(k-1)}$ has been visited.*

We now have classified histories as those that could be extended into terminating ones (either in a goal state or not) and those that have a non-decaying loop.

Lemma 9. $\tilde{\lambda}_{\text{goal}}^{(0)} + \tilde{\lambda}_{\text{fail}}^{(0)} = \mathbf{LTER} = 1 - \tilde{\lambda}_{\text{noter}}^{(0)}$

5.5. Rolling up the α values

Next, we see what should happen to the $\boldsymbol{\alpha}_{\text{goal}}$, $\boldsymbol{\alpha}_{\text{fail}}$, $\boldsymbol{\alpha}_{\text{noter}}$, $\boldsymbol{\alpha}_{\text{loop}}$ values when the current history changes. Clearly, when h_{curr} is extended, $\boldsymbol{\alpha}_{\text{goal}}$, $\boldsymbol{\alpha}_{\text{fail}}$, and $\boldsymbol{\alpha}_{\text{noter}}$ should be extended with an additional zero item, and $\boldsymbol{\alpha}_{\text{loop}}$ should be extended with an additional row&column of zeros (lines 6–7).

When h_{curr} is shortened when the AND-STEP function returns, the last element (or last row&column) of these variables needs to be integrated to the previous ones before deleting them (CUMULATEALPHA function at lines 17, 45). The approach is similar to how the first iteration of λ_{goal} and λ_{loop} was calculated: in fact, the new α values are chosen so that CALCLAMBDA returns the same values before and after CUMULATEALPHA is called. It is important to note that this change in the α values doesn't affect our earlier results:

Lemma 10. *After $\boldsymbol{\alpha}_{\text{goal}}$, $\boldsymbol{\alpha}_{\text{fail}}$, $\boldsymbol{\alpha}_{\text{noter}}$, $\boldsymbol{\alpha}_{\text{loop}}$ are assigned the values returned by* CUMULATEALPHA*, Lemmas 5, 6, 7, 8 still hold.*

5.6. Correctness of the search

Our final theorem states that PANDOR meets desiderata **D1–D3**, and solves Problem 1 correctly.

Theorem 3. *Given a planning problem $\mathcal{P}$, integer N, and $LGT^\star \in (0,1)$, the search algorithm* PANDOR *is* sound *and* complete: *every FSC C returned by* PANDOR-SYNTH *is N-bounded and* **LGT** $\geq$ *$LGT^\star$ for $\mathcal{P}$, and if there exists an N-bounded controller that is* **LGT** $\geq$ *$LGT^\star$ for $\mathcal{P}$, then one such FSC will be found.*

Proof sketch. **Soundness.** A controller C is returned by PANDOR-SYNTH only if $LGT^\star \leq \lambda_{\text{goal}}^{(0)}$ (line 2.11). By Lemmas 3 and 6:

$$\lambda_{\text{goal}}^{(0)} \leq \tilde{\lambda}_{\text{goal}}^{(0)} = \mathbf{LGT}, \tag{12}$$

making the controller $LGT^\star \leq$ **LGT** for $\mathcal{P}$.

Completeness. Suppose there exists an N-bounded controller C_{good} which is **LGT** $\geq LGT^\star$ for $\mathcal{P}$.

Suppose a smaller controller $C' \prec C_{\text{good}}$ is rejected (property †). A failing or non-terminating history of C' has the same property for C_{good} as well, and is a valid history for the system $\langle \mathcal{E}, C_{\text{good}} \rangle$ (property ‡). A failing or non-terminating history does not terminate in a goal state (property ★).

$$\begin{aligned}
\mathbf{LGT}(C_{\text{good}}) &\leq \\
&\leq 1 - \tilde{\lambda}_{\text{fail}}^{(0)}(C_{\text{good}}) - \tilde{\lambda}_{\text{noter}}^{(0)}(C_{\text{good}}) && \text{by } \star \\
&\leq 1 - \tilde{\lambda}_{\text{fail}}^{(0)}(C') - \tilde{\lambda}_{\text{noter}}^{(0)}(C') && \text{by } \ddagger \\
&\leq 1 - \lambda_{\text{fail}}^{(0)}(C') - \lambda_{\text{noter}}^{(0)}(C') && \text{Lemma 7, 8} \\
&< LGT^\star && \text{by } \dagger
\end{aligned} \tag{13}$$

This is against the premise that C_{good} is **LGT** $\geq LGT^\star$, contradiction: no smaller controller is rejected.

Suppose that when the current controller is $C' \prec C_{\text{good}}$, at a non-deterministic choice the next controller C'' is such that $C' \prec C'' \npreceq C_{\text{good}}$. If this execution branch of C'' does not fail, then a controller was returned, which was **LGT** $\geq LGT^\star$ by the soundness of the search.

In a finite environment, a system with an FSC has finitely many combined states. This implies that the number and length of the at-most-once-looping histories is bounded above, and so is the number of OR-steps required to explore these histories. At the end of the execution, every such history of C has been simulated, resulting in $\lambda_{\text{goal}}^{(0)} + \lambda_{\text{fail}}^{(0)} + \lambda_{\text{noter}}^{(0)} = 1$ by Lemmas 6, 7, 8, and 9. At this time one of the termination conditions is fulfilled.

If at every non-deterministic choice, C'' is chosen such that $C' \prec C'' \preceq C_{\text{good}}$, then as C'' can't be rejected (by the argument above), either C'' or an extension of it will be returned. □

5.7. Planning for minimum likelihood of termination

The above results outline how our algorithm can plan for a minimum **LGT**, but only minor modifications are required for setting a lower bound on **LTER** as well. Formally,

Problem 3. Given a planning problem $\mathcal{P}$, an integer N, $LTER^\star \in (0,1)$, $LGT^\star \in (0,1)$, find a finite-state controller with at most N states that is **LTER** $\geq LTER^\star$ and **LGT** $\geq LGT^\star$ for $\mathcal{P}$.

To solve this, the only change required in the pseudocode is in lines 11–15, where we simply extend the criteria for early termination and failure. These changes are shown in Alg. 4.

Algorithm 4 Changes required in Algorithm 2 to specify a lower bound on both **LTER** and **LGT**.

```
Require: $\mathcal{P}$, $N$, $LGT^\star$: as before,
         $LTER^\star$: the desired minimum LTER

11: if $\lambda_{goal} \geq LGT^\star$ and $\lambda_{goal} + \lambda_{fail} \geq LTER^\star$
12:     return $C$
13: else if $1 - \lambda_{fail} - \lambda_{noter} < LGT^\star$ or $1 - \lambda_{noter} < LTER^\star$
14:     fail this non-deterministic branch
15: end if
```

This new search process is sound and complete with respect to Problem 3.

Theorem 4. *Given a planning problem $\mathcal{P}$, integer N, $LTER^\star \in (0,1)$, and $LGT^\star \in (0,1)$, the search algorithm* PANDOR *is* sound *and* complete*: every FSC C returned by* PANDOR-SYNTH *is N-bounded and* **LTER** $\geq LTER^\star$ *and* **LGT** $\geq LGT^\star$ *for $\mathcal{P}$, and if there exists an N-bounded controller that is* **LTER** $\geq LTER^\star$ *and* **LGT** $\geq LGT^\star$ *for $\mathcal{P}$, then one such FSC will be found.*

Proof sketch. **Soundness.** A controller C is returned by PANDOR-SYNTH only if $LGT^\star \leq \lambda^{(0)}_{goal}$ and $LTER^\star \leq \lambda^{(0)}_{goal} + \lambda^{(0)}_{fail}$ (line 4.11). By Lemmas 3, 6, and 7, 9:

$$\lambda^{(0)}_{goal} \leq \tilde{\lambda}^{(0)}_{goal} = \mathbf{LGT}, \tag{14}$$

$$\lambda^{(0)}_{goal} + \lambda^{(0)}_{fail} \leq \tilde{\lambda}^{(0)}_{goal} + \tilde{\lambda}^{(0)}_{fail} = \mathbf{LTER}, \tag{15}$$

making the controller $LGT^\star \leq$ **LGT** and $LTER^\star \leq$ **LTER** for $\mathcal{P}$.

Completeness. Suppose there exists an N-bounded controller C_{good} which is **LGT** $\geq LGT^\star$ and **LTER** $\geq LTER^\star$ for $\mathcal{P}$.

Suppose a smaller controller $C' \prec C_{good}$ is rejected – this can happen either for not meeting the bound on $LGT^\star$ (property †) or on $LTER^\star$ (††). A failing or non-terminating history of C' has the same property for C_{good} as well, and is a valid history for the system $\langle \mathcal{E}, C_{good} \rangle$ (property ‡). A failing or non-terminating history does not terminate in a goal state (property ⋆). If the controller is rejected for †, then inequality 13 holds, otherwise:

$$\begin{aligned}
\mathbf{LTER}(C_{good}) &\leq 1 - \tilde{\lambda}^{(0)}_{noter}(C_{good}) && \text{by } \star \\
&\leq 1 - \tilde{\lambda}^{(0)}_{noter}(C') && \text{by } \ddagger \\
&\leq 1 - \lambda^{(0)}_{noter}(C') && \text{Lemma 9} \\
&< LTER^\star && \text{by } \dagger\dagger
\end{aligned} \tag{16}$$

Either Eq. 13 or Eq. 16 is against the premise that C_{good} is **LGT** $\geq LGT^\star$ and **LTER** $\geq LTER^\star$, contradiction: no smaller controller is rejected.

The algorithm terminates for the reasons described in the proof of Theorem 3.

Suppose that when the current controller is $C' \prec C_{good}$, at a non-deterministic choice the next controller C'' is such that $C' \prec C'' \npreceq C_{good}$. If this execution branch of C'' does not fail, then a controller was returned, which was **LGT** $\geq LGT^\star$ and **LTER** $\geq LTER^\star$ by the soundness of the search.

If at every non-deterministic choice, C'' is chosen such that $C' \prec C'' \preceq C_{good}$, then as C'' can't be rejected, either C'' or an extension of it will be returned. □

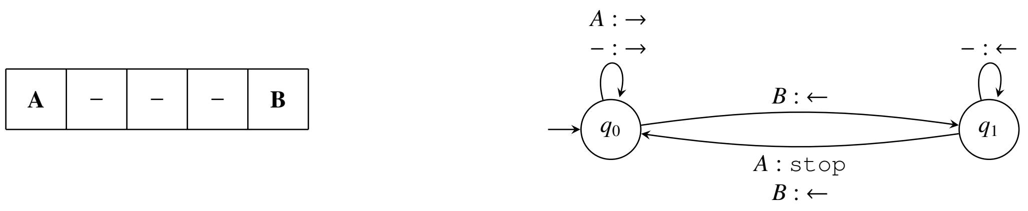

Figure 4: *Left*: The Noisy Hall-A(1 × 5) planning problem has the same state space as the deterministic variant, but every action leaves the environment in the same state with probability 0.5. *Right*: A correct finite state controller for this probabilistic problem, as synthesized by PANDOR.

5.8. Time and space complexity

In order to explore the whole environment with a given controller, we need to take $O(b^{h_{\max}})$ steps, where b is the branching factor at the AND-step (the maximum number of outcomes of an action), and $h_{\max}$ is the length of the longest possible history without a repeated state (i.e. $h_{\max} \leq |\mathcal{S}|$). At every AND-step, PANDOR needs to calculate the λ vectors, which takes $O(h_{\max}^2)$ steps due to the size of α_{loop}. This exploration needs to be done, usually to different depths, for every possible non-isomorphic N-bounded FSC, which we denote by $\#_C$. A controller is defined by its transitions, hence $\#_C < (N \cdot |O|)^{N \cdot |\mathcal{A}|}$. It follows that the time complexity of the algorithm is $O(b^{h_{\max}} \cdot \#_C \cdot h_{\max}^2) = O(b^{h_{\max}} \cdot \#_C)$.

These numbers are realized in an adversarial environment with extremely low probability (depending on the action/outcome selection); in most realistic situations (i.e., non-adversarial environments), failure/success would be orders of magnitude quicker. First, whenever a controller is found to not meet the desired $LGT^\star$, all of its extensions are discarded immediately, leaving us with $\#_C \ll (N \cdot |O|)^{N \cdot |\mathcal{A}|}$. Secondly, most controllers are unable to explore the whole environment, and require orders of magnitude fewer than $b^{h_{\max}}$ steps. We believe the search process could be further improved using admissible heuristics.

Analogously, we need to store the alpha vectors and the α_{loop} matrix at each controller extension, which each require $O(h_{\max}^2)$ space. If the maximum number of controller transitions is $\#_T$ (where $\#_T \leq N \cdot |O|$), then this results in a space complexity of $O(h_{\max}^2 \cdot \#_T)$.

6. Preliminary Empirical Evaluations

The contributions of this paper are solely on the theoretical front. Nonetheless, we have released a proof-of-concept implementation of the pseudocode in Alg. 2 and 3.[3] In our preliminary evaluations, we observed that in deterministic domains, our planner has the same runtime as the planner of [12], and the difference is only a small linear factor for additional bookkeeping. But suppose we were to consider a noisy variant of the Hall-A domain in Fig. 1 on page 2, where every action has a 50-50% probability of either succeeding or leaving the current state unchanged. Here, because moving left in cell B is not guaranteed to succeed, we can see that an extra transition is required to move the agent out of B in case the first attempt fails ($q_1 \xrightarrow{B\,:\,\leftarrow} q_0$). And indeed, the FSC synthesized by PANDOR contains this transition, as shown in Figure 4.

In addition to the above, we now consider two new domains designed for testing PANDOR, and the performance and output of the planner on them. We remark that although there are examples of interesting loopy planning problems in the literature [7, 13], as well as noisy domains for probabilistic planners [21], challenging problems that test the correctness of loopy plans in noisy domains are still needed.

6.1. The BridgeWalk domain

This environment simulates an agent that stands on the handrail of a bridge: on one side the sidewalk, on the other side the river, and on the handrail n steps away is the goal. With every step taken on the handrail, the agent has a 0.1 probability of falling into the river (which is an absorbing state, i.e., it only has state transitions to itself), and

[3]The implementation can be accessed at `https://github.com/treszkai/pandor`

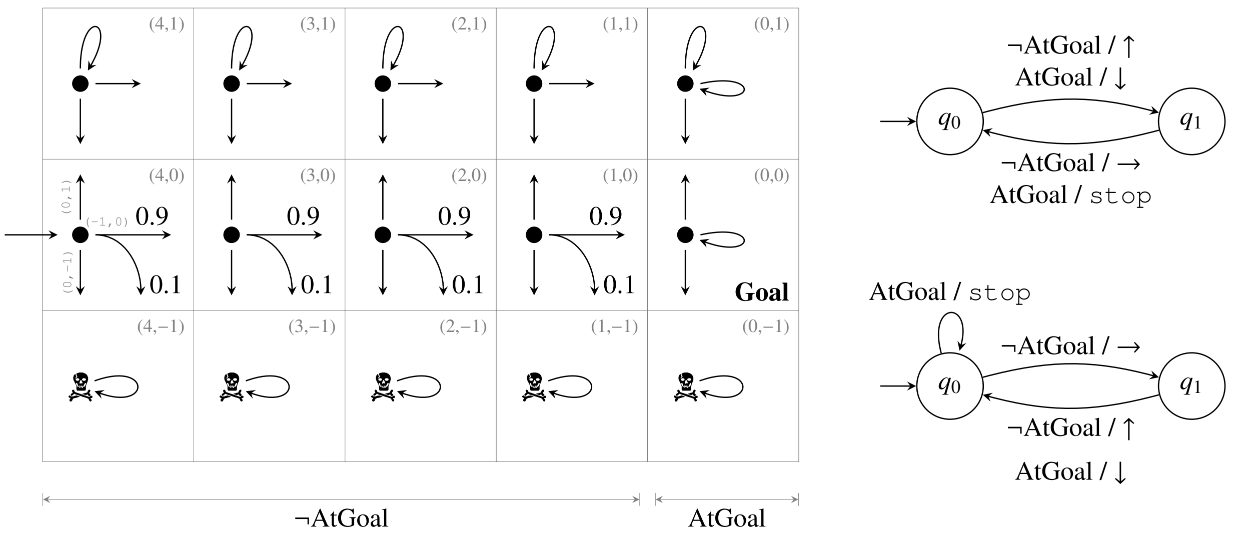

Figure 5: *Left*: The BridgeWalk(4) environment. ☠ indicates a dead end state; other states are marked with ●. For readability purposes, most action names are omitted. *Right*: Two synthesized controllers for the BridgeWalk domain. *Above*: $LGT^\star = 1$. *Below*: $LGT^\star = 0.9$.

0.9 probability of moving forward one step. Every other move is deterministic, which include climbing on or off the handrail (moving up or down), or stepping forward on the sidewalk (moving right in the top row). Trying to move off the grid leaves the agent in the same state. The observation space is {AtGoal, ¬AtGoal}, indicating whether the agent is in line with the goal (but potentially on the sidewalk).

The state space of the BridgeWalk(n) environment is an $(n+1) \times 3$ grid world, with the initial state $(n, 0)$, the only goal state $(0, 0)$. The actions are `(0,1)`, `(0,-1)`, `(-1,0)` (in the figure ↑, ↓, →, respectively), with transition probabilities as described above. (The actions act as vector addition to the current state.) The observation at state $\langle x, y \rangle$ is AtGoal if $x = n$, and ¬AtGoal otherwise. The environment is illustrated on the left of Fig. 5.

In this problem, there are multiple goal histories from the initial state, and various controllers with different values of **LGT** and state counts. A single-state controller can only attempt n forward steps: due to the noisy actions, this controller has **LGT** $= 0.9^n$. When the maximum number of states is 2, different $LGT^\star$ parameters result in different controllers – these are shown on the right of Fig. 5.

This environment demonstrates that our planner fulfills **D1**: PANDOR is capable of synthesizing controllers that are less than perfect but whose **LGT** exceeds the user-defined $LGT^\star$ parameter. Table 1 lists the number of steps and time needed to synthesize the controllers.

6.2. *Noisy Hall-A*

Our Noisy Hall-A domains are the noisy versions of the deterministic Hall-A problems from [7], which came in two variants; we described the one-dimensional variant at the beginning of this section. The synthesized controller demonstrates that our algorithm satisfies **D2** and **D3**: it is able to handle looping histories correctly.

We also modified the two-dimensional variant, Halls-A($n \times n$), into the Noisy Halls-A($n \times n$) environment (Fig. 6), similarly as we did with the one-dimensional version. This is a square-shaped corridor, with the initial state in the upper right corner. The goal is to visit every corner, return to state A, and terminate there (therefore every state has an unobservable property of which corner has been visited). The observations are A, B, C, D, $-$, depending on whether the current state is in either corner or a corridor cell. It can be shown that every correct controller has at least 4 states; the controller synthesized by PANDOR is shown in Figure 6. An analysis of the histories of this controller shows that it makes some sub-optimal moves, such as traversing the $B---C$ corridor back and forth when an attempt to move left from B in state q_3 fails. These run-time inefficiencies have a limited effect on planning time, because during planning no combined state is visited more than twice.

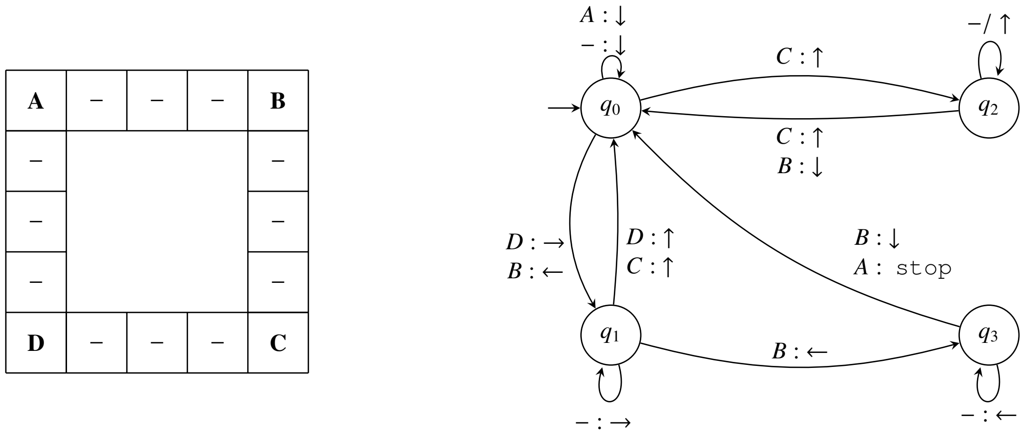

Figure 6: *Left*: The Noisy Hall-A(5 × 5) planning problem. *Right*: A correct finite state controller for this probabilistic problem, as synthesized by PANDOR.

Domain	Instance	N	Steps	Time	$LGT^\star$	**LGT**
BridgeWalk	4	1	6	<0.01 s	0.6	$0.9^4 \approx 0.66$
	4	2	124	0.04 s	0.999	1
	100	2	1,034	7.7 s	0.999	1
Noisy Hall-A	1×4	2	40	0.01 s	0.999	1
	1×100	2	424	4.4 s	0.999	1
	3×3	4	9,468	6.4 s	0.999	1
	4×4	4	11,126	9.4 s	0.999	1
	5×5	4	12,784	11.9 s	0.999	1

Table 1: Summary of experimental results. N is the maximum number of allowed controller states, which is equal to the number of states of the synthesized controllers. $LGT^\star$ is the parameter used for synthesizing the controller; **LGT** is the property of the synthesized controller. The number of simulated steps includes those that belong to a discarded controller. Times measured on a computer with a 2.7 GHz Intel Core i5 processor from 2014.

7. Related Work

Our results are related to a number of recent approaches on bounded search and FSC synthesis, but as we discuss below, the nature of our results and the thrust of our proof strategy is significantly different from these approaches. At the outset, our contributions should be seen as a full generalization of the work of Hu and De Giacomo [12] to stochastic domains, in that it provides a generic technique for FSC synthesis in a whole range of planning frameworks (cf. [12]) that can now be considered with probabilistic nondeterminism.

The work of Hu and De Giacomo [12] is positioned in the area of generalized planning. We will briefly touch on approaches to generating loopy plans, and then discuss related correctness concerns.

Early approaches to loopy plans can be seen as deductive methodologies, often influenced by program synthesis and correctness [11]. Manna and Waldinger [23] obtained recursive plans by matching induction rules, and Stephan and Biundo [34] refine generic plan specifications, but required input from humans. See [22] for a recent approach using induction.

Most recent proposals differ considerably from this early work using deduction:

- Levesque [19] expects two parameters with the planning problem; the approach plans for the first parameter, winds it to form loops and tests it for the second.

- Winner and Veloso [36] synthesize a plan sequence with partial orderings, and exploit repeated occurrences of subplans to obtain loops.
- Srivastava [32] considers an abstract state representation that groups objects into equivalences classes, the idea being that any concrete plan can be abstracted wrt these classes and repeated occurrences of subplans can be leveraged to generate compact loopy plans.
- Bonet et al. [7] integrate the dynamics of a memoryless plan with a planning problem, and convert that to a conformant planning problem; the solution to this latter problem is shown to generalize to multiple instances of the original problem.
- Hu and De Giacomo [12] propose a bounded AND/OR search procedure that is able to synthesize loopy plans, which is what we build on.

On the matter of correctness, Levesque [18] argued that generalized plans be tested for termination and correctness against all problem instances; Lin and Levesque [20] extended this account to define goal achievability. In later work, Cimatti et al. [10] defined the notions of weak, strong and strong cyclic solutions in the presence of nondeterminism.[4] These notions are widely used in the planning community [6]; see, for example, [3] for an account of strong planning with a sensor model. Recently, Srivastava et al. [33] synthesize loopy plans in domains with nondeterministic quantitative effects, for which strong cyclic solutions are studied. Our account of correctness is based on [2], which generalized Levesque's account [18].

Synthesizing FSCs is a very active area of research within Markov Decision Processes (MDPs) and partially observable MDPs (POMDPs) [26, 29, 1]. But the majority of algorithms in this space either solve an approximation of the problem, or they come without correctness guarantees. In contrast, emphasizing correctness, Junges et al. [17] show how FSC synthesis for POMDPs can be reduced to parameterized Markov chains, under the requirement of *almost-sure plans that do not enter bad states.* Similarly, Chatterjee et al. [9] propose the synthesis of *almost-sure plans* by means of a SAT-based oracle. Not only are the algorithms significantly different from our own, but the correctness specification too is formulated differently. Thus, our contributions are complementary to this major body of work, and orthogonal to a large extent; for the future, it would be interesting to relate these strategies more closely.

When it comes to similarity to our algorithms, there are a variety of approaches based on bounded AND-OR search, such as AO*, LAO*, and LRTDP (e.g., [5]). Here too, the specification criteria, the correctness bound and the nature of the analysis are largely orthogonal to ours. For example, LAO* allows for loops, but the solution is not necessarily a sound and complete N-bounded FSC: it yields a partial policy and does not allow for arbitrary likelihood scenarios (e.g., an anytime bound such as the goal of generating a FSC where >20% of the paths reach the goal state). Perhaps one could think of the difference between LAO* and LRTDP vs. PANDOR as being analogous to the difference between the paradigms of dynamic programming vs. Monte Carlo methods in reinforcement learning [35]. With dynamic programming, incremental changes are made to the policy of the controller (sometimes based on the entire state space), whereas with Monte Carlo methods, sample trajectories are used to evaluate the performance of the current policy. In fact, we think our proposal is suitable to combine the best of the two worlds, but that is a topic for future research. We are also excited about the prospect of extending our contributions to the continuous case, and potentially providing asymptotic guarantees for provably correct FSC synthesis.

8. Conclusions

In this paper, we presented new theoretical results on a generic technique for synthesizing FSCs in stochastic environments, allowing for highly granular specifications on termination and goal satisfaction. We then proved the soundness and completeness of that synthesis algorithm.

There are many interesting directions for the future. For example, one could investigate: *(a)* effective sampling strategies; *(b)* the tradeoff between higher **LGT** bounds vs scalability (i.e., demands on **LGT** bounds may be different

[4]Variant additional stipulations in the literature include things like *fairness*, where every outcome of a nondeterministic action must occur infinitely often [6].

across applications); and *(c)* the merits and demerits of the various correctness criteria from the literature for safety-critical applications. To that end, along with recent advances in the area, we hope that our results provide theoretical foundations, new proof strategies and a fresh perspective on FSC synthesis in stochastic domains.

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Solving POMDPs using quadratically constrained linear programs. In *IJCAI*, pages 2418–2424, 2007.

[2] V. Belle and H. J. Levesque. Foundations for generalized planning in unbounded stochastic domains. In *Proc. KR 2016*, pages 380–389. AAAI Press, 2016.

[3] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170(4–5):337 – 384, 2006. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2006.01.004. URL `http://www.sciencedirect.com/science/article/pii/S0004370206000075`.

[4] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *AIPS*, pages 52–61. AAAI, 2000.

[5] B. Bonet and H. Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, volume 3, pages 12–21, 2003.

[6] B. Bonet and H. Geffner. Policies that generalize: Solving many planning problems with the same policy. In *IJCAI*, 2015. URL `http://ijcai.org/papers15/Abstracts/IJCAI15-396.html`.

[7] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS 2009*. AAAI Press, 2009. URL `http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/752`.

[8] S. J. Celorrio, J. S. Aguas, and A. Jonsson. A review of generalized planning. *Knowledge Eng. Review*, 34:e5, 2019. doi: 10.1017/S0269888918000231. URL `https://doi.org/10.1017/S0269888918000231`.

[9] K. Chatterjee, M. Chmelík, and J. Davies. A symbolic sat-based algorithm for almost-sure reachability with small strategies in pomdps. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[10] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147 (1-2):35–84, 2003. doi: 10.1016/S0004-3702(02)00374-0.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969. doi: 10.1145/363235.363259. URL `http://doi.acm.org/10.1145/363235.363259`.

[12] Y. Hu and G. De Giacomo. A generic technique for synthesizing bounded finite-state controllers. In *Proc. ICAPS 2013*. AAAI, 2013.

[13] Y. Hu and H. Levesque. Planning with loops: Some new results. In *Workshop on Generalized Planning, ICAPS 2009*, 2009.

[14] Y. Hu and H. J. Levesque. A correctness result for reasoning about one-dimensional planning problems. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011*, pages 2638–2643. IJCAI/AAAI, 2011.

[15] A. Ireland and J. Stark. Combining proof plans with partial order planning for imperative program synthesis. *Automated Software Engineering*, 13(1):65–105, 2006.

[16] S. Jiménez, J. Segovia-Aguas, and A. Jonsson. A review of generalized planning. *The Knowledge Engineering Review*, 34:e5, 2019. doi: 10.1017/S0269888918000231.

[17] S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J. Katoen, and B. Becker. Finite-state controllers of pomdps via parameter synthesis. In *Proceedings of the UAI*, 2018.

[18] H. J. Levesque. What is planning in the presence of sensing? In *Proc. AAAI / IAAI*, pages 1139–1146, 1996.

[19] H. J. Levesque. Planning with loops. In *IJCAI-05*, pages 509–515. Professional Book Center, 2005.

[20] F. Lin and H. J. Levesque. What robots can do: robot programs and effective achievability. *Artificial Intelligence*, 101(1-2):201–226, 1998.

[21] I. Little and S. Thiébaux. Probabilistic planning vs. replanning. *Workshop, ICAPS 2007*, 2007.

[22] M. Magnusson and P. Doherty. Deductive planning with inductive loops. In *KR*, pages 528–534, 2008. URL `http://www.aaai.org/Library/KR/2008/kr08-051.php`.

[23] Z. Manna and R. J. Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, 1980. doi: 10.1145/357084.357090. URL `http://doi.acm.org/10.1145/357084.357090`.

[24] M. J. Matarić. *The robotics primer*. MIT Press, 2007.

[25] G. H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, Sept. 1955. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1955.tb03788.x.

[26] N. Meuleau, K. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving pomdps by searching the space of finite policies. In *UAI*, pages 417–426. Morgan Kaufmann, 1999.

[27] F. Patrizi, N. Lipovetzky, G. De Giacomo, and H. Geffner. Computing infinite plans for LTL goals using a classical planner. In T. Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2003–2008. IJCAI/AAAI, 2011. ISBN 978-1-57735-516-8. doi: 10.5591/978-1-57735-516-8/IJCAI11-334. URL `https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-334`.

[28] F. Patrizi, N. Lipovetzky, and H. Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 2343–2349. IJCAI/AAAI, 2013. ISBN 978-1-57735-633-2. URL `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6653`.

[29] P. Poupart and C. Boutilier. Bounded finite state controllers. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *NIPS*, pages 823–830. MIT Press, 2003. URL `http://papers.nips.cc/paper/2372-bounded-finite-state-controllers`.

[30] C. Pralet, G. Verfaillie, M. Lemaître, and G. Infantes. Constraint-based controller synthesis in non-deterministic and partially observable domains. In *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 681–686. IOS Press, 2010.

[31] U. Schmid and F. Wysotzki. Applying inductive program synthesis to macro learning. In *AIPS*, pages 371–378, 2000.

[32] S. Srivastava. *Foundations and Applications of Generalized Planning*. PhD thesis, University of Massachusetts Amherst, 2010.

[33] S. Srivastava, S. Zilberstein, A. Gupta, P. Abbeel, and S. J. Russell. Tractability of planning with loops. In *Proc. AAAI*, pages 3393–3401, 2015. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9754`.

[34] W. Stephan and S. Biundo. Deduction-based refinement planning. In *AIPS*, pages 213–220, 1996. URL `http://www.aaai.org/Library/AIPS/1996/aips96-027.php`.

[35] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (second edition)*. The MIT Press, 2018.

[36] E. Winner and M. M. Veloso. LoopDISTILL: Learning domain-specific planners from example plans. In *Workshop on AI Planning and Learning, ICAPS*, 2007.