Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Stochastic Fairness and Language-Theoretic Fairness in Planning in Nondeterministic Domains

Benjamin Aminof
TU Wien, Austria
aminof@forsyte.at

Giuseppe De Giacomo
Univ. Roma "La Sapienza", Italy
degiacomo@diag.uniroma1.it

Sasha Rubin
Univ. Sydney, Australia
sasha.rubin@sydney.edu.au

Abstract

We address two central notions of fairness in the literature of nondeterministic fully observable domains. The first, which we call stochastic fairness, is classical, and assumes an environment which operates probabilistically using possibly unknown probabilities. The second, which is language-theoretic, assumes that if an action is taken from a given state infinitely often then all its possible outcomes should appear infinitely often; we call this state-action fairness. While the two notions coincide for standard reachability goals, they differ for temporally extended goals. This important difference has been overlooked in the planning literature and has led to the use of a product-based reduction in a number of published algorithms which were stated for state-action fairness, for which they are incorrect, while being correct for stochastic fairness. We remedy this and provide a correct optimal algorithm for solving state-action fair planning for LTL/LTL$_f$ goals, as well as a correct proof of the lower bound of the goal-complexity. Our proof is general enough that it also provides, for the no-fairness and stochastic-fairness cases, multiple missing lower bounds and new proofs of known lower bounds. Overall, we show that stochastic fairness is better behaved than state-action fairness.

1 Introduction

Nondeterminism in planning captures uncertainty that the agent has at planning time about the effects of its actions. For instance, "remove block A from the table" may either succeed, resulting in "block A is not on the table", or fail, resulting in "block A is on the table". Plans in nondeterministic environments are not simply sequences of actions as in classical planning; rather, the next action may depend on the sequences of actions (and observations[1]) so far, and are captured by policies (also known as strategies and controllers).

Broadly speaking, nondeterminism manifests in one of two ways: stochastic and adversarial environments.

Stochastic Environments Nondeterministic environments with probabilities are often modeled in planning as Markov Decision Processes (MDPs), i.e., as state-transition systems in which the probability of an effect depends only on the current state and action. However, sometimes the probabilities of action effects are not available, non stationary, or hard to estimate, e.g., an agent may encounter an unexpected obstacle, or an exogenous event or failure occurs. A long thread in the literature aims to understand what it means to plan in such environments (Daniele, Traverso, and Vardi 1999; Pistore and Traverso 2001; Cimatti et al. 2003; D'Ippolito, Rodríguez, and Sardiña 2018). One common intuition is that the goal should be achievable by trial-and-error, expecting only a finite amount of bad luck: e.g., repeating "remove block A from the table" should eventually succeed. This amounts to assuming that some unknown distribution assigns a nonzero probability to each of the alternative effects. Thus, although there are no explicit probabilities, the stochastic principle is still in place. We call such assumptions *stochastic fairness*. Plans in this setting are called strong-cyclic, and their importance is evidenced by the existence of several tools for finding them, e.g., NDP (Alford et al. 2014), FIP (Fu et al. 2016), myND (Mattmüller et al. 2010), Gamer (Kissmann and Edelkamp 2011), PRP (Muise, McIlraith, and Beck 2012), GRENADE (Ramírez and Sardiña 2014), and FOND-SAT (Geffner and Geffner 2018). Such policies ensure the goal with probability 1 (Geffner and Bonet 2013).

Adversarial Environments Nondeterministic environments without probabilities are often modeled as fully observable nondeterministic planning domains (FOND). These are state-transition systems in which the effect of an action is a set of possible states, rather than a single state as in classical planning. Policies that guarantee success, i.e., the goal is achieved no matter how the nondeterminism is resolved, are called strong solutions. Under adversarial nondeterminism it is often reasonable to require that a policy should guarantee success under some additional assumptions about the environment. For instance, a typical assumption is that repeating an action in a given state results in all possible effects, e.g., repeating the action "remove block A from the table" would eventually succeed (and eventually fail). Note that this can be expressed as a property of traces, and so for the purpose of this paper, we call such notions *language-theoretic fairness*. We focus on one central such notion which we call *state-action fairness* and which says, of a trace, that if an

[1] In this paper we assume there is no uncertainty about the current state of the system, i.e., environments are fully observable.

action a is taken from a state s infinitely often in the trace, and if s' is a possible effect of a from s, then infinitely often in the trace s' is the resulting effect of action a from state s. Although there are many notions of fairness, this particular notion has been identified as providing sufficient assumptions that guarantee the success of solutions that repeatedly retry (D'Ippolito, Rodríguez, and Sardiña 2018).

What is the relationship between fairness in an adversarial setting and fairness in a stochastic setting? Planning assuming either notion of fairness means that the policy can ignore some traces, which are guaranteed not to be produced by the environment. Also, it turns out that when planning for reachability goals (i.e., eventually reach a certain target set of states) the two notions of fairness are interchangeable. More precisely, a policy achieves the reachability goal assuming stochastic fairness (i.e., it is a strong-cyclic solution) iff it achieves the reachability goal assuming state-action fairness (i.e., the target set is reached on all state-action fair traces). On the other hand, it turns out that the two notions of fairness are not interchangeable in the context of planning for temporally extended goals, such as those expressed in linear temporal logic LTL or its finite-trace variant LTL_f.

Outline of the Paper and Contributions In Section 3 we point out the distinction between stochastic fairness and state-action fairness in the context of planning. Once this distinction has been noted, in Section 4 we analyze a product-based reduction for solving fair planning problems that has often been used in the literature. We show that this technique is correct for stochastic fairness but not for state action fairness. As a result, there is currently no published correct algorithm for handling state-action fairness for temporally extended goals. We provide one in Section 5, as well as complexity upper bounds for the problem: 2EXPTIME for combined and goal complexity, and 1NEXPTIME for domain complexity. In Section 6 we provide a proof of the matching 2EXPTIME lower bound for combined and goal complexity. Our proof is general enough that it also provides, for the no-fairness and stochastic-fairness cases, multiple missing lower bounds and new proofs of known lower bounds. Domain complexity is 1EXPTIME-hard already for reachability goals, leaving a gap between deterministic and nondeterministic exponential time. In Section 7 we discuss related work in the verification literature where various notions of fairness have been studied in a different context, and the difference between stochastic fairness and language-theoretic fairness such as state-action fairness had not been overlooked.

2 Fair Planning Problems

In this section we define planning domains, temporally extended goals, and isolate the two notions of fairness.

Planning Domains A *nondeterministic planning domain* is a tuple (St, Act, s_0, Tr) where St is a finite set of *states*, s_0 is an *initial state*, Act is a finite set of *actions*, and $Tr \subseteq St \times Act \times St$ is a *transition relation*. We will sometimes write Tr in functional form, i.e., $Tr(s, a) \subseteq St$. We say that the action a is *applicable* in state s if $Tr(s, a) \neq \emptyset$. We assume, by adding a dummy action and state if needed, that for every state there is an applicable action.

For a finite set X let $Dbn(X)$ denote the set of *(probability) distributions* over X, i.e., functions $d : X \to [0, 1]$ such that $\sum_{x \in X} d(x) = 1$. An element x is in the *support* of d if $d(x) > 0$. A *stochastic planning domain* is a tuple (D, Pr) where $D = (St, Act, s_0, Tr)$ is the *induced* nondeterministic planning domain, and Pr, called the *probabilistic transition function*, is a partial function $Pr : St \times Act \to Dbn(St)$ defined only for pairs (s, a) where a is applicable in s, satisfying that the support of $Pr(s, a)$ is equal to $Tr(s, a)$. Stochastic domains are variants of Markov Decision Processes (MDPs). However, MDPs typically have Markovian rewards, while stochastic planning problems may have goals that depend on the history.

We will refer to both nondeterministic and stochastic planning domains simply as *domains*. Unless otherwise stated, *domains are compactly represented*, e.g., in variants of the Planning Domain Description Language (PDDL), and thus can usually be represented with a number of bits which is polylogarithmic in the number of states and actions. In particular, the states are encoded as assignments to Boolean variables $\mathcal{F}$ called *fluents*, and thus: $St = 2^{\mathcal{F}}$. For symmetry, also the actions are encoded as assignments to Boolean variables $\mathcal{A}$ that are disjoint from $\mathcal{F}$, and thus: $Act = 2^{\mathcal{A}}$. Although the literature also contains formalisms for compactly representing stochastic domains (such as Probabilistic PDDL), we will not be concerned with a detailed formalization of probabilistic transition functions since it is known (as we later discuss) that they essentially play no role in the stochastic-fair planning problem (formally defined below).

Traces and Policies Let D be a domain. A *trace* τ of D is a finite or infinite sequence $(s_0 \cup a_0)(s_1 \cup a_1) \cdots$ over the alphabet $(St \cup Act) = 2^{\mathcal{F} \cup \mathcal{A}}$ where s_0 is the initial state, and $(s_{i-1}, a_{i-1}, s_i) \in Tr$ for all i with $1 \leq i < |\tau|$, where $|\tau| \in \mathbb{N} \cup \{\infty\}$ is the length of τ. Note that for explicitly represented domains we sometimes find it convenient to write $\tau = (s_0, a_0)(s_1, a_1) \cdots$. The sequence $s_0 s_1 \cdots$ of states is called the *path* induced by τ. A transition (s, a, s') is *enabled* at position i of trace τ if $s_i = s$ and $a_i = a$. The transition is then *taken* if $s_{i+1} = s'$. A *policy* is a function $f : (St)^+ \to Act$ such that for every $u \in (St)^+$ the action $f(u)$ is applicable in the last state of u. Here $(St)^+$ is the set of finite non-empty sequences of states. Note that policies are history dependent in this paper. A trace τ is *generated by* f, and simply called an *f-trace*, if every finite prefix $(s_0 \cup a_0) \cdots (s_i \cup a_i)$ of τ satisfies $f(s_0 s_1 \cdots s_i) = a_i$. A *finite-state policy* is a policy that can be represented as a finite-state input/output automaton that, on reading $u \in (St)^+$ as input, outputs the action $f(u)$. As usual, a stochastic domain D combined with a policy f induces a (possibly infinite-state) Markov chain, denoted (D, f), which gives rise to a probability distribution over the set of infinite f-traces in D (Vardi 1985).

Example 1 The following domain will be used in counterexamples. Let $D = (\{l, m, r\}, \{a\}, l, Tr)$ be a domain where Tr consist of the triples $(l, a, m), (m, a, l), (m, a, r)$ and (r, a, m). Note that there is only one policy available: always do the action a. Define the trace τ as

$((l,a)(m,a)(r,a)(m,a))^\omega$.[2] Note that this trace takes each of the transitions (m,a,r) and (m,a,l) infinitely often.

Linear Temporal Logic Linear Temporal Logic (LTL) is a formalism that was introduced into the verification literature for describing computations of programs without the use of explicit time stamps. The logic has since been used in planning as a language for specifying temporally extended goals and for expressing search control, see, e.g., Fainekos, Kress-Gazit, and Pappas (2005), Bacchus and Kabanza (2000).

The syntax of LTL consists of atoms AP and is closed under the Boolean operations $\neg$ and $\wedge$ and the temporal operators $\bigcirc$ (read "next") and $\mathcal{U}$ (read "until"):

$$\psi ::= p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \bigcirc\psi \mid \psi_1 \,\mathcal{U}\, \psi_2$$

with p varying over the elements of AP.

We use the usual short-hands, e.g., $\texttt{false} := p \wedge \neg p$, $\psi_1 \supset \psi_2 := \neg\psi_1 \vee \psi_2$, $\Diamond\psi := \texttt{true}\,\mathcal{U}\,\psi$ (read "eventually ψ"), and $\Box\psi := \neg\Diamond\neg\psi$ (read "always ψ").

Formulas of LTL are interpreted over infinite sequences $\tau = \tau_0\tau_1\cdots$ over the alphabet 2^{AP}. Define $\tau, j \models \psi$ inductively on the structure of ψ, simultaneously for all time points $j \geq 0$, as follows:

- $\tau, j \models p$ if $p \in \tau_j$,
- $\tau, j \models \psi_1 \wedge \psi_2$ if $\tau, j \models \psi_i$ for $i = 1, 2$,
- $\tau, j \models \bigcirc\psi$ if $\tau, j+1 \models \psi$,
- $\tau, j \models \psi_1 \,\mathcal{U}\, \psi_2$ if $\tau, k \models \psi_2$ for some $k \geq j$, and $\tau, i \models \psi_1$ for all $j \leq i < k$.

We also consider the variant LTL_f of LTL interpreted over finite sequences. It has the same syntax and semantics as LTL except that τ is a finite sequence and that one defines $\bigcirc$ as follows (cf. Bacchus and Kabanza (2000), Baier and McIlraith (2006), De Giacomo and Vardi (2013)): $\tau, j \models \bigcirc\psi$ if $j < |\tau| - 1$ and $\tau, j+1 \models \psi$ (recall that the last position on τ is $|\tau| - 1$ since the first one is 0).

If ψ is an LTL (resp. LTL_f) formula and τ is an infinite (resp. finite) sequence over AP, we write $\tau \models \psi$, and say that τ *satisfies* ψ, if $\tau, 0 \models \psi$.

We also make the following convention for interpreting LTL_f formulas over infinite traces: if τ is infinite and ψ is an LTL_f formula, then $\tau \models \psi$ means that some finite prefix of τ satisfies ψ. We remark that this existential quantification is implicitly under the control of the agent determining the policy, and is analogous to the use of an explicit "stop" action, or to using partial policies, as done elsewhere.

In the context of a planning domain D, we take $AP = \mathcal{F} \cup \mathcal{A}$ (this is for convenience; some papers take $AP = \mathcal{F}$). Given a domain D and a policy f, we say that f *enforces* ψ, and write $(D, f) \models A\psi$, if every infinite f-trace satisfies ψ.

Planning Problems A *goal* G is a set of infinite traces of D. A *planning problem* $\langle D, G\rangle$ consists of a domain D and a goal G. *Solving* the planning problem is to decide, given D (compactly represented) and G (suitably represented), if there is a policy f such that every infinite f-trace satisfies G (i.e., is in G). In this paper, goals will typically be represented by LTL/LTL_f formulas.

[2] For a finite string u, we write u^ω for the infinite string $uuu\ldots$.

Fair Planning Problems We define the fair planning problems mentioned in the introduction. A trace τ is *state-action fair* if every transition enabled in τ infinitely often is taken in τ infinitely often. This is expressible in LTL:

$$\phi_{D,fair} := \bigwedge_{(s,a,s')\in Tr} (\Box\Diamond(s \wedge a) \supset \Box\Diamond(s \wedge a \wedge \bigcirc s'))\,.$$

A policy f solves the *state-action fair planning problem* $\langle D, \psi\rangle$, written $(D, f) \models A^{\text{sa-fair}}\psi$, if every state-action fair f-trace satisfies ψ.

For a stochastic domain D, we write $(D, f) \models A^{=1}\psi$ to mean that the probability that an f-trace satisfies ψ is equal to 1, and we say that f *almost surely enforces* ψ. It is known that $(D, f) \models A^{=1}\psi$ does not depend on the probabilistic transition function of D, but only on its induced nondeterministic domain; i.e., it only depends on the supports of the distributions $Pr(s, a)$, which are specified by the transition relation Tr of the induced nondeterministic domain (cf. Vardi and Wolper (1986)). Hence, we can actually extend this probabilistic notion of enforcing also to nondeterministic domains, as follows. For a nondeterministic domain D, we write $(D, f) \models A^{=1}\psi$ to mean that $(D', f) \models A^{=1}\psi$ where D' is any stochastic domain whose induced nondeterministic domain is D. Thus, for a domain D (nondeterministic or stochastic), we say that f solves the *stochastic-fair planning problem* $\langle D, \psi\rangle$ if $(D, f) \models A^{=1}\psi$.

Planning under either notion of fairness allows the agent to ignore some traces: in the language-theoretic setting it can ignore exactly the set of traces that are not state-action fair, while in the stochastic setting it can ignore any set of traces whose probability measure is zero. Since the set of traces that are not state-action fair has probability 0 (Vardi and Wolper 1986) and, furthermore, there are many natural supersets of it that also have probability 0, stochastic fairness allows the agent much more freedom with respect to the traces it can ignore. This fact is at the heart of why stochastic fairness is more well-behaved, and is used in the next section where we clarify the distinction between the two notions of fairness in the context of planning.

Connection with Planning for Reachability Goals The classic goal in planning is reachability, typically represented as a Boolean combination $target$ of fluents, i.e., it can be expressed by an LTL/LTL_f formula $\Diamond target$. A policy enforcing $\Diamond target$ is known as a *strong solution*, and a policy enforcing $\Diamond target$ assuming state-action fairness is known as a *strong cyclic solution* (Cimatti et al. 2003).

Computational Complexity Planning problems have two inputs: the domain (represented compactly) and the goal (typically represented as a formula). *Combined complexity* measures the complexity in terms of the size of both inputs. *Goal complexity* (resp. *domain complexity*) only measures the complexity in the size of the goal (resp. domain). Formally, the *goal complexity is in a complexity class* $\mathcal{C}$ if for every domain D, the complexity of the problem, that takes as input a goal ψ and decides if there is a solution to the planning problem $\langle D, \psi\rangle$, is in $\mathcal{C}$; the *goal complexity is hard for* $\mathcal{C}$ if there is a domain D for which the above

problem is $\mathcal{C}$-hard. Domain complexity is defined symmetrically. Such measures were first introduced in database theory (Vardi 1982).

Automata-theoretic Approach to Planning A typical approach for solving planning problems with temporally-extended goals is to use an automata-theoretic approach. Here we recall just enough for our needs in Sections 4 and 5.

A *deterministic automaton* is a tuple $M = (\Sigma, Q, q_0, \delta, \mathcal{C})$ where Σ is the *input alphabet*, Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $\mathcal{C}$ is the *acceptance condition* (described later). A (finite or infinite) input word $u = u_0 u_1 \cdots$ determines a run, i.e., the sequence $q_0 q_1 \cdots$ of states starting with the initial states and respecting the transition function, i.e., $\delta(q_{i-1}, u_{i-1}) = q_i$ for all $1 \leq i < |u|$. M *accepts* a word if its run on it is accepting, i.e., satisfies the acceptance condition $\mathcal{C}$. If M is to accept only finite words, then one typically has $\mathcal{C} \subseteq Q$; and we say that a finite run satisfies $\mathcal{C}$ if its last state is in $\mathcal{C}$. Such an automaton is called a *deterministic finite word automaton (DFW)*. If M is to accept only infinite words, then there are a number choices for $\mathcal{C}$. We will not be concerned with the specific choice until Section 5.

The *synchronous product* of a domain D and a deterministic automaton A over the input alphabet $2^{\mathcal{F} \cup \mathcal{A}}$ is a domain, denoted $D \times A$, whose states are pairs (d, q) where d is a state of D and q is a state of A, and that can transition from state (d, q) to state (d', q') on action a if $(d, a, d') \in Tr$ and the automaton can go from q reading $d \cup a$ to q'. Intuitively, $D \times A$ simulates both D and A simultaneously. Such products are used in algorithms for planning with LTL/LTL$_f$ goals in Section 4 and Section 5. We remark that the product is sometimes also compactly represented, although the details depend on the context and will not concern us.

3 Stochastic Fairness $\not\equiv$ State-action Fairness

We now compare the two notions of fairness in the context of planning. It turns out that they are equivalent for reachability goals, but not for all LTL/LTL$_f$ goals. The first fact is known, e.g. Rintanen (2004), and is repeated here for completeness.

Proposition 1 *Let D be a (nondeterministic or stochastic) domain and let $target$ be a Boolean combination of fluents. The following are equivalent for every finite-state policy f:*

1. $(D, f) \models A^{\text{sa-fair}}(\Diamond target)$*, i.e., the target is reached on state-action fair traces.*
2. $(D, f) \models A^{=1}(\Diamond target)$*, i.e., the target is reached with probability* 1.

Proof. Assume that $(D, f) \models A^{\text{sa-fair}}(\Diamond target)$. Observe that the set of state-action fair traces has probability 1, cf. (Vardi and Wolper 1986), and thus, by definition, $(D, f) \models A^{=1}(\Diamond target)$. For the other direction, assume by way of contradiction that 2. holds but 1. doesn't, and pick an infinite state-action fair f-trace τ that does not satisfy $\Diamond target$. Let M be the finite-state Markov chain induced by D and f, viewed as a directed graph, and let π be the path in M induced by τ. Since τ is state-action fair, π reaches a bottom strongly connected component C of M, and visits every state in C. By the assumption that $\tau \not\models \Diamond target$, π contains no state in which $target$ holds. Let ρ be some (fixed) prefix of π that ends in a state in C, and consider the set E of infinite f-traces whose induced paths have ρ as a prefix. Observe that the probability of E is positive, and that none of the traces in E satisfy $\Diamond target$, contradicting 2. ∎

We now turn to goals expressed as LTL/LTL$_f$ formulas. Unfortunately, in this case the analogue of Proposition 1 does not hold. Indeed, only the forward direction holds.

Proposition 2 *Let D be a domain, ψ an LTL/LTL$_f$ formula, and f a finite-state policy. If $(D, f) \models A^{\text{sa-fair}}(\psi)$ then $(D, f) \models A^{=1}(\psi)$.*

Proof. As in Proposition 1, simply use the fact that the set of infinite state-action fair f-traces has probability 1. ∎

The converse of Proposition 2 does not hold:

Proposition 3 *There is a domain D, a finite-state policy f, and an LTL$_f$ goal ψ such that $(D, f) \models A^{=1}(\psi)$, but for no policy g does it hold that $(D, g) \models A^{\text{sa-fair}}(\psi)$.*

Proof. Let D be the domain from Example 1. Let ψ be the LTL/LTL$_f$ formula $\Diamond(l \wedge \bigcirc \bigcirc l)$ (i.e., eventually l and two steps afterwards l again). There is only one policy f available: it always chooses the action a. Observe that $(D, f) \models A^{=1}(\psi)$, but that $(D, f) \not\models A^{\text{sa-fair}}(\psi)$ as witnessed by the state-action fair trace $\tau := (\{l, a\}\{m, a\}\{r, a\}\{m, a\})^\omega$. ∎

4 Analysis of Product-based Reductions for Solving Fair Planning

A natural approach for solving a planning problem with a temporally extended goal is to reduce it to solving a planning problem on a domain that simultaneously simulates the original domain and a deterministic automaton for the goal formula, and whose goal reflects the acceptance condition of the automaton. We show that this reduction is correct for handling stochastic fairness but not for state-action fairness.

The Product Reduction We begin by describing the reduction with no mention of fairness. Given a planning problem $\langle D, \psi \rangle$, take a deterministic automaton A_ψ that recognizes exactly the traces that satisfy ψ. Second, define the domain $D' = D \times A_\psi$ as the synchronous product of D and A_ψ (see end of Section 2). Finally, define the planning problem $\langle D', Acc \rangle$ where Acc is a goal consisting of those traces of D' whose second component is an accepting run of A_ψ.

Analysis of the Reduction If this reduction is to be used to give an exact algorithm for planning assuming state-action fairness, it should be sound and complete, i.e., $\langle D, \psi \rangle$ is solvable assuming state-action fairness iff $\langle D', Acc \rangle$ is solvable assuming state-action fairness. The reduction is indeed complete because every state-action fair trace in the product domain D' projects to a state-action fair trace in D (this follows immediately from the definition of state-action fairness and of the synchronous product). On the other hand, the reduction is not sound because there may be fair traces in D that do not induce any fair trace in D' (intuitively, this is due to synchronization in D' between the domain D and

the automaton A_ψ). We formalise this in the following theorem which actually shows that the reduction is not sound no matter which deterministic automaton A_ψ for ψ is used (we mention that the ψ in the Theorem can easily be represented by a deterministic automaton with very simple acceptance conditions, e.g., Büchi acceptance condition).

Theorem 1 *There is a domain D, and an* LTL/LTL$_f$ *goal ψ, s.t.: a) there is no solution to the state-action fair planning problem $\langle D, \psi \rangle$, but b) for every deterministic automaton A_ψ accepting exactly the traces that satisfy ψ, there is a solution to the state-action fair planning problem $\langle D', Acc \rangle$, where D' is the product of D and A_ψ, and Acc captures the acceptance condition of A_ψ.*

Proof. Let D and τ be the domain and trace from Example 1. Recall that there is a single policy f ('always do a'), and that τ is an f-trace. Let ψ be the formula $\neg l \vee \Diamond(l \wedge \bigcirc\bigcirc\neg r) \vee \Diamond(l \wedge \bigcirc\bigcirc\bigcirc\bigcirc\neg l)$. Note that all traces of D, except τ, satisfy ψ. Since τ is state-action fair, there is no solution to the state-action fair problem $\langle D, \psi \rangle$.

We claim that the policy f is a solution to $\langle D', Acc \rangle$. For this, it is enough to show that every state-action fair trace in D' induces in D a trace that satisfies ψ, i.e., a trace other than τ. Let τ' be a trace in D' that induces τ. To see that τ' is not state-action fair, let (m, s) be a state that appears in τ' infinitely often after a state of the form $(l, ?)$. Note that (m, s) never appears as a source of a transition to a state of the form $(l, ?)$. Indeed, since l occurs on τ exactly every four steps, the source of such a transition is only reached three steps after reading an l; and while reading τ, A_ψ is always in a different state than s three steps after reading an l (so not to confuse occurrences of $\neg l$ four steps after an l with ones two steps after it). Thus, the transition $((m, s), a, (l, q))$ is enabled infinitely often but never taken. ∎

Note, however, that if one uses stochastic fairness instead of state-action fairness then the reduction above is sound and complete. This is because stochastic-fairness is preserved by taking a product with deterministic automata, a fact which is used in the automata theoretic approach to verification (Vardi 1985; Courcoubetis and Yannakakis 1995; Bianco and de Alfaro 1995; Bollig and Leucker 2004).

Theorem 2 *Let $\langle D, \psi \rangle$ be a planning problem, and let $\langle D', Acc \rangle$ be a planning problem obtained from it by the product reduction. There is a policy solving $\langle D, \psi \rangle$ assuming stochastic fairness iff there is a policy solving $\langle D', Acc \rangle$ assuming stochastic fairness.*

Proof. [Sketch] Turn D' into a stochastic domain by defining the probability $Pr_{D'}((s, q), a)(s', q')$ to equal $Pr_D(s, a)(s')$ if $(q, s \cup a, q')$ is a transition of the automaton, and 0 otherwise. There is a natural bijection between traces $\pi' \mapsto \pi$ (resp. sets of traces $X' \mapsto X$, resp. policies $f' \mapsto f$) in D' and those in D, i.e., projection onto the domain component. Moreover, the probability in D' that an f'-trace is in X' is equal to the probability in D that an f-trace is in X (this follows from the fact that this property is true on cones, and thus on all measurable sets). Thus $(D, Pr_D) \models A^{=1}(\psi)$ iff $(D', Pr_{D'}) \models A^{=1}(Acc)$. ∎

Unfortunately, this product-based reduction, has been incorrectly used for state-action fairness, e.g., Patrizi, Lipovetzky, and Geffner (2013)[Theorem 3], De Giacomo and Rubin (2018)[Theorem 4] and Camacho and McIlraith (2019)[Theorem 2]. Our analysis shows this reduction is correct for stochastic fairness but not for state-action fairness, so these theorems hold only if stochastic fairness (rather than state-action fairness) is assumed.

One may conjecture that some confusion in the proofs and algorithms for state-action fair planning in the literature arise from the mistaken intuition that state-action fairness always behaves like stochastic fairness, which, as we show, it does not in the presence of even simple LTL/LTL$_f$ formulas (that are not reachability formulas).

5 Algorithm for State-action Fair Planning

As discussed in Section 4, previous literature does not provide correct algorithms for state-action fair planning for temporally extended goals. In this section we provide a correct algorithm, using a sound and complete reduction to the problem of solving *Rabin games* (defined below), and obtain the following theorem:

Theorem 3 *The combined (and thus goal) complexity of solving planning with* LTL/LTL$_f$ *goals assuming state-action fairness is in* 2EXPTIME*, and the domain complexity is in* 1NEXPTIME *(in the size of a compactly represented domain).*

The main approach to solving such a problem is to use, explicitly or implicitly, an automata theoretic approach. However, as we now remark, naive applications of this approach yield a 3EXPTIME domain complexity (which we lower to 1NEXPTIME), a 3EXPTIME combined complexity (which we lower to 2EXPTIME), and a 2EXPTIME goal complexity.

Remark 1 *Solving the state-action fair planning problem $\langle D, \psi \rangle$ where ψ is an* LTL/LTL$_f$ *formula is equivalent to solving the planning problem $\langle D, \phi_{D,fair} \supset \psi \rangle$ where $\phi_{D,fair}$ is an* LTL *formula (given in Section 2) expressing state-action fairness in the domain D; for more on this equivalence see Aminof et al. (2019a). However, the size of $\phi_{D,fair}$ is* exponential *in the size of D (when compactly represented). i.e., this reduces the problem to solving planning for an* LTL *goal of size exponential in the size of D and linear in the size of ψ. In turn, there are algorithms that solve planning with* LTL *goals (no fairness assumptions) that run in* 1EXPTIME *in the size of the domain and* 2EXPTIME *in the size of the goal (Aminof et al. 2019a; Camacho, Bienvenu, and McIlraith 2019). Putting this together results in an algorithm for state-action fair planning problems that runs in* 3EXPTIME *in the size of the domain D and* 2EXPTIME *in the size of the original formula ψ.*

The main insight in Theorem 3 is that one should use Rabin conditions. A *Rabin condition* over a set X is a set $\mathcal{R}$ of pairs of the form (I, F) with $I, F \subseteq X$. The pairs are called *Rabin pairs*. An infinite sequence τ over the alphabet X *satisfies the Rabin condition* $\mathcal{R}$ if there is a pair $(I, F) \in X$ such that some $x \in I$ appears infinitely often in τ and no

$x \in F$ appears infinitely often in τ.[3] We chose the Rabin condition, instead of some other condition, since it can capture very general properties, including LTL/LTL$_f$; it is trivially closed under union; and it can naturally express that a trace is not state-action fair. Below we use Rabin conditions in two ways: as acceptance conditions (for automata) and as winning conditions (in games).

Rabin Automata A *Deterministic Rabin Word (DRW) automaton* $M = (\Sigma, Q, q_0, \delta, \mathcal{R})$ has as its acceptance condition $\mathcal{R}$ a Rabin condition over Q. Its *size* is the number of its states and its *index* is the number of pairs in $\mathcal{R}$.

Theorem 4 *[cf. Vardi (1995)] Given an* LTL/LTL$_f$ *formula ψ one can build a DRW M_ψ accepting exactly the infinite traces satisfying ψ.*[4] *The size (resp. index) of M_ψ is at most doubly (resp. singly) exponentially larger than the size of ψ.*

Lemma 1 *Given a domain D one can build a DRW $M_{D,unfair}$ that accepts exactly the infinite traces of D that are not state-action fair. Moreover, the size and index of $M_{D,unfair}$ is at most singly exponentially larger than size of D (compactly represented).*

Proof. Intuitively, the states of the DRW stores the last state-action pair of D, the transition function ensures that only valid transitions of the domain are taken (otherwise the automaton goes to a fail state f), and the Rabin pairs ensure that the trace of D is unfair. Formally, given domain $D = (St, Act, s_0, Tr)$, define the DRW $(\Sigma, Q, q_0, \delta, \mathcal{R})$ with alphabet $\Sigma = St \times Act$; states $Q = St \times Act \cup \{q_0, f\}$; and the transition function defined by $\delta(q_0, (s_0, a)) = (s_0, a)$ if a is applicable in s_0, by $\delta((s, a), (s', a')) = (s', a')$ if $(s, a, s') \in Tr$ and a' is applicable in s', and by $\delta(q, \sigma) = f$ in all other cases; and the Rabin condition $\mathcal{R}$ containing all pairs $(\{(s, a)\}, \{(s', a')\})$ s.t. $(s', a') = \delta((s, a), (s', a'))$. ∎

Lemma 2 *Given DRW M_1, M_2 one can build a DRW M, denoted $M_1 \vee M_2$, that accepts the words accepted by M_1 or M_2. The size of M is the product of the sizes of the M_is, and the index of M is the sum of the indices of the M_is.*

Proof. Let $M_i = (\Sigma, Q_i, q_i, \delta_i, \mathcal{R}_i)$, and define $M = (\Sigma, Q_1 \times Q_2, (q_1, q_2), \delta', \mathcal{R}')$ where $\delta'((s_1, s_2), \sigma) = (\delta_1(s_1, \sigma), \delta_2(s_2, \sigma))$, and $\mathcal{R}$ consists of all pairs of the form $(Q_1 \times I, Q_1 \times F)$ for $(I, F) \in \mathcal{R}_2$ and all pairs of the form $(I \times Q_2, F \times Q_2)$ for $(I, F) \in \mathcal{R}_1$. ∎

Rabin Games The other use for the Rabin condition is to give winning conditions in games. A *Rabin game* is an *explicitly* represented planning problem whose goal is expressed as a Rabin condition $\mathcal{R}$ over the set of states.

Theorem 5 *(Buhrke, Lescow, and Vöge 1996; Emerson and Jutla 1988)*

1. *There is an algorithm that solves Rabin games in time $O(d!n^d m)$ where d is the number of Rabin pairs, n is the number of states, and m is the number of transitions.*
2. *In addition, solving Rabin games is* NP*-complete.*

[3] The reader might find it helpful to read the Rabin condition in LTL notation: $\bigvee_{(I,F)\in\mathcal{R}} \Box\Diamond I \wedge \neg\Box\Diamond F$.

[4] Recall that we define that an infinite trace satisfies an LTL$_f$ formula ψ if some prefix of it satisfies ψ.

Algorithm for Theorem 3 Given a state-action fair planning problem $\langle D, \psi\rangle$, reduce it to the problem of solving the Rabin game $G = (Ar, Acc)$ constructed as follows. The arena Ar is defined as the synchronous product of the domain $D = (St, Act, s_0, Tr)$ explicitly represented, and the DRW $M = M_{D,unfair} \vee M_\psi$. The Rabin winning condition Acc is induced by the Rabin acceptance condition $\mathcal{R}$ of M, i.e., Acc consists of all pairs of the form $(St \times I, St \times F)$ for $(I, F) \in \mathcal{R}$. To see that this reduction is sound and complete, note that a policy f solves the state-action fair planning problem $\langle D, \psi\rangle$ iff every f-trace in D is accepted by the DRW M iff every trace in G generated by the strategy that maps $(s_0, q_0)(s_1, q_1) \cdots (s_n, q_n)$ to the action $f(s_0 s_1 \cdots s_n)$ satisfies the Rabin condition Acc. The first iff is due to Theorem 4 and the fact that a trace is accepted by M means that if it were fair then it would satisfy ψ, and the second iff follows from the definition of Rabin condition and of the synchronous product. For the complexity analysis, consider the constructed Rabin game, and apply Theorem 5: n is polynomial in $|St|$ and doubly-exponential in $|\psi|$; d is polynomial in $|St|$ and exponential in $|\psi|$; and m is polynomial in $|St|$ and $|Act|$, and doubly-exponential in $|\psi|$. Recall that $|St|$ and $|Act|$ are at most exponential in the size of D compactly represented. Thus, the stated combined complexity follows from part 1 of Theorem 5, and the domain complexity from part 2 of Theorem 5.

6 Lower Bounds for State-action Fair Planning

For domain complexity, we note that existing results show the problem is 1EXPTIME-hard. This leaves open whether the domain complexity can be lowered from 1NEXPTIME to 1EXPTIME. For the goal (and thus combined) complexity we show that we can match the 2EXPTIME upper bound.

Domain Complexity It is not hard to get a 1EXPTIME lower bound for the domain complexity assuming state-action fairness by reducing from the problem of stochastic-fair planning with reachability goals, which is 1EXPTIME-complete (Littman 1997; Rintanen 2004). Indeed, introduce a fresh fluent p and fix the goal $\Diamond p$. Then, for a stochastic-fair planning problem with domain D and reachability goal $\Diamond target$, build a new domain D_p from D by adding the fluent p and a new action with precondition $target$ and post-condition p. Then the stochastic-fair problem $\langle D, \Diamond target\rangle$ has a solution iff the stochastic-fair problem $\langle D_p, \Diamond p\rangle$ has a solution. Moreover, the latter holds iff it has a finite state solution. By Proposition 1, this is equivalent to the fact that the state-action fair problem $\langle D_p, \Diamond p\rangle$ has a solution.

Goal Complexity We give a general technique to prove 2EXPTIME-hardness for goal complexities. Our proof is inspired by the 2EXPTIME-hardness of a closely related problem, i.e., if *every* policy almost surely enforces the LTL goal (Courcoubetis and Yannakakis 1995). That proof, and ours, is based on a reduction from the halting problem for EXPSPACE Turing-machines, which is 2EXPTIME-complete.

Theorem 6 *The goal complexity (and therefore also combined complexity) of planning for* LTL/LTL$_f$ *goals assuming*

either (1) no fairness, (2) stochastic fairness, or (3) state-action fairness is 2EXPTIME*-hard.*

Proof. We prove the no fairness case, and then show how to handle fairness. We provide a polynomial-time construction that, given an alternating EXPSPACE Turing machine M and an input word x, produces a probabilistic domain D (explicitly represented) and an LTL formula Φ, s.t. M accepts x iff $\exists f.(D,f) \models A\Phi$. To handle the goal complexity, the domain D will be fixed (i.e., independent of M and x).

An alternating Turing machine is a tuple $(Q, \Sigma, \Delta, q_0, q_a, q_r)$ where Q is the set of states partitioned into $Q_\exists$ and $Q_\forall$ (called the existential and universal modes), Σ is the tape alphabet, $\Delta \subseteq (\Sigma \times Q)^2 \times \{L, R, N\}$ is the transition relation, $q_0 \in Q$ is the initial state, and $q_a, q_r \in Q$ are the accepting and rejecting states. A *configuration* is a string matching the regular expression $\Sigma^* \cdot (\Sigma \times Q) \cdot \Sigma^*$; it is *initial* (resp. *accepting*, *rejecting*) if the state is q_0 (resp. q_a, q_r). A computation of M is a sequence of configurations, starting in an initial configuration, respecting the transition relation, and ending in an accepting or rejecting state. W.l.o.g, we assume that the existential and universal modes of M strictly alternate, with the existential going first.

Say M runs in space $2^{p(|x|)}$ for some polynomial $p(\cdot)$. I.e., a configuration of M running on x has length $\leq 2^{p(|x|)}$. Let $n := p(|x|)$. The domain D is constructed to allow the agent and the environment to generate strings of the form $C_0 \cdot (\# \cdot T_1 \cdot \#' \cdot C_1 \cdot \#'' \cdot K_1) \cdot (\# \cdot T_2 \cdot \#' \cdot C_2 \cdot \#'' \cdot K_2) \cdots (\# \cdot T_j \cdot \#' \cdot C_j \cdot \#'' \cdot K_j) \cdot \# \cdot \bot \cdot \bot \cdot \bot \cdots$ where the C_is are arbitrary strings over $\{0, 1, \%, \$\}$, the T_is and K_is are arbitrary strings over $\{0, 1\}$. The environment is responsible for generating the K_is (for all i) and the T_is (for even i), and the agent generates everything else. Intuitively, the C_is encode configurations of M, the T_is encode transitions of M, and the K_is encode a position/index $k \in [1, 2^n]$ on the tape that the environment wants to check. Finally, $\bot$ holds in a sink of the domain that the agent can go to when it is done. Note that this allows the agent to never go to the sink, but such traces will be rejected by the goal formula.

We now define the LTL$_f$ goal $\Phi := \Phi_{Env} \supset \Phi_{Ag}$. Intuitively, Φ will enforce that as long as the environment encodes its parts correctly (i.e., Φ_{Env} holds), then so does the agent, and the accepting state is reached (i.e., Φ_{Ag} holds). The formula $\Phi_{Ag} := \Phi_{conf} \wedge \Phi^{odd}_{tran} \wedge \Phi_{chal} \wedge \Phi_{acc}$, and $\Phi_{Env} := \Phi_{num} \wedge \Phi^{even}_{tran}$, where Φ_{conf} says that each C_i encodes a configuration, with C_0 encoding the initial configuration; Φ^{odd}_{tran} (resp. Φ^{even}_{tran}) says that each T_i with i odd (resp. even) encodes a transition of M; Φ_{num} says that each K_i encodes a number in $[1, 2^n]$; and Φ_{acc} says that an accepting configuration is reached; Φ_{chal} (think of it as a "challenge") says that the K_i position of the configuration C_i is the result of applying T_i to C_{i-1}. Recall that the environment generates the K_is and the transitions at even positions (which are the transitions from the universal mode). Thus, an agent policy f that enforces Φ_{Ag} has to satisfy Φ_{chal} for every possible value of K_i and T_i, for all i, i.e., regardless of which position of C_i is checked against C_{i-1}, and which transition is taken. Intuitively, this ensures that for every i, the configuration C_i is indeed the result of applying T_i to C_{i-1}, and that the accepting state is reached regardless of the transitions chosen from the universal mode.

Choose an $m \in \mathbb{N}$ to encode all members of Q and $\Sigma \times Q$ as binary strings of length exactly m. Let SYM denote a set of binary strings of length m that encode either a tape letter l or a tape letter/state pair (l, q). Let bin(i) denote the binary string of length n whose numeric value is i. The possible configurations of M are encoded by the strings of the form $\%\cdot\text{bin}(0)\cdot\$\cdot\text{SYM}\cdot\%\cdot\text{bin}(1)\cdot\$\cdot\text{SYM}\cdots\%\cdot\text{bin}(2^n{-}1)\cdot\$\cdot\text{SYM}$ which have exactly one symbol encoding a tape letter/state pair (l, q). The reason for the bin(i)s is that they allow the formula to check if an encoding of one configuration can be reached in one step of M from an encoding of another. We call the substring $\% \cdot \text{bin}(i) \cdot \$ \cdot w$ the ith block, where i is the *block number* and w is the *block symbol*. One can write an LTL$_f$ formula $\texttt{conf}$, of size linear in n and the size of M, that enforces this structure. Indeed, using a standard encoding of the binary counter on n-bit strings, the formula says that exactly one symbol encodes a tape letter/state pair, and all the other symbols encode just tape letters. It also says that $\text{bin}(0) = 0^n$, $\text{bin}(2^n - 1) = 1^n$, and for every $j \leq n$, the jth bit in a block is flipped in the next block iff all bits strictly lower in this block are 1s. Thus, the formula Φ_{conf} can be defined as $\texttt{init} \wedge \Box(\#' \supset \bigcirc\texttt{conf})$ where $\texttt{init}$ is a formula that encodes the initial configuration (which can be hardcoded by a polynomial sized formula by explicitly specifying the first $|x|$ blocks, and that the rest of the blocks in the configuration contain the encoding of the blank tape symbol). Writing linearly sized LTL$_f$ formulas Φ^{odd}_{tran}, Φ^{even}_{tran}, Φ_{num}, and Φ_{acc} poses no particular problem.

It remains to show how to build the formula Φ_{chal}. It will be the conjunction of two formulas Φ^1_{chal} and Φ^2_{chal}. The first handles the first challenge, and the second handles all the rest. We now show how to build the second (the first is similar). Define Φ^2_{chal} as: $\Box \bigwedge \left[(\texttt{cha} \wedge \texttt{cur}_x \wedge \texttt{nx}_y \wedge \texttt{nxnx}_z \wedge \texttt{tr}_t) \supset \texttt{img}_{y'}\right]$ where the conjunction is over tuples (x, y, z, t, y') such that applying the transition t to the triple of tape contents xyz (including a possible state) results in the tape content y' of the middle cell (e.g., for $t = (q, l, q', l', R)$, if $x = (l, q)$ then $y' = (y, q')$, if $x = l$ then $y' = y$, etc.). Intuitively, $\texttt{cha}$ expresses that we are currently at the start of a block of a configuration, say C_i, whose number is one less than the challenge number encoded by K_{i+1}; the formula $\texttt{cur}_x$ expresses that the symbol in the current block is x; the formula $\texttt{nx}_y$ expresses that the symbol in the next block is y; the formula $\texttt{nxnx}_z$ expresses that the symbol in the block after that is z; the formula $\texttt{tr}_t$ says that T_{i+1} encodes the transition t; and the formula $\texttt{img}_{y'}$ says that the block whose number is encoded by K_{i+1} in the configuration C_{i+1} is y'.

We use the following shorthand that can scan the string for patterns: define $\phi_1 \mathcal{J}^1 \phi_2 := (\neg\phi_1)\mathcal{U}(\phi_1 \wedge \bigcirc\phi_2)$ and $\phi_1 \mathcal{J}^2 \phi_2 := \phi_1 \mathcal{J}^1 (\phi_1 \mathcal{J}^1 \phi_2)$. Intuitively, $\phi_1 \mathcal{J}^i \phi_2$ means ϕ_2 holds one step after the ith occurrence of ϕ_1.

Formally, define $\texttt{cha}$ as:

$$\% \wedge \left[\%\mathcal{J}^2 \left(\bigwedge_{i\in[0,n)} \bigwedge_{b\in\{0,1\}} \left[\bigcirc^i b \iff \#''\mathcal{J}^2 \bigcirc^i b\right]\right)\right].$$

Define $\texttt{cur}_x$ as: $\$\mathcal{J}^1(\bigwedge_{i:0\leq i<m} \bigcirc^i b_i)$ where $b_1 b_2 \cdots b_m$ encodes the symbol x, and define $\texttt{nx}_y$ and $\texttt{nxnx}_z$ similarly. Define $\texttt{tr}_t$ as: $\#\mathcal{J}^1(\bigwedge_{i:0\leq i<m} \bigcirc^i t_i)$ where $t_1 t_2 \cdots t_m$ encodes the symbol t. Define $\texttt{img}_{y'}$ as: $\#'\mathcal{J}^1\left[(\texttt{match} \supset \bigcirc^n \bigwedge_{i:0\leq i<m} \bigcirc^i y'_i)\,\mathcal{U}\,\#''\right]$ where $y'_1 y'_2 \cdots y'_m$ encodes the symbol y', and match is $\bigwedge_{i\in[0,n)} \bigwedge_{b\in\{0,1\}} (\bigcirc^i b \iff \#''\mathcal{J}^1 \bigcirc^i b)$. Intuitively, it says that in the next configuration, if a block number equals the challenge number, then the block symbol should be y'.

This completes the definition of Φ, and the proof for the case of no fairness. For the fairness cases note that: a) if M accepts x then, already with no fairness assumptions, there is a solution, and b) if M rejects x, then for every policy f, the environment can, within a finite number of steps, prevent any hope of satisfying the goal: either by exposing that the agent is cheating in the simulation, or by reaching a rejecting configuration. Since every finite f-trace can be extended to a fair infinite f-trace, the policy f is not a solution to the state-action fair planning problem, nor to the stochastic fair planning problem since the set of infinite f-traces that extend this finite f-trace has positive probability. ∎

Note that some of the results in Theorem 6 are known: goal complexity with no fairness and LTL goals (Camacho, Bienvenu, and McIlraith 2019; Aminof et al. 2019a) and LTL_f goals (De Giacomo and Rubin 2018). The combined complexity in the stochastic fairness case for LTL goals is implicit in Courcoubetis and Yannakakis (1995). The case of state-action fairness, goal-complexity, LTL_f goals was stated in De Giacomo and Rubin (2018), but unfortunately with an incorrect proof.[5] Overall, our technique gives a uniform proof yielding tight lower bounds for combined and goal complexity for LTL/LTL_f with both kinds of fairness and no fairness, thus providing multiple missing lower-bounds.

7 Related Work and Discussion

Related Work in Verification We have discussed how the distinction between stochastic and state-action fairness is so far missing from the planning/AI literature. On the other hand, as we now discuss, this distinction is present in the verification literature. Early work in verification was motivated by the problem of providing formal methods (such as proof systems or model-checking algorithms) to reason about probabilistic concurrent systems. As such, some effort was made to abstract probabilities and capture stochastic fairness by language-theoretic properties. In fact, sophisticated forms of language-theoretic fairness were introduced to do this (Pnueli and Zuck 1993; Baier and Kwiatkowska 1998), since simple language-theoretic notions (similar to state-action fairness) were known not to capture stochastic fairness (Pnueli 1983).

Language-theoretic notions of fairness were explored in Aminof, Ball, and Kupferman (2004) where it was also noticed that product constructions may not preserve certain fairness notions (closely related to state-action fairness). A comprehensive study of fairness in reactive systems is provided in Völzer and Varacca (2012) where fairness is characterized language theoretically, game theoretically, topologically, and probabilistically. Fairness is used in verification of concurrent systems in order to prove liveness properties, i.e., that something good eventually happens. The limitations of fairness for proving liveness properties, as well as ways they may be overcome, are analysed in van Glabbeek and Höfner (2019).

The verification literature on probabilistic concurrent programs typically considers policies as schedulers. In particular, the central decision problem there is different to the planning problem: it asks whether every (rather than some) policy f almost surely enforces the temporally extended goal (Vardi 1985; Pnueli and Zuck 1993; Bianco and de Alfaro 1995; Courcoubetis and Yannakakis 1995). Logics for expressing game-theoretic properties of multi-agent stochastic systems have been studied, e.g., in Aminof et al. (2019b).

Discussion Planning in nondeterministic domains for LTL/LTL_f goals, with and without fairness, is attracting a lot of interest lately (Camacho et al. 2017; De Giacomo and Rubin 2018; Camacho, Bienvenu, and McIlraith 2019; Camacho and McIlraith 2019; Aminof et al. 2019a; Patrizi, Lipovetzky, and Geffner 2013; Camacho et al. 2017; De Giacomo and Rubin 2018; Brafman, De Giacomo, and Patrizi 2018). We now have a quite complete picture of the computational complexities, see Table 1.

	no f.	stoch. f.	state-action f.
Combined	2EXP-c	2EXP-c	2EXP-c
Goal	2EXP-c	2EXP-c	2EXP-c
Domain	1EXP-c	1EXP-c	NEXP/1EXP-h

Table 1: Planning in (compactly represented) nondeterministic domains for LTL/LTL_f goals.

In this table only one upper bound is not matched by the corresponding lower bound, namely the domain complexity of planning for LTL/LTL_f goals under state-action fairness.

Note that there is no difference in the complexities between LTL and LTL_f goals. However, in spite of the complexities being the same we do have well behaved algorithms for LTL_f goals with no fairness and with stochastic fairness, although not for LTL goals. Unfortunately, algorithms for automata over infinite traces are not as easy to implement as for finite traces (Fogarty et al. 2013). Regarding state-action fairness, for the moment it is not clear that even for LTL_f goals we can avoid the difficulties of handling LTL: our algorithm (Section 5) indeed requires automata over infinite traces.

8 Acknowledgements

Supported by the Austrian Science Fund (FWF): P 32021, and in part by the European Research Council under the European Union's Horizon 2020 Programme through the ERC Advanced Grant WhiteMech (No. 834228).

[5] That proof assumes that if f solves a state-action fair problem with LTL_f goal, then an f-trace visits every state of the domain at most once (before the goal is satisfied). Although true for memoryless strategies, which suffice for reachability goals, it is not true for finite-state strategies that are required for general LTL_f goals.

References

Alford, R.; Kuter, U.; Nau, D. S.; and Goldman, R. P. 2014. Plan aggregation for strong cyclic planning in nondeterministic domains. *Artif. Intell.* 216:206–232.

Aminof, B.; Ball, T.; and Kupferman, O. 2004. Reasoning about systems with transition fairness. In *LPAR*.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019a. Planning under LTL environment specifications. In *ICAPS*.

Aminof, B.; Kwiatkowska, M.; Maubert, B.; Murano, A.; and Rubin, S. 2019b. Probabilistic strategy logic. In *IJCAI*.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artif. Intell.* 116(1–2):123–191.

Baier, C., and Kwiatkowska, M. Z. 1998. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Inf. Process. Lett.* 66(2):71–79.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI*.

Bianco, A., and de Alfaro, L. 1995. Model checking of probabilistic and nondeterministic systems. In *FSTTCS*.

Bollig, B., and Leucker, M. 2004. Verifying qualitative properties of probabilistic programs. In *Validation of Stochastic Systems - A Guide to Current Research*, 124–146.

Brafman, R. I.; De Giacomo, G.; and Patrizi, F. 2018. LTL_f/LDL_f non-markovian rewards. *AAAI*.

Buhrke, N.; Lescow, H.; and Vöge, J. 1996. Strategy construction in infinite games with Streett and Rabin chain winning conditions. In *TACAS*.

Camacho, A., and McIlraith, S. A. 2019. Strong fully observable non-deterministic planning with LTL and LTLf goals. In *IJCAI*.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. 2017. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2019. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 1–2(147).

Courcoubetis, C., and Yannakakis, M. 1995. The complexity of probabilistic verification. *J. ACM* 42(4):857–907.

Daniele, M.; Traverso, P.; and Vardi, M. Y. 1999. Strong cyclic planning revisited. In *ECP*.

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of FOND planning for LTLf and LDLf goals. In *IJCAI*.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*.

D'Ippolito, N.; Rodríguez, N.; and Sardiña, S. 2018. Fully observable non-deterministic planning as assumption-based reactive synthesis. *J. Artif. Intell. Res.* 61:593–621.

Emerson, E. A., and Jutla, C. S. 1988. The complexity of tree automata and logics of programs. In *FOCS*.

Fainekos, G. E.; Kress-Gazit, H.; and Pappas, G. J. 2005. Temporal logic motion planning for mobile robots. In *ICRA*, 2020–2025. IEEE.

Fogarty, S.; Kupferman, O.; Vardi, M. Y.; and Wilke, T. 2013. Profile trees for Büchi word automata, with application to determinization. In *GandALF*.

Fu, J.; Jaramillo, A. C.; Ng, V.; Bastani, F. B.; and Yen, I. 2016. Fast strong planning for fully observable nondeterministic planning problems. *AMAI* 78(2):131–155.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. M&C.

Geffner, T., and Geffner, H. 2018. Compact policies for fully observable non-deterministic planning as SAT. In *ICAPS*.

Kissmann, P., and Edelkamp, S. 2011. Gamer, a general game playing agent. *Künst. Intell.* 25(1):49–52.

Littman, M. L. 1997. Probabilistic propositional planning: Representations and complexity. In *AAAI*.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *ICAPS*.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *IJCAI*.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI*.

Pnueli, A., and Zuck, L. D. 1993. Probabilistic verification. *Inf. Comput.* 103(1):1–29.

Pnueli, A. 1983. On the extremely fair treatment of probabilistic algorithms. In *STOC*.

Ramírez, M., and Sardiña, S. 2014. Directed fixed-point regression-based planning for non-deterministic domains. In *ICAPS*.

Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS*.

van Glabbeek, R., and Höfner, P. 2019. Progress, justness, and fairness. *ACM Comput. Surv.* 52(4):69:1–69:38.

Vardi, M. Y., and Wolper, P. 1986. An automata-theoretic approach to automatic program verification. In *LICS*.

Vardi, M. Y. 1982. The complexity of relational query languages. In *STOC*.

Vardi, M. Y. 1985. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*.

Vardi, M. Y. 1995. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency – Structure versus Automata (Banff)*, volume 1043 of *LNCS*.

Völzer, H., and Varacca, D. 2012. Defining fairness in reactive and concurrent systems. *J. ACM* 59(3):13:1–13:37.