

Generalized Planning with Loops under Strong Fairness Constraints

Giuseppe De Giacomo and Fabio Patrizi

Dipartimento di Informatica e Sistemistica
SAPIENZA Università di Roma
Roma, Italy
{degiacomo, patrizi}@dis.uniroma1.it

Sebastian Sardina

School of Computer Science and IT
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract

We consider a generalized form of planning, possibly involving loops, that arises in nondeterministic domains when explicit strong fairness constraints are asserted over the planning domain. Such constraints allow us to specify the necessity of occurrence of selected effects of nondeterministic actions over domain's runs. Also they are particularly meaningful from the technical point of view because they exhibit the expressiveness advantage of LTL over CTL in verification. We show that planning for reachability and maintenance goals is EXPTIME-complete in this setting, that is, it has the same complexity as conditional planning in nondeterministic domains (without strong fairness constraints). We also show that within the EXPTIME bound one can solve the more general problems of realizing agent planning programs as well as composition-based planning in the presence of strong fairness constraints.

Introduction

In this paper we consider a generalized form of planning, possibly involving loops, that arises in nondeterministic domains when explicit *strong fairness constraints* are asserted over the planning domain. Such constraints allow us to specify the necessity of occurrence of selected effects of nondeterministic actions over domain's runs.

More precisely, we consider a standard nondeterministic planning domain, that is, a finite state transition system described in the standard manner by means of action preconditions and *nondeterministic* effects (Rintanen 2004). On top of such a domain, we introduce *strong fairness constraints* expressed in Linear-time Temporal Logic (LTL), see e.g., (Clarke, Grumberg, and Peled 1999; Vardi 1996), that assert further properties of (possibly infinite) domain runs. Through such constraints, one can restrict the nondeterminism of actions in nontrivial ways. Specifically, one can specify that some selected effects of a nondeterministic action must occur infinitely often along every infinite evolution of the planning domain. For example, in modeling a gambling domain, one may specify that using a “Las Vegas” style slot machine, both winning and loosing happen infinitely often. We remark that, in general, such type of constraints does not apply to all nondeterministic actions: the action of chopping

a tree may result in the tree being (still) up or the tree falling down; however, once the tree falls, the former effect does not occur ever again.

In other words, strong fairness constraints, in particular on action executions, provide great flexibility in modeling planning scenarios. They allow for expressing *long-term* effects of action repetitions (e.g., tossing a coin an infinite number of times yields an infinite number of heads), or action fairness wrt effects (e.g., an infinite number of tails is also obtained). They also allow for distinguishing between those actions that guarantee their nondeterministic effects to eventually occur and those that do not.

Interestingly, fairness assumptions in nondeterministic domains are considered in the work on *strong cyclic* plans (Cimatti et al. 2003): a strong cyclic plan is a plan guaranteed to reach the goal under the (implicit) fairness assumption that *every* effect of a nondeterministic action *eventually* does occur. While this is often a realistic assumption—in particular, when nondeterminism stems from probabilistic effects like throwing a die or tossing a coin—in some cases it is not satisfactory. For instance, imagine a classical (mechanical) slot machine and an electronic one; the former is guaranteed to be fair, while the latter may not, due to a potential bug. Both machines have essentially the same description in the planning domain (apart, perhaps, from action names). Yet, they are very different, in that if the latter does indeed have a bug, then it may enable infinite losing runs. So, in order to eventually win one has to repeatedly play in the classical machine—no plan guarantees winning in the buggy electronic machine. In this paper, we aim then at giving the modeler the ability to control the nature of nondeterministic choices, by allowing her/him to state strong fairness conditions on selected effects of selected actions.

Strong fairness conditions are notoriously difficult to deal with in verification (Clarke, Grumberg, and Peled 1999). The most common temporal properties in verification are the following: (i) *reachability*: eventually something (good) becomes true; (ii) *maintenance* or *safety*: something (good) will be true forever; (iii) *weak fairness* or *response*: forever eventually something becomes true; we also have a well-known generalized form, sometimes called generically *liveness*: forever, every time something becomes true (i.e., the request), eventually something else becomes true (i.e., the

response); and (iv) *strong fairness* or *reactivity*: if something becomes true infinitely often, then something else becomes true infinitely often, as well. Note that strong fairness constraints generalize liveness constraints, in the sense that not necessarily all occurrences of the request need to be taken but infinitely many of them do. Such types of dynamic properties yield a sort of hierarchy (with reachability and maintenance together at the bottom) based on increasingly more sophisticated technical machinery required. The two main formalisms used in verification, LTL and CTL, can deal with the first four, but only LTL is able to deal with strong fairness (Clarke, Grumberg, and Peled 1999). Indeed, strong fairness is possibly the single reason that makes LTL the logic of choice (over CTL) for industrial verification (Accellera 2004; Armoni et al. 2002; Vardi 2007).

In this paper, then, we study techniques to solve planning problems in nondeterministic full observable domain *in the presence of strong fairness constraints*. Our work is thoroughly based on literature on verification and is quite novel in the context of AI. Indeed, most work on using temporal logic for planning in AI is based on CTL, which is currently the standard logic for planning by model checking (Ghallab, Nau, and Traverso 2004). Previous work on the use of LTL in planning has mainly focused on dealing with *temporally-extended goals* (Bacchus and Kabanza 1998; De Giacomo and Vardi 2000; Kerjean et al. 2006; Baier, Bacchus, and McIlraith 2009), often considering the temporal goals as complementary properties to be verified while reaching a main goal on finite runs only. Such complementary properties are typically used as a declarative means to control the search. Notice that LTL on finite runs is substantially simpler than standard LTL on infinite runs. Full LTL goals have been considered in (De Giacomo and Vardi 2000), where the domain was specified however as a transition system, possibly with partial observability. Very sophisticated forms of domain specification, based on second-order LTL, were considered in (Calvanese, De Giacomo, and Vardi 2002), again under partial observability.

Our results directly extend those for conditional planning. Indeed, by dropping constraints on the runs, we are left with a standard nondeterministic planning domain with full observability, and a solution for reachability goals thus amounts to finding a conditional plan. Hence, the fact that conditional planning is EXPTIME-complete provides the complexity lower bound for the problems that we are interested in. A gross upper bound for such problems is also available off-the-shelf: synthesis for arbitrary LTL formulas is 2EXPTIME-complete (Pnueli and Rosner 1989) and one can readily represent our planning problem under strong fairness constraints as an LTL synthesis problem. Unfortunately, techniques for *full* LTL synthesis, though known for a long time, have been resistant to practical implementations, due to the need of complementation of automata on trees, see e.g., (Kupferman, Piterman, and Vardi 2006).

We will show here that planning under strong fairness constraints in full observable domains remains EXPTIME-complete for a variety of increasingly sophisticated goals. To do so we adapt a specific form of LTL synthesis de-

veloped for so-called Generalized Reactivity GR(1) class, which is based on model checking of game structures (Piterman, Pnueli, and Sa’ar 2006) and which admits efficient symbolic implementation. Nonetheless, we are not able to use such techniques off-the-shelf, because the presence of strong fairness constraints gives rise to LTL formulas that fall outside the GR(1) class. Consequently, we first need to reduce the problem with strong fairness constraints to a problem with weak fairness constraints of the required GR(1) form.

The rest of the paper is organized as follows. We first introduce some preliminary notions on LTL synthesis. Then, we present planning under strong fairness constraints and show how to solve it. After that, we move to more advanced forms of planning, by considering recently introduced agent planning programs (De Giacomo, Patrizi, and Sardina 2010), and composition-based planning (Sardina, Patrizi, and De Giacomo 2008), both under strong fairness. We close the paper with some final remarks.

Preliminaries on LTL

Linear-time Temporal Logic (LTL) is a well-known logic used to specify dynamic or temporal properties of programs, see e.g., (Vardi 1996). *Formulas* of LTL are built from a set P of atomic propositions and are closed under the boolean operators, the unary temporal operators \bigcirc (*next*), \diamond (*eventually*), and \square (*always*, from now on), and the binary temporal operator \mathcal{U} (something eventually will hold and, *until* then, something else always holds). Interesting examples of LTL formulas are:

- $\diamond\varphi$: goal formula φ is eventually reached.
- $\square\varphi$: the goal formula φ is always maintained true.
- $\psi\mathcal{U}\varphi$: achieve goal φ while maintaining ψ .
- $\square\diamond\varphi$: formula φ is true infinitely often; this formula expresses “weak fairness” or “responsiveness.”
 $\square(\psi \rightarrow \diamond\varphi)$: forever, if formula ψ becomes true, then φ will eventually become true; this formula expresses a form of “liveness.”
- $\diamond\square\varphi$: eventually formula φ becomes true and remains true forever; this formula expresses “persistence.”
- $\square\diamond\varphi \rightarrow \square\diamond\psi$: if formula φ is true infinitely often, then also ψ is true infinitely often; this formula expresses “strong fairness” or “reactivity.”

All above formulas except the last one can be also expressed in CTL. The last one, “strong fairness,” is expressible only in LTL (cf. Introduction) and is the main focus of this paper.

LTL formulas are interpreted over infinite sequences σ of propositional interpretations for P , i.e., $\sigma \in (2^P)^\omega$. The set of (true) propositions at position i is denoted by $\sigma(i)$, hence σ is denoted by $\sigma(0), \sigma(1), \dots$. If σ is an interpretation, i a natural number, and ϕ is an LTL formula, we denote by $\sigma, i \models \phi$ the fact that ϕ holds in model σ at position i , which is inductively defined as follows (here, $p \in P$ is any proposition and ϕ, ψ any LTL formulas; we omit *until* for

brevity):

$\sigma, i \models p$	iff	$p \in \sigma(i)$;
$\sigma, i \models \phi \vee \psi$	iff	$\sigma, i \models \phi$; or $\sigma, i \models \psi$;
$\sigma, i \models \neg\phi$	iff	$\sigma, i \not\models \phi$;
$\sigma, i \models \bigcirc\phi$	iff	$\sigma, i+1 \models \phi$;
$\sigma, i \models \diamond\phi$	iff	for some $j \geq i$, we have that $\sigma, j \models \phi$;
$\sigma, i \models \square\phi$	iff	for all $j \geq i$, we have that $\sigma, j \models \phi$.

An interpretation σ satisfies ϕ , written $\sigma \models \phi$, if $\sigma, 0 \models \phi$. Standard logical tasks such as satisfiability or validity are defined as usual, e.g., a formula ϕ is *satisfiable* if there exists an interpretation that satisfies it. Checking satisfiability or validity for LTL is PSPACE-complete.

Here we are interested in a different kind of logical task, which is called *realizability* (aka *Church problem*) or *synthesis* (Vardi 1996; Pnueli and Rosner 1989). Namely, we partition P into two disjoint sets \mathcal{X} and \mathcal{Y} . We assume to have *no control* on the truth value of the propositions in \mathcal{X} , while we can control those in \mathcal{Y} . The problem then is: *can we control the values of \mathcal{Y} so that for all possible values of \mathcal{X} a certain LTL formula remains true?* More precisely, interpretations now assume the form $\sigma = (X_0, Y_0)(X_1, Y_1)(X_2, Y_2) \dots$, where (X_i, Y_i) is the propositional interpretation at the i -th position in σ , now partitioned in the propositional interpretation X_i for \mathcal{X} and Y_i for \mathcal{Y} . Let us denote by $\sigma_{\mathcal{X}|i}$ the interpretation σ projected only on \mathcal{X} and truncated at the i -th element (included), i.e., $\sigma_{\mathcal{X}|i} = X_0 X_1 \dots X_i$. The *realizability problem* checks the existence of a function $f : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for all σ with $Y_i = f(\sigma_{\mathcal{X}|i})$, σ satisfies formula φ . The *synthesis problem* consists in actually computing such a function. Observe that in realizability/synthesis we have no way of constraining the value assumed by \mathcal{X} propositions: the function we are looking for only acts on propositions in \mathcal{Y} . This means that the most interesting formulas for the synthesis have the form $\varphi_a \rightarrow \varphi_r$, where φ_a captures the “relevant” assignments of propositions in \mathcal{X} (and \mathcal{Y}) and φ_r specifies the property we want to assure for such relevant assignments. The realizability (and actual synthesis) are 2EXPTIME-complete for arbitrary LTL formulas (Pnueli and Rosner 1989). However, recently, several well-behaved patterns of LTL formulas have been identified, for which efficient procedures based on model checking technologies applied to game structures can be devised. Here, we focus on one of the most general well-behaved patterns, called “*Generalized Reactivity (1)*” or *GR(1)* (Piterman, Pnueli, and Sa’ar 2006). Such formulas have the form $\varphi_a \rightarrow \varphi_r$, with φ_a and φ_r of the following shapes:

$$\begin{aligned} \varphi_a: & \Phi[\mathcal{X}, \mathcal{Y}] \wedge \bigwedge_j \square\Phi_j[\mathcal{X}, \mathcal{Y}, \bigcirc\Phi[\mathcal{X}]] \wedge \bigwedge_k \square\diamond\Phi_k[\mathcal{X}, \mathcal{Y}], \\ \varphi_r: & \Phi[\mathcal{X}, \mathcal{Y}] \wedge \bigwedge_j \square\Phi_j[\mathcal{X}, \mathcal{Y}, \bigcirc\Phi[\mathcal{X}, \mathcal{Y}]] \wedge \bigwedge_k \square\diamond\Phi_k[\mathcal{X}, \mathcal{Y}], \end{aligned}$$

where $\Phi[\mathcal{Z}]$ stands for any boolean combination of symbols from \mathcal{Z} . Notice that: (i) the first conjunct expresses initial conditions; (ii) the second (big) conjunct expresses transitions –with the limitation that φ_a cannot talk about the next value of \mathcal{Y} propositions; and (iii) the third (big) conjunct expresses *weak fairness* constraints of the form “it is always true that eventually something holds.” However, *one cannot express strong fairness constraints in GR(1) formulas* (but

see below). For GR(1) formulas, realizability and synthesis are substantially simpler than for general LTL formulas:

Theorem 1 (Piterman, Pnueli, and Sa’ar 2006).

*Realizability (and synthesis) of GR(1) LTL formulas $\varphi_a \rightarrow \varphi_r$ can be determined in time $O((p * q * w)^3)$, where p and q are the number of conjuncts of the form $\square\diamond\Phi$ in φ_a and φ_r , respectively,¹ and w is the number of possible value assignments of \mathcal{X} and \mathcal{Y} under the conditions of φ_a ’s first two conjuncts.*

Planning under Strong Fairness

The system we are interested to plan on consists of (i) a standard nondeterministic *planning dynamic domain*, modeling the potential evolutions of world, enriched with (ii) a set of *strong fairness constraints*, ruling out unfeasible evolutions.

Dynamic domain A *dynamic domains* is a tuple $\mathcal{D} = \langle P, \Sigma, A, S_0, \rho \rangle$, where:

- $P = \{p_1, \dots, p_n\}$ is a finite set of *domain propositions*;
- $\Sigma = 2^P$ is the set of *domain states*;
- $A = \{a_1, \dots, a_r\}$ is the finite set of *domain actions*;
- $S_0 \in \Sigma$ is the *initial state*;
- $\rho \subseteq \Sigma \times A \times \Sigma$ is the *domain transition relation*. We freely interchange transition notations $\langle S, a, S' \rangle \in \rho$ and $S \xrightarrow{a} S'$ in \mathcal{D} .

At each time point, a dynamic domain is in one of its states; initially, S_0 . An action a is *executable* in a state S if $S \xrightarrow{a} S'$ in \mathcal{D} for some S' . In such a case, S' is a (possible) *a-successor* of S . An action a is *nondeterministic* if there exists a state S having more than one a -successor. Each state $S \in \Sigma$ represents a complete valuation $\mu : P \mapsto \{\top, \perp\}$ such that $\mu(p) = \top$ iff $p \in S$. Consequently, a propositional formula φ identifies a subset of Σ , namely, those states whose valuations satisfies φ .

Dynamic domain’s potential evolutions are called *runs*.

Technically, a *run* λ is a sequence of the form $S^0 \xrightarrow{a^0} S^1 \xrightarrow{a^1} \dots$ such that $S^0 = S_0$ and $\langle S^i, a^i, S^{i+1} \rangle \in \rho$, for $i \geq 0$. For convenience, and wlog, we take runs to be *infinite*. To that end, we assume the existence of a special proposition *end* and a special action *noOp*, which when executed in any state S leads to an absorbing state $S \cup \{end\}$ —at any time point, one can stop executing domain actions (forever).

Strong fairness constraints In addition to the usual *step-by-step* constraints that the domain transition relation induces on runs, we consider more general constraints affecting runs’ whole extension. Formally, a *constraint on domain runs* is an LTL formula γ over propositional vocabulary $PROP = P \cup P_A$, where $P_A = \{act = a \mid a \in A\}$ is the set of *action propositions*: proposition

¹We assume that both φ_a and φ_r contain at least one conjunct of such a form, if not, we vacuously add the trivial one $\square\diamond\top$.

($act = a$) states that the current action is a . To interpret such formulas over domain runs of the form $\lambda = S^0 \xrightarrow{a^0} S^1 \xrightarrow{a^1} \dots$, we simply consider the corresponding sequence $\sigma_\lambda = (S^0 \cup \{act = a^0\}), (S^1 \cup \{act = a^1\}), \dots$, and say that λ *satisfies a constraint* γ (denoted $\lambda \models \gamma$) iff $\sigma_\lambda \models \gamma$ as explained in the previous section.

A *strong fairness constraint* is an LTL formula of the form $\Box \Diamond \phi \rightarrow \Box \Diamond \psi$ over $PROP = P \cup P_A$, with ϕ and ψ containing no temporal operator other than \bigcirc , which can never occur nested² (e.g., $\bigcirc \bigcirc \phi$ and $\bigcirc(p \wedge \bigcirc q)$ are not allowed). Observe that *weak fairness* (i.e., $\Box \Diamond \phi$) is captured by constraints of the form $\Box \Diamond \top \rightarrow \Box \Diamond \phi$. Similarly, *persistence* (i.e., $\Diamond \Box \phi$) is captured by $\Box \Diamond \neg \phi \rightarrow \Box \Diamond \perp$.

To better understand how strong fairness constraints can help expressing certain domains, let us next illustrate their use with some examples.

Example 1. The “Las Vegas” slot machine scenario presented in the Introduction can be easily modeled as a domain $\mathcal{D} = \langle \{win\}, \{\{win\}, \emptyset\}, \{play\}, \emptyset, \{S, play, S'\} \mid S, S' \subseteq \{win\} \rangle$. States $\{win\}$ and \emptyset (i.e., $\neg win$) represent the cases in which the player has just won and lost, respectively. The domain transition relation states that playing at any states leads \mathcal{D} to evolve nondeterministically to either states $\{win\}$ or \emptyset .

So, in order to win, the best a player can do is play indefinitely (assuming an infinite budget, of course). However, the always losing sequence $\lambda_{lose} = \emptyset \xrightarrow{play} \emptyset \xrightarrow{play} \emptyset \xrightarrow{play} \dots$ is a perfectly valid run under \mathcal{D} —the possibility that the player never wins does exist! Consequently, there is no strategy that can guarantee the player’s ambitious goal.

A natural assumption for *real-world* slot machines, though, is that if someone plays infinitely often, then there will indeed be infinitely many good rounds and, of course, infinitely many bad ones. Technically, such information can be easily stated using two strong fairness constraints: $\Box \Diamond act = play \rightarrow \Box \Diamond \neg win$ and $\Box \Diamond act = play \rightarrow \Box \Diamond win$. ■

Observe it is not always the case, though, that such constraints apply to *all* effects of a nondeterministic action.

Example 2. In the *tree chopping* scenario (Sardina et al. 2006): with each chop, a tree may or may not fall down, but once down, the tree remains as such; also, continuous chopping does eventually bring the tree down. While the first part of the scenario can be easily captured by a standard domain (in particular, the fact that once a tree has fallen down, it remains down), the effectiveness of the chopping action can be captured by means of strong fairness, via the following constraint: $\Box \Diamond act = chop \rightarrow \Box \Diamond down$. ■

As it can be seen, the example does not include a strong fairness constraint for one of the possible effects of the chopping action, that is: $\Box \Diamond act = chop \rightarrow \Box \Diamond \neg down$. Also, note that the constraint capturing the effectiveness of the chopping action applies even in a domain where the tree can be brought up again.

²In fact, nesting of operator \bigcirc raises no conceptual obstacle. We avoid it for readability reasons only.

A dynamic domain with strong fairness constraints is called a *dynamic system*. Concretely, a *dynamic system* is a pair $\mathcal{S} = \langle \mathcal{D}, \mathcal{C} \rangle$, where \mathcal{D} is a planning dynamic domain and \mathcal{C} is a finite set of strong fairness constraints.

Plans Next, we formalize the notion of *plans* and their respective *executions* on a dynamic system $\mathcal{S} = \langle \mathcal{D}, \mathcal{C} \rangle$.

The *length* of a generic finite sequence $\tau = S^0 \xrightarrow{a^0} S^1 \dots S^{\ell-1} \xrightarrow{a^{\ell-1}} S^\ell$ is $|\tau| \doteq \ell + 1$, if τ is infinite, then $|\tau| \doteq \infty$. Also, if τ is finite, we define $last(\tau) = S^\ell$. Given a (finite or infinite) sequence $\tau = S^0 \xrightarrow{a^0} S^1 \xrightarrow{a^1} \dots$, we define its *finite prefix of length* k (for $0 < k < |\tau| + 1$) as the sequence $\tau|_k = S^0 \xrightarrow{a^0} \dots \xrightarrow{a^{k-2}} S^{k-1}$.

A *history* of \mathcal{S} is a *finite prefix* of a run of \mathcal{S} . The set of all histories of \mathcal{S} is referred to as \mathcal{H} . A *general plan* over \mathcal{S} is a function $\pi : \mathcal{H} \mapsto A$. The set of all general plans over a domain \mathcal{S} is referred to as Π .

An *execution* of a general plan π on system \mathcal{S} is a, possibly infinite, sequence $\eta = S^0 \xrightarrow{a^0} S^1 \xrightarrow{a^1} \dots$ such that (i) $S^0 = S_0$; (ii) for all $0 < k \leq |\eta|$, $\tau|_k$ is a history of \mathcal{S} ; (iii) $a^{k-1} = \pi(\tau|_k)$, for all $0 < k < |\eta|$; and (iv) if η is finite, then $\pi(last(\eta))$ is undefined.

When all possible executions of a general plan are finite, the plan is called a *generalized conditional plan* (generalized since, in the presence of constraints on runs, they may involve loops). Informally, generalized conditional plan executions are guaranteed to eventually terminate.

Goals We generalize the classical notion of reachability goal. Given a dynamic domain \mathcal{S} , let ϕ and ψ be propositional formulae over P . A general plan π *achieves ϕ while maintaining ψ* (written $\pi \models \psi \mathcal{U} \phi$) if for each of its (finite or infinite) executions η , there exists a k , $0 \leq k \leq |\eta|$, such that $S^k \models \phi$ and $S^{k'} \models \psi$, for all $0 \leq k' \leq k$, and if η is finite, then $k = |\eta|$.

Example 3. In a production line, when component items reach the assemblage section, they are often *dusty* and *greasy*. The line provides two cleaning methods: one by spraying air and another by spraying a special solvent. A finite number (depending on how dirty the item is) of air sprays ensures all the dust to be eventually removed from a given item; analogously a finite number of solvent sprays removes all the grease. Air (resp., solvent) sprays are *sometimes* also effective in removing grease (resp., dust)—there is no guarantee for this though.

Figure 1 reports a fragment of the PDDL specification corresponding to dynamic domain \mathcal{D} for the scenario. Expression $when((cond)(eff))$ states that if $cond$ holds before action execution then eff holds after it (i.e., conditional effects); whereas $oneof(e_1, \dots, e_n)$ states that the (nondeterministic) action yields one effect among e_1, \dots, e_n .

Predicates *dusty* and *greasy* represent the item’s current state. The domain provides two nondeterministic actions `sprayAir` and `spraySol` with no precondition and the following effects: (i) if the item is not dusty (greasy), then it

```

(define (domain productionLine)
  (:predicates (dusty) (greasy))
  (:action sprayAir
    :effect (and
      (when (not (dusty)) (not (dusty)))
      (when (not (greasy)) (not (greasy)))
      (when (dusty) (oneof (dusty) (not (dusty))))
      (when (greasy) (oneof (greasy) (not (greasy))))))
  (:action spraySol ... ); see sprayAir
  (:init (and (dusty) (greasy))))

```

Figure 1: Planning domain for the production line example.

remains as such after execution; and (ii) if the item is dusty (greasy), then dust (grease) may or may not be removed after execution. The effectiveness of these actions on dust and grease, respectively, is captured by the following two strong fairness constraints:

$$\begin{aligned} & \Box \Diamond (act = \text{sprayAir} \wedge dusty) \rightarrow \\ & \quad \Box \Diamond (act = \text{sprayAir} \wedge dusty \wedge \bigcirc \neg dusty); \\ & \Box \Diamond (act = \text{spraySol} \wedge greasy) \rightarrow \\ & \quad \Box \Diamond (act = \text{spraySol} \wedge greasy \wedge \bigcirc \neg greasy). \end{aligned}$$

A procedure is needed to prepare each item for the assemblage process, that is, each item needs to be free of dust and grease. Formally, this requires a plan π such that $\pi \models \top \mathcal{U}(\neg dusty \wedge \neg greasy)$. Clearly, such a plan does exist: first repeat action `sprayAir` until no dust is present on the item, and then repeat action `spraySol` until no grease remains. Note that loops are required: one spray may not be enough. Also, observe that there can be executions where spraying air (solvent) is enough alone to remove both dust and grease, so that no further processing is needed. Nonetheless, only executing both loops guarantees all runs to eventually reach the goal. Of course, other plans may exist, e.g., one where both actions are interleaved. ■

With this example at hand, let us next see how to effectively solve our *extended* planning problems.

Solving Planning under Strong Fairness

Here, we face the problem of building plans that achieve and maintain desired goal formulae. Let us start by formally stating the extended planning task: given a dynamic system $\mathcal{S} = \langle \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{D} = \langle P, \Sigma, A, S_0, \rho \rangle$, an achievement goal ϕ and a maintenance goal ψ (both being propositional formulae over P), the problem of *planning for reachability and maintenance under strong fairness constraints* requires to build a general plan π over \mathcal{S} such that $\pi \models \psi \mathcal{U} \phi$.

Observe that when $\psi = \top$, we obtain the standard notion of reachability goal, and if in addition $\mathcal{C} = \emptyset$, then the problem reduces to classical planning with nondeterminism and full observability, for which the following is a known result; see, e.g., (Rintanen 2004).

Theorem 2. *Given a dynamic system $\mathcal{S} = \langle \mathcal{D}, \emptyset \rangle$ and an achievement goal ϕ , synthesizing a plan π over \mathcal{S} such that $\pi \models \top \mathcal{U} \phi$ is EXPTIME-complete.*

In order to deal with the cases in which $\mathcal{C} \neq \emptyset$, we shall reduce the problem to synthesis of GR(1) specifications (Piterman, Pnueli, and Sa’ar 2006). To do so, we exploit the construction proposed in (Kesten, Piterman, and Pnueli 2005) to reduce “*fair discrete systems (fds)*” to “*just discrete systems (jdf)*”, so as to come up with a problem formulation compliant with the GR(1) form. To ease the presentation, we present the reduction as a three-step process.

LTL encoding of dynamic systems In the first step, we develop an LTL encoding $\widehat{\varphi}_{\mathcal{S}}$ of \mathcal{S} , whose runs capture all \mathcal{S} evolutions, where the occurrences of operator \bigcirc in strong fairness constraints are compiled away. This will be useful in the next step.

Let \mathcal{D} be as above. The set of propositions that $\widehat{\varphi}_{\mathcal{S}}$ is built upon is $\widehat{P} = P \cup P_A \cup P_X$, where P_X contains one proposition $px.\xi$ for each subformula $\bigcirc\xi$ appearing in some strong fairness constraint $\gamma \in \mathcal{C}$. The intended meaning of proposition $px.\xi$ is to hold iff on next state ξ holds.

We stress that $px.\xi$ is introduced only for syntactic convenience, so as to ease the reduction of the obtained LTL encoding to the GR(1) form. Moreover, for convenience, for each state $S \in \Sigma$, we define a propositional formula $\gamma_S = \bigwedge_{i=1}^n l_i$, where $l_i = p_i$ if $p_i \in S$, and $l_i = \neg p_i$ otherwise.³

So, we define $\widehat{\varphi}_{\mathcal{S}} = \widehat{\varphi}_{\mathcal{S}}^{init} \wedge \Box \widehat{\varphi}_{\mathcal{S}}^{trans} \wedge \widehat{\varphi}_{\mathcal{S}}^{rc}$, with $\widehat{\varphi}_{\mathcal{S}}^{trans} = \widehat{\varphi}_{\mathcal{S}}^{\rho} \wedge \widehat{\varphi}_{\mathcal{S}}^{next}$, where:

- $\widehat{\varphi}_{\mathcal{S}}^{init} = \gamma_{S_0}$, i.e., \mathcal{D} starts in its initial state;
- Formula $\widehat{\varphi}_{\mathcal{S}}^{\rho}$ is defined as follows:

$$\bigwedge_{S \in \Sigma, p_a \in P_A} [\gamma_S \wedge p_a \wedge \bigwedge_{p'_a \in P_A \setminus \{p_a\}} \neg p'_a \rightarrow \bigcirc \bigvee_{\langle S, a, S' \rangle \in \rho} \gamma_{S'}],$$

where p_a abbreviates proposition $(act = a)$. Each conjunct asserts that if \mathcal{D} is in state S and action a is to be executed, then one of the possible successor states w.r.t. ρ is indeed the next state of \mathcal{D} (an empty set of disjuncts is assumed \perp);

- $\widehat{\varphi}_{\mathcal{S}}^{next} = \bigwedge_{px.\xi \in P_X} px.\xi \leftrightarrow \bigcirc \xi$, that is $px.\xi$ holds in current state iff ξ will hold in next state;
- $\widehat{\varphi}_{\mathcal{S}}^{rc} = \bigwedge_{\gamma \in \mathcal{C}} \gamma[\bigcirc \xi / px.\xi]$, where $\gamma[\alpha / \beta]$ means the formula obtained by replacing each occurrence of subformula α with formula β in γ . By doing so, each constraint in $\widehat{\varphi}_{\mathcal{S}}^{rc}$ assumes the form $\widehat{\gamma} = \Box \Diamond \widehat{\phi} \rightarrow \Box \Diamond \widehat{\psi}$, where $\widehat{\phi}$ and $\widehat{\psi}$ are temporal operator free, while preserving its semantics, due to the above constraint on $px.\xi$.

Observe that the obtained specification is, essentially, an LTL representation of the original system \mathcal{S} , where $\widehat{\varphi}_{\mathcal{S}}^{init}$

³Note that the set of formulae has linear size in the number of states, which is exponential in the number of propositions, as the domain transition relation is described using explicit states. However, if a compact representation is used, as done, e.g., in PDDL, it can be polynomially encoded in LTL. In any case, as shown later, our results do not depend on the size of the domain’s LTL encoding, but directly on the size of the domain’s state space.

and $\widehat{\varphi}_S^{trans}$ capture the information about \mathcal{D} , and $\widehat{\varphi}_S^{rc}$ represents the (\bigcirc -free) strong fairness constraints. So, the construction yields the following, straightforward, result:

Lemma 3. *A sequence $\lambda = S^0 \xrightarrow{a^0} S^1 \xrightarrow{a^1} \dots$ is a run of \mathcal{S} iff $\sigma_\lambda \models \widehat{\varphi}_S$, where $\sigma_\lambda = (S^0 \cup \{p_{a^0}\} \cup P_X^0), (S^1 \cup \{p_{a^1}\} \cup P_X^1), \dots$ such that for all $i \geq 0$, $px.\xi \in P_X^i$ iff $S^{i+1} \models \xi$.*

From strong to weak fairness The second step consists in compiling away the strong fairness constraints. This is needed to guarantee a GR(1) specification to be obtained at the end of the reduction process. Precisely, $\widehat{\varphi}_S$ above is transformed into an equivalent formula (i.e., satisfied by exactly the same runs) $\varphi_S = \varphi_S^{init} \wedge \square \varphi_S^{trans} \wedge \varphi_S^{rc}$, where each conjunct of φ_S^{rc} is a *weak* fairness constraint.

First, we define P_C as the set containing one proposition $n_{\widehat{\phi}}$ for each conjunct $\widehat{\gamma} = \square \diamond \widehat{\phi} \rightarrow \square \diamond \widehat{\psi}$ occurring in $\widehat{\varphi}_S^{rc}$. Let $\widehat{\mathcal{C}}$ be the set of all such conjuncts. Each proposition $n_{\widehat{\phi}}$ is intended to hold at a given point in a run if, from that point on, $\widehat{\phi}$ remains false forever; in addition, we introduce a proposition x_c becoming true at a given point if some $n_{\widehat{\phi}}$ is a mis-prediction (see below).

Then, we define φ_S 's conjuncts as follows:

- $\varphi_S^{init} = \widehat{\varphi}_S^{init} \wedge \neg x_c \wedge \bigwedge_{n_{\widehat{\phi}} \in P_C} \neg n_{\widehat{\phi}}$;
- $\varphi_S^{trans} = \widehat{\varphi}_S^{trans} \wedge \varphi_S^{\widehat{\phi}} \wedge \varphi_S^c$, where:
 - $\varphi_S^{\widehat{\phi}} = \bigwedge_{n_{\widehat{\phi}} \in P_C} n_{\widehat{\phi}} \rightarrow \bigcirc n_{\widehat{\phi}}$;
 - $\varphi_S^c = \bigcirc x_c \leftrightarrow (x_c \vee \bigvee_{(\square \diamond \widehat{\phi} \rightarrow \square \diamond \widehat{\psi}) \in \widehat{\mathcal{C}}} (\widehat{\phi} \wedge n_{\widehat{\phi}}))$;
- $\varphi_S^{rc} = \square \diamond \neg x_c \wedge \bigwedge_{(\square \diamond \widehat{\phi} \rightarrow \square \diamond \widehat{\psi}) \in \widehat{\mathcal{C}}} \square \diamond (n_{\widehat{\phi}} \vee \widehat{\psi})$.

Observe that no strong fairness constraint appears in the obtained formula. Indeed, φ_S^{rc} is a conjunction of weak fairness formulae ($\square \diamond \xi$) only. Exploiting the results in (Kesten, Piterman, and Pnueli 2005) we get:

Lemma 4. *For every run σ , $\sigma \models \widehat{\varphi}_S$ if and only if $\sigma \models \varphi_S$.*

Building plans We can now show the final step of the reduction, i.e., the encoding of the problem as a GR(1) specification. Taking the LTL formula φ_S as above, we start building the GR(1) formula $\Upsilon = \varphi_a \rightarrow \varphi_r$ by specifying the sets of uncontrolled and controlled propositions, and then build the assumption and requirement formulas.

Uncontrolled and controlled propositions. The set of *uncontrolled* propositions is $\mathcal{X} = P \cup P_X \cup P_C \cup \{x_c\} \cup \{ach, mnt\}$, where all sets except $\{ach, mnt\}$ are as above. Proposition *ach* is intended to record that formula ϕ has already been achieved along a run (either in the current or in the past); similarly, proposition *mnt* records that ψ has been maintained along a run up to the state (included) where ϕ has been satisfied. Finally, the set of *controlled* propositions is simply $\mathcal{Y} = P_A$, i.e., the domain actions.

Assumption formula. The formula encoding how the domain is *expected* to behave when a plan is under execution is defined as $\varphi_a = \varphi_a^{init} \wedge \varphi_a^{trans} \wedge \varphi_a^{rc}$.

Propositional formula $\varphi_a^{init} = \varphi_S^{init} \wedge ach \equiv \phi \wedge mnt \equiv \psi$ characterizes the initial state of the system: \mathcal{S} starts in its initial state and *ach* and *mnt* hold iff ϕ and ψ do, respectively.

LTL formula $\varphi_a^{trans} = \square(\varphi_S^{trans} \wedge \varphi_{ach}^{trans} \wedge \varphi_{mnt}^{trans})$ characterizes the assumptions on *ach*, *mnt* and \mathcal{S} 's evolutions. Formula φ_S^{trans} has been discussed above, whereas φ_{ach}^{trans} and φ_{mnt}^{trans} are defined as follows:

$$\varphi_{ach}^{trans} = (ach \rightarrow \bigcirc ach) \wedge (\neg ach \rightarrow \bigcirc(ach \equiv \phi));$$

$$\varphi_{mnt}^{trans} = (\neg mnt \rightarrow \bigcirc \neg mnt) \wedge (mnt \wedge ach \rightarrow \bigcirc mnt) \wedge (mnt \wedge \neg ach \rightarrow \bigcirc(mnt \equiv \psi)).$$

That is, *ach* holds if ϕ has been achieved in the past or in the current state, whereas *mnt* holds iff ψ was never violated in the past before ϕ was achieved.

Finally, we simply take $\varphi_a^{rc} = \varphi_S^{rc}$ so as to capture the strong fairness constraints originally defined on \mathcal{D} 's runs (re-modeled during the above reduction phase).

Requirement formula. Lastly, we construct formula $\varphi_r = \square \varphi_{act}^{trans} \wedge \varphi_r^{goal}$ capturing the *requirements* for the plan to be synthesized. Formula φ_{act}^{trans} encodes the action execution constraints, requiring one and only one action to be executed at each step: $\varphi_{act}^{trans} = \bigvee_{y \in \mathcal{Y}} [y \wedge \bigwedge_{y' \in \mathcal{Y} \setminus \{y\}} \neg y']$. As for the synthesis “objective”, it is simply the *weak fairness* formula $\varphi_r^{goal} = \square \diamond (ach \wedge mnt)$. That is, we require a successful plan to always eventually bring about ϕ in every run (i.e., equivalent to *ach* being eventually true) while not violating ψ up to then (i.e., equivalent to not falsifying *mnt*).

It is not hard to check that the LTL formula Υ obtained is indeed in GR(1) form. Hence, we can apply the results from (Piterman, Pnueli, and Sa’ar 2006) and thus obtain the following theorem.

Theorem 5 (Soundness & Completeness). *There exists a plan over \mathcal{S} that achieves ϕ while maintaining ψ iff LTL formula Υ , constructed as above, is realizable.*

As for complexity considerations, analyzing the structure of Υ , we observe that: (i) φ_a contains as many subformulae of the form $\square \diamond \xi$ as strong fairness constraints in \mathcal{C} , namely, $|P_C|$; (ii) φ_r contains just one such subformula; (iii) the number of possible value assignments of \mathcal{X} and \mathcal{Y} under the conditions of $\varphi_a \rightarrow \varphi_r$ is $O(2^{|P|+|P_X|+|P_C|} \cdot |P_A|)$ given that only one P_A proposition can be true at each step. Consequently, from Theorem 1, checking the existence of a plan for reachability and maintenance can be done in $O((|P_C| \cdot 1 \cdot (2^{|P|+|P_X|+|P_C|} \cdot |P_A|))^3)$. Hence together with the EXPTIME-hardness of Theorem 2, we get a tight complexity characterization for our problem.

Theorem 6 (Complexity). *Checking the existence of a plan that achieves ϕ while maintaining ψ in a dynamic domain under strong fairness constraints is EXPTIME-complete.*

We remark that the technique of (Piterman, Pnueli, and Sa’ar 2006) synthesizes an actual solution to the problem, not merely verifies its existence: one actually gets the plan out of the realizability checking.

Agent Planning Programs

In this section we turn our attention to *agent planning programs* (De Giacomo, Patrizi, and Sardina 2010), which are high-level specifications of *desired* agent behaviors in terms of *declarative goals*.

Agent planning programs An *agent planning program*, or simply a *planning program*, for a dynamic domain \mathcal{S} is a tuple $\mathcal{T} = \langle T, \mathcal{G}, t_0, \delta \rangle$, where:

- $T = \{t_0, \dots, t_n\}$ is the finite set of *program states*;
- \mathcal{G} is a set of (extended) goals of the form (ψ, ϕ) : *achieve ϕ while maintaining ψ* ;
- $t_0 \in T$ is the *program initial state*;
- $\delta \subseteq T \times \mathcal{G} \times T$ is the *transition relation*. We freely interchange notations $\langle t, \psi, \phi, t' \rangle \in \delta$ and $t \xrightarrow{\psi, \phi} t'$ in \mathcal{T} .

When an agent planning program is “realized,” the following happens: at any point in time, the planning program is in a state t and the system, or more precisely the domain, in a state S (initially, states t_0 and S_0 , respectively); the agent then chooses to perform any transition $t \xrightarrow{\psi, \phi} t'$ (outgoing from t); then, starting from S , a course of actions that brings the domain to a state satisfying ϕ while only traversing states satisfying ψ is executed; finally, the agent planning program moves to t' and the agent may choose a new transition $t' \xrightarrow{\psi', \phi'} t''$, and so on. Notice that the executed actions must guarantee, at any point in time, the feasibility of all possible (planning program’s) transitions the agent can choose next, once the planning program has reached its successor state. This is because the agent makes its decisions in a step-by-step fashion. The problem we deal with in this section amounts to concretely *realizing* such programs.

We formalize the planning program semantics by generalizing what was proposed in (De Giacomo, Patrizi, and Sardina 2010) to our context.

An agent planning program \mathcal{T} is *realizable* if there exists a function, called *\mathcal{T} -realization*, $\omega : \mathcal{H} \times \delta \mapsto \Pi$ such that:

1. for all transitions $t_0 \xrightarrow{\psi, \phi} t$ in \mathcal{T} , $\omega(S_0, t_0 \xrightarrow{\psi, \phi} t)$ is defined, that is, there is a plan for every possible initial request;
2. if $\omega(h, t \xrightarrow{\psi, \phi} t')$ is defined with $\omega(h, t \xrightarrow{\psi, \phi} t') = \pi$, then:
 - (a) π is a generalized conditional plan for system \mathcal{S}_h , where \mathcal{S}_h is obtained from \mathcal{S} by replacing its initial state S_0 with state $last(h)$;
 - (b) $\pi \models \psi \mathcal{U} \phi$ in \mathcal{S}_h , that is, π achieves ϕ while maintaining condition ψ when executed in system \mathcal{S}_h ;
 - (c) for all π ’s executions η and all possible next transitions $t' \xrightarrow{\psi', \phi'} t'' \in \delta$, $\omega(h \cdot \eta, t' \xrightarrow{\psi', \phi'} t'')$ is defined (note $h \cdot \eta$ is well defined as they end and start in the same state, respectively.)

The problem we are concerned with is then: *how such a function ω can be built?* Once again, it turns out that the problem can be solved by resorting to LTL synthesis for GR(1) specifications.

Realizing agent planning programs We build a GR(1) formula $\Theta = \varphi_a \rightarrow \varphi_r$ in an analogous way as done in the previous section. So, assume \mathcal{T} is an agent planning program to be realized in a dynamic system \mathcal{S} .

Let $\varphi_{\mathcal{S}}$ be the corresponding LTL specification with no strong fairness constraints, obtained as shown before, and let $P_{\varphi_{\mathcal{S}}} = P \cup P_X \cup P_C \cup \{mnt, ach, x_c\}$. In the following construction, we shall refer to the reduction presented in the previous section and often reuse symbols defined there.

Uncontrolled and controlled propositions. The set of *uncontrolled* propositions is $\mathcal{X} = P_{\varphi_{\mathcal{S}}} \cup \mathcal{X}_T \cup \mathcal{X}_{req}$, where (i) $P_{\varphi_{\mathcal{S}}}$ is as above; (ii) \mathcal{X}_T contains one proposition for each program state t , denoting the current state of \mathcal{T} ; and (iii) $\mathcal{X}_{req} = \{req_{\psi}^{\phi} \mid \langle t, \psi, \phi, t' \rangle \in \delta\}$ contains one proposition for each program transition: req_{ψ}^{ϕ} states that the agent, according to program \mathcal{T} , is currently requesting to achieve ϕ while maintaining ψ .

The set of *controlled* propositions is $\mathcal{Y} = P_A \cup \{last\}$, where P_A is as above and proposition *last* states that last action of current plan is to be executed next (then, after execution, the agent can issue a new request).

Assumption formula. The assumption formula is $\varphi_a = \varphi_a^{init} \wedge \square \varphi_a^{trans} \wedge \varphi_a^{rc}$. For legibility, we define for each program state $t \in T$, a propositional formula $req_t = \bigvee_{\langle t, \psi, \phi, t' \rangle \in \delta} req_{\psi}^{\phi}$ representing the fact that the agent is requesting at least one transition available in program state t .

Propositional formula $\varphi_a^{init} = \varphi_{\mathcal{S}}^{init} \wedge t_0$ characterizes the (legal) initial state of the overall system. Note no constraints on *last* nor on propositions in \mathcal{X}_{req} are imposed.

LTL formula $\varphi_a^{trans} = \varphi_{\mathcal{S}}^{trans} \wedge \varphi_{\mathcal{T}}^{trans}$ characterizes the assumptions on the overall evolution. In particular, $\varphi_{\mathcal{T}}^{trans}$ is the same as that in $\varphi_{\mathcal{S}}$ of the previous section, while $\varphi_{\mathcal{T}}^{trans}$ defines the “transition rules” for the planning program, built as the conjunction of the following formulae:

- $\bigvee_{t \in \mathcal{X}_T} [t \wedge \bigwedge_{t' \in \mathcal{X}_T \setminus \{t\}} \neg t']$, that is, the program is in exactly one of its states;
- $\bigwedge_{t \in \mathcal{X}_T} [t \rightarrow req_t]$, that is, in each state, the agent executing the program ought to be requesting some of the possible transitions available in its current state;
- $\bigwedge_{req_{\psi}^{\phi}, req_{\psi'}^{\phi'} \in \mathcal{X}_{req}, req_{\psi}^{\phi} \neq req_{\psi'}^{\phi'}} [req_{\psi}^{\phi} \rightarrow \neg req_{\psi'}^{\phi'}]$, that is, at most one program transition can be requested at a time;
- $\bigwedge_{\langle t, \psi, \phi, t' \rangle \in \delta} [t \wedge req_{\psi}^{\phi} \wedge last \rightarrow \bigcirc t']$, that is, if transition $t \xrightarrow{\psi, \phi} t'$ is currently being requested and the last action of current plan is to be executed next, then the program shall move next to its successor state t' ;
- $\bigwedge_{t \in \mathcal{X}_T} [(t \wedge \neg last) \rightarrow \bigcirc t]$, that is, the program remains still if the current plan is still not completed (new requests are not allowed if the latest is not fulfilled);
- $\bigwedge_{t \in \mathcal{X}_T, \langle t, \psi, \phi, t \rangle \in \delta} [(t \wedge req_{\psi}^{\phi} \wedge \neg last) \rightarrow \bigcirc req_{\psi}^{\phi}]$, that is, the agent remains requesting the same transition if the current plan is still not completed.

Finally, $\varphi_a^{rc} = \varphi_{\mathcal{S}}^{rc}$, where $\varphi_{\mathcal{S}}^{rc}$ includes the reduction of strong fairness constraints into weak fairness ones, as defined in the previous section.

Requirement formula. Now, we build formula $\varphi_r = \Box \varphi_r^{trans} \wedge \varphi_r^{goal}$, which captures the requirements for the realization (i.e., requirements on function ω).

LTL formula $\varphi_r^{trans} = \varphi_{act}^{trans} \wedge \varphi_{last}^{trans} \wedge \varphi_{maint}^{trans}$ encodes the constraints on action executions and how planning program transitions are successfully carried out:

- φ_{act}^{trans} requires exactly one domain action to be executed per step (see previous section);
- $\varphi_{last}^{trans} = \bigwedge_{req_\psi^\phi \in \mathcal{X}_{req}} [req_\psi^\phi \wedge last \rightarrow \bigcirc \phi]$, that is, upon plan completion, requested achievement goal ϕ is indeed achieved;
- $\varphi_{maint}^{trans} = \bigwedge_{req_\psi^\phi \in \mathcal{X}_{req}} [req_\psi^\phi \rightarrow \psi]$, that is, maintenance goal ψ in current request is respected.

Finally, we encode the synthesis “objective” by means of just one weak fairness conjunct: $\varphi_{goal}^{req} = \Box \Diamond last$.

The following result comes by comparing the above construction with the definition of planning program realization.

Theorem 7 (Soundness & Completeness). *There exists a realization of an agent planning program \mathcal{T} over a dynamic system \mathcal{S} iff LTL formula Θ built as above is realizable.*

Clearly, Θ is a GR(1) formula, hence by reasoning analogously to Theorem 6 we get:

Theorem 8 (Complexity). *Checking the existence of a realization of an agent planning program over a dynamic domain under strong fairness constraints is EXPTIME-complete.*

Again, the technique proposed actually synthesizes the realization of the planning programs during the checking.

Example 4. Consider an extension the production line scenario, in which items arrive on request and can be placed in one of three locations: on a workbench, to be processed; in a storage, to be sent to the assemblage tape; or in a garbage area, to be disposed of. The PDDL fragment reported in Figure 2, describes the domain using the following predicates: *dusty* and *greasy*, stating whether the current item is dusty and greasy, respectively; *onWb*, *stored* and *disposed*, capturing the fact that the current item is on the workbench, in the storage or in the garbage area, respectively; and *grabbed*, stating whether the arm is holding an item. Initially, the workbench is empty and the arm not holding anything.

The domain provides actions to manipulate the items. Action *load* places a new item in the workbench, which is required to be empty. Actions *sprayAir* and *spraySol* work as in Example 3, except that they require the item to be on the workbench. Actions *store* and *dispose* move the current item to the storage or the garbage area, respectively, and can only be executed if the arm holds the item, which is achieved by executing action *grab*, executable when the item is on the workbench and the arm is not holding anything. Effectiveness of *sprayAir* and *spraySol* is captured by the very same fairness constraints as in Example 3.

The items preparation process is captured by the planning program depicted in Figure 3. Maintenance goals are omitted as all \top . Initially, in state t_0 , the program requires a new item to be loaded on the workbench. Then, from state t_1 ,

```
(define (domain productionLine2)
  (:predicates (dusty) (greasy) (onWb) (stored)
              (disposed) (grabbed))

  (:action load
   :precondition (not (onWb))
   :effect (and (onWb)
                (not (stored)) (not (disposed))
                (oneof (dusty) (not (dusty)))
                (oneof (greasy) (not (greasy)))))

  (:action sprayAir
   :precondition (onWb)
   :effect (and
            (when (not (dusty)) (not (dusty)))
            (when (not (greasy)) (not (greasy)))
            (when (dusty) (oneof (dusty) (not (dusty))))
            (when (greasy) (oneof (greasy) (not (greasy)))))

  (:action spraySol ... ) ;analogous to sprayAir

  (:action grab
   :precondition (and (onWb) (not (grabbed)))
   :effect (and (not (onWb)) (grabbed)))

  (:action store
   :precondition (grabbed)
   :effect (and (not (onWb)) (stored)
                (not (grabbed))))

  (:action dispose
   :precondition (grabbed)
   :effect (and (not (onWb)) (disposed)
                (not (grabbed))))

  (:init (not (onWb)) (not (grabbed))))
```

Figure 2: Planning domain for the production line example.

there are two possible choices: either the item is not stored (possibly because it is damaged) or it is cleaned and then stored for the assemblage stage—the choice is under the autonomous agent running the program. Finally, the routine starts again from state t_0 for processing a new item.

The planning program has a realization, which is as follows. Transition $t_0 \xrightarrow{onWb} t_1$ is served by executing *load*, thus obtaining a new item, possibly dusty and greasy, on the workbench. Next, if $t_1 \xrightarrow{\neg stored} t_0$ is requested, it is served by the sequence *grab*; *dispose*. Observe that, in principle, the transition could be realized by simply leaving the item on the workbench. However, that will preclude the next request $t_0 \xrightarrow{onWb} t_1$ to be realized, as the workbench needs to be empty to load a new item. If transition $t_1 \xrightarrow{\neg dusty \wedge \neg greasy} t_2$ is requested instead, it can be served by iterating actions *sprayAir* and *spraySol* as necessary (cf. Example 3), from where next request $t_2 \xrightarrow{stored} t_0$ can simply be achieved by the sequence *grab*; *store*. ■

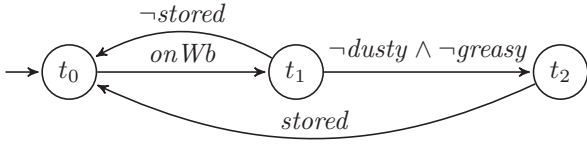


Figure 3: Planning program for the item preparation routine

Component-based Planning

Up to now, we have assumed that actions are always available, of course as long as their preconditions are fulfilled. However, it is often the case that actions can be carried out *only* through certain available actuators, such as a gripper, a motor, or a web-browser. Also, these actuators generally have their own internal logic that needs to be respected and even *local* actions (e.g., a camera needs to be turned on before a picture can be taken). To account for this, in this section, we consider planning in the presence of fairness constraints *but where actions are available only by means of a set of so-called “available behaviors.”* We point out that the new task is similar to that of behavior composition (De Giacomo and Sardina 2007; Sardina, Patrizi, and De Giacomo 2008) and that of agent planning programming (De Giacomo, Patrizi, and Sardina 2010), except for two main differences. First, here, we are interested in solving a standard planning problem, rather than implementing a desired target system. More importantly, the representations that we shall use here for behaviors is substantially more expressive than the ones used in such works, in that strong fairness constraints will be used.

So, besides the dynamic system $\mathcal{S} = \langle \mathcal{D}, \mathcal{C} \rangle$, as before, we assume also a set of available *behaviors* modeling the components at disposal (typically, physical devices or software modules). A representation for such behaviors needs to capture the following features: first, behaviors’ logic may depend on properties of the domain (e.g., an arm can grab a block if not holding anything); second, different choices may be available when activating the behavior (e.g., at some point the arm may be used to move a block or to flip it). Finally, being *abstractions* of actual components, the representation needs to be able to accommodate lack of information about the internals of the components.

Formally, a *behavior* over a dynamic system \mathcal{S} (with set of states Σ) is a tuple $\mathcal{B} = \langle B, O, b_0, G, \varrho, C_{\mathcal{B}} \rangle$, where:

- B is the finite set of behavior’s states;
- O is the finite set of behavior’s actions s.t. $O \cap A \neq \emptyset$;
- $b_0 \in B$ is the behavior’s initial state;
- G is a set of *guards*, that is, boolean functions of the form $g : \Sigma \mapsto \{\top, \perp\}$;
- $\varrho \subseteq B \times G \times O \times B$ is the \mathcal{B} ’s transition relation. We freely interchange notations $\langle b, g, a, b' \rangle \in \varrho$ and $b \xrightarrow{\langle g, a \rangle} b'$ in \mathcal{B} ;
- $C_{\mathcal{B}}$ is a finite set of strong fairness constraints over the set $B \cup \{act = o \mid o \in O\}$.

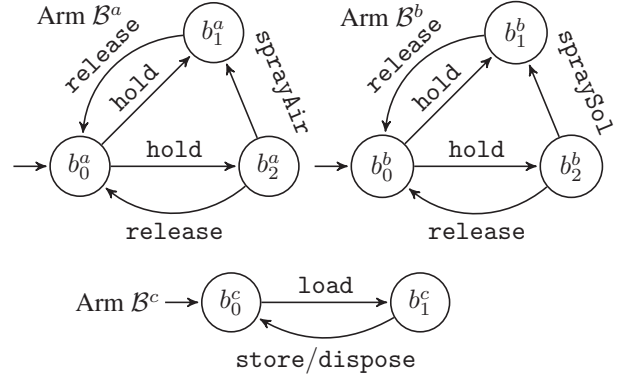


Figure 4: The three different arms available in the domain.

When a behavior is in a certain state, it can be instructed to perform any of the actions available from that state due to an (outgoing) transition whose guard holds true. Observe that behaviors are, in general, nondeterministic, that is, given a state and an action, there may be *several* transitions whose guards evaluate to \top . Consequently, when choosing the action to execute next, one cannot be certain of the resulting state, and hence of which actions will be available later on, since this depends on what particular transition happens to take place. In other words, nondeterministic behaviors are only *partially controllable*, which captures the fact that one has *incomplete* information on the component’s functioning. Finally, since behaviors can include their own *local* actions, that is actions in O that are not in A , it is assumed that those actions are always possible in the dynamic system and have no effect on its state.

Example 5. Continuing our production line example, we now imagine that we cannot act in the world if not by using the three available robotic arms depicted in Figure 4. All actions are consequences of any of these arms. We observe that arms \mathcal{B}^a and \mathcal{B}^b are nondeterministic when trying to hold a component item. This is because the arm may fail to properly hold the item, falling then into state b_1^a and b_1^b , respectively. In that case, the arms cannot do their actual job—spraying air or solvent—and all they can do is to just release the item. Only when the devices are successful in holding the item, reaching thus states b_2^a and b_2^b , can the item be sprayed (and afterwards finally released). Finally, arm \mathcal{B}^c is capable of loading items into the workbench and unload items by storing or disposing them. For simplicity, we assume no guards in behaviors transitions, though, it is easy to imagine arms that do require some condition to be true in the world to be able to perform an action (e.g., arm \mathcal{B}^c may only be able to load items of no more than a certain weight).

Now, consider again the nondeterminism of arms \mathcal{B}^a and \mathcal{B}^b when it comes to hold an item for spraying. Under this setting, the original goal of preparing each item for the assemblage process cannot be guaranteed anymore. The reason is that even though one can use arms \mathcal{B}^a and \mathcal{B}^b to spray the items to remove their dust and grease, it could happen that one of the arms can never succeed grabbing the item.

Technically, this means that when requesting action `hold` in, say, arm \mathcal{B}^a , its actual evolution always results in state b_1^a from where it is not possible to spray the item with air. Nonetheless, the domain expert knows that while arms can indeed fail to properly hold an item at some point, they will eventually succeed with enough tries. To accommodate this domain information we include the following strong fairness constraints for behaviors \mathcal{B}^a and \mathcal{B}^b :

$$\begin{aligned} \Box\Diamond(b_0^a \wedge \text{act} = \text{hold}) &\rightarrow \Box\Diamond b_2^a \text{ (for behavior arm } \mathcal{B}^a\text{);} \\ \Box\Diamond(b_0^b \wedge \text{act} = \text{hold}) &\rightarrow \Box\Diamond b_2^b \text{ (for behavior arm } \mathcal{B}^b\text{).} \end{aligned}$$

With this extra domain information, it is now possible to solve the (extended) planning task, namely, it is possible to build a plan π such that $\pi \models \top \mathcal{U}(\neg \text{dusty} \wedge \neg \text{greasy})$ and π relies only on the three arms at disposal. As it can be easily seen, such a plan exists: first, use arm \mathcal{B}^c to load the item; then repetitively use arms \mathcal{B}^a and \mathcal{B}^c to spray air and solvent to the item; finally, once processed the item can be either stored or disposed using arm \mathcal{B}^c one again.

We note two important points next. Firstly, in contrast with the solutions in previous sections, one cannot spray an item several times in a row anymore. Instead, each spray should be preceded by a holding operation and followed by a releasing: this are constraints coming from the available actuators, not the actual environment. Secondly, in a successful run, there could be several failed tries of cleaning a block, that is, actions `hold` followed immediately by actions `release`. However, due to the strong fairness constraint, we know that by trying enough times, the arm will eventually succeed holding, and thus spraying, the item. ■

As one can clearly see from the example, the possibility of using strong fairness constraints in defining available behaviors is of ultimate importance. Indeed such constraints allow the domain expert to accommodate more expressive kind of incomplete information that could have a big impact in the overall problem. Note that in, e.g., (De Giacomo and Sardina 2007; Sardina, Patrizi, and De Giacomo 2008; De Giacomo, Patrizi, and Sardina 2010), such types of constraints are not accounted for, and the problem described in the above example would yield no solution.

From Behavior Programs to Richer Domains

The problem now is how to solve a planning problem by acting in the domain *only through the available behaviors*.

In a similar way as done in (De Giacomo, Patrizi, and Sardina 2010), we can reduce the component-based planning task to the original component-free problem. To that end, we show below how to suitably embed the behavior descriptions into a dynamic system.

Let $\mathcal{B}_1, \dots, \mathcal{B}_n$, with $\mathcal{B}_i = \langle B_i, O_i, b_{i0}, G_i, \varrho_i, C_i \rangle$, be the set of available behaviors over a dynamic system $\mathcal{S} = \langle \langle P, \Sigma, A, S_0, \rho \rangle, C \rangle$. For each \mathcal{B}_i , let \overline{B}_i be a tightest set of boolean propositions that is large enough to provide a binary encoding of \mathcal{B}_i states (clearly, $|\overline{B}_i|$ is logarithmic in $|B_i|$). We assume all \overline{B}_i 's pairwise disjoint and disjoint from P . We represent the encoding of a generic element $b \in B_i$, as $\overline{b} \in 2^{\overline{B}_i}$, under the assumption that \overline{b} contains all and only B_i 's propositions evaluating to \top in the encoding of b .

To combine all \mathcal{B}_i 's and \mathcal{D} into a planning domain, we build a new dynamic system $\widehat{\mathcal{S}} = \langle \langle \widehat{P}, \widehat{\Sigma}, \widehat{A}, \widehat{S}_0, \widehat{\rho} \rangle, \widehat{C} \rangle$, where:

1. $\widehat{P} = P \cup \overline{B}_1 \cup \dots \cup \overline{B}_n$ is the set of (extended) domain propositions;
2. $\widehat{\Sigma} = 2^{\widehat{P}}$ is the set of $\widehat{\mathcal{S}}$ states. We denote each state $\widehat{S} \in \widehat{\Sigma}$ as $\widehat{S} = S \cup \overline{b}_1 \cup \dots \cup \overline{b}_n$, making explicit the components S , representing the state of the original domain \mathcal{D} , and \overline{b}_i representing the state of behavior \mathcal{B}_i ($i = 1, \dots, n$);
3. $\widehat{S}_0 = S_0 \cup \overline{b}_{10} \cup \dots \cup \overline{b}_{n0}$, that is, initially \mathcal{D} is in its initial state and so is each available behavior;
4. $\widehat{A} = \bigcup_{i=1}^n \{ \langle a, i \rangle \mid a \in O_i \}$, i.e., all domain actions are made available through behaviors. Observe that \widehat{A} also contains all behaviors' local actions (i.e., those not in A);
5. $\widehat{\rho} \subseteq \widehat{\Sigma} \times \widehat{A} \times \widehat{\Sigma}$ is such that $\langle \widehat{S}, \langle a, i \rangle, \widehat{S}' \rangle \in \widehat{\rho}$, where $\widehat{S} = S \cup \overline{b}_1 \cup \dots \cup \overline{b}_n$ and $\widehat{S}' = S' \cup \overline{b}'_1 \cup \dots \cup \overline{b}'_n$, iff
 - if $a \in A$, then $\langle S, a, S' \rangle \in \rho$, that is, domain action a may evolve the dynamic system \mathcal{D} from S to S' ;
 - if $a \notin A$, then $S = S'$, that is, a is a local action for a behavior and so it has no effects on \mathcal{D} ;
 - there exists a transition $\langle b_i, g, a, b'_i \rangle \in \varrho_i$ such that $g(S) = \top$, that is, action a is indeed enabled in \mathcal{B}_i ;
 - for each $j \neq i$, $\overline{b}'_j = \overline{b}_j$, that is, all non-activated behaviors remain still;
6. \widehat{C} is a set of strong fairness constraint such that $\widehat{c} \in \widehat{C}$ iff
 - there exists a strong fairness constraint $c \in \mathcal{C}$ such that \widehat{c} is obtained by replacing each occurrence of atomic propositions of the form $(\text{act} = a)$ in c with $\bigvee_{i=1}^n (\text{act} = \langle a, i \rangle)$; or
 - for some $i \in \{1, \dots, n\}$, there exists a strong fairness constraint $c \in C_i$ such that \widehat{c} is obtained by replacing: (i) each occurrence of propositions of the form $(\text{act} = a)$ in c with proposition $(\text{act} = \langle a, i \rangle)$; and (ii) each occurrence of (proposition denoting) state $b \in B_i$ with (the conjunction of literals denoting) its encoding \overline{b} .

Informally, the above transformation compiles away all behaviors by encoding them into an extended dynamic system. A state in this extended system encodes not only the state of the original system, but also the state of each available behavior. Moreover, actions in the extended systems include the specific behavior (i.e., its index) where it is carried out.

The main difference with the LTL compilation in (De Giacomo, Patrizi, and Sardina 2010) is that we consider here strong fairness constraints, both of the dynamic domain \mathcal{D} and in available behaviors.⁴

At this point, it is not difficult to see that solving the component-based planning problem of achieving ϕ while

⁴In addition, with respect to (De Giacomo, Patrizi, and Sardina 2010), we directly use an efficient (logarithmic) encoding of behaviors and we avoid introducing the generalized notion of histories and plans to account for behavior delegation. The reason is that such delegation is already included in the action itself in the extended dynamic system.

maintaining ψ in \mathcal{S} by means of operating a set of available behaviors $\mathcal{B}_1, \dots, \mathcal{B}_n$ is equivalent to solving the component-free planning problem in the extended dynamic system $\hat{\mathcal{S}}$, as constructed above. As for complexity, observe that, when n behaviors are present, we get that $|\hat{P}| = O(|P| + n * \log(\max_{i=1}^n |B_i|))$. Recalling that the complexity of planning under strong fairness constraints is exponential in the number of domain propositions (see discussion before Theorem 6), we get that this variant of the problem can also be solved in EXPTIME.

Conclusion

In this work we have tackled strong fairness constraints in planning. Such constraints have a strong modeling power in regulating nondeterminism in nondeterministic domains, and have great theoretical interest, as the ability of expressing them is one of the most distinctive advantages of LTL over CTL in verification. We have shown that quite advanced forms of planning can be dealt with in presence of such constraints, while remaining in the same EXPTIME-complete class of standard conditional planning in nondeterministic domains with full observability (Rintanen 2004). We conclude by stressing that, even if the study in this paper is substantially theoretical, the technique adopted for solving such forms of planning is readily implementable through systems for LTL synthesis, based on model checking game structures, such as TLV⁵, Anzu⁶, and Ratsy⁷.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. Fabio Patrizi is partially funded by the IST Programme of the EU Commission – Project SM4All (FP7-224332). Sebastian Sardina acknowledges the support of Agent Oriented Software and the Australian Research Council (under grant LP0882234).

References

Accellera. 2004. *Property Specification Language Reference Manual*. www.eda.org/vfv/docs/PSL-v1.1.pdf.

Armoni, R.; Fix, L.; Flaisher, A.; Gerth, R.; Ginsburg, B.; Kanza, T.; Landver, A.; Mador-Haim, S.; Singerman, E.; Tiemeyer, A.; Vardi, M. Y.; and Zbar, Y. 2002. The For-Spec temporal logic: A new temporal property-specification language. In *Proceedings of the TACAS'02*, 296–211.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22:5–27.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence Journal* 173(5-6):593–618.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about actions and planning in LTL action theories. In *Proc. of KR'02*, 593–602.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence Journal* 147(1-2):35–84.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model checking*. Cambridge, MA, USA: The MIT Press.

De Giacomo, G., and Sardina, S. 2007. Automatic synthesis of new behaviors from a library of available behaviors. In Veloso, M. M., ed., *Proc. of IJCAI'07*, 1866–1871.

De Giacomo, G., and Vardi, M. Y. 2000. Automata-theoretic approach to planning for temporally extended goals. In *Proc. of ECP'99*, 226–238.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2010. Agent programming via planning programs. In *Proc. of AAMAS'10*. To appear.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Kerjean, S.; Kabanza, F.; St-Denis, R.; and Thiébaux, S. 2006. Analyzing LTL model checking techniques for plan synthesis and controller synthesis (work in progress). *Electronic Notes in Theoretical Comp. Science* 149(2):91–104.

Kesten, Y.; Piterman, N.; and Pnueli, A. 2005. Bridging the gap between fair simulation and trace inclusion. *Information and Computation* 200(1):35 – 61.

Kupferman, O.; Piterman, N.; and Vardi, M. Y. 2006. Safrless compositional synthesis. In *Proc. of CAV'06*, 31–44.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of reactive(1) designs. In *Proc. of VMCAI'06*, volume 3855 of *Lecture Notes in Computer Science (LNCS)*, 364–380. Springer.

Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *Proc. of POPL'89*, 179–190.

Rintanen, J. 2004. Complexity of planning with partial observability. In *Proc. of ICAPS'04*.

Sardina, S.; De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2006. On the Limits of Planning over Belief States. In *Proc. of KR'06*, 463–471.

Sardina, S.; Patrizi, F.; and De Giacomo, G. 2008. Behavior composition in the presence of failure. In *Proc. of KR'08*, 640–650.

Vardi, M. Y. 1996. An Automata-Theoretic Approach to Linear Temporal Logic. In *Logics for Concurrency: Structure versus Automata*. Springer.

Vardi, M. Y. 2007. Automata-theoretic model checking revisited. In *Proc. VMCAI'07*, 137–150.

⁵www.cs.nyu.edu/acsys/tlv/

⁶www.ist.tugraz.at/staff/jobstmann/anzu/

⁷rat.fbk.eu/ratsy/