Towards Fully Observable Non-Deterministic Planning as Assumption-Based Automatic Synthesis

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

Nicolas D’Ippolito  
Departamento de Computación, FCEN  
Universidad de Buenos Aires  
Buenos Aires, Argentina  
ndippolito@dc.uba.ar

Abstract

Whereas previous work on non-deterministic planning has focused on characterizing (and computing) “loopy” but “closed” plans, we look here at the kind of environments that these plans are to be executed in. In particular, we provide a logical characterization of the standard “fairness” assumption used, and show that strong cyclic plans are correct solution concepts for fair environments. We argue then that such logical characterization allows us to recast non-deterministic planning as a reactive synthesis task, and show that for a special case, recent efficient synthesis techniques can be applied.

1 Introduction

In this paper, we relate advanced forms of planning to the general long-standing Computer Science problem of automatic synthesis [Manna and Waldinger, 1987; Abadi et al., 1989; Pnueli and Rosner, 1989]: the problem of automatically building an executable piece of code from (high-level) user intent. Despite planning and synthesis not being areas too far from each other, the connection between the two worlds have not been much explored. We are not in fact the first in noticing that such a relation exists, in the planning community others have pointed it out; see [Pistore and Traverso, 2001]. Still, as no (complete) formalization has been presented, the relation is only informally analyzed. We concentrate in this work on fully observable non-deterministic (FOND) planning, a type of planning that has recently attracted much attention and which aims to accommodate events outside the control of the agent [Daniele et al., 2000]. In FOND planning, actions are non-deterministic, in that their execution yields one of a set of possible effects, and this is Nature’s choice. Once the effect has ensued, however, the agent is able to observe it.

Strong cyclic plans—those that re-try until success is obtained—have arguably become the de-facto solution concept for FOND planning. Daniele et al. [2000] provided its first characterization in GtL, as those plans for which “in any of its executions, it is always the case that the goal is always reachable.” This was later refined in [Pistore and Traverso, 2001; Cimatti et al., 2003] to disregard what happens after goal achievement. Then, with the advancement of planning and model checking technologies, several promising techniques and systems have emerged for solving FOND planning problems, such as PRP [Muise et al., 2012], NDP [Kuter et al., 2008], and FIP [Fu et al., 2011], MBP [Cimatti et al., 2003], and GRENDEL [Ramirez and Sardina, 2014]. The first three are built on top of classical planners, whereas the last two perform fix-point reasoning via model-checking type techniques. Nonetheless, the point is that they all amount to specialized—clever though often involved—algorithms for constructing “loopy” but “closed” plans.

In this work, we revisit what planning under non-deterministic actions is with the aim of framing it as a special case of controller synthesis [Pnueli and Rosner, 1989]. The study of both novel controller synthesis and planning techniques could lead to interesting developments in the planning community by taking advantages of the expressiveness and guarantees provided by the controller synthesis approaches. Moreover, whereas control synthesis was deemed computationally impossible in the past, recent approaches (e.g., [Bloem et al., 2012; D’Ippolito et al., 2011]) have shown that for some quite expressive specifications, the task is amenable for computation. In turn, planning problem specifications can inform meaningful specifications—for goals and assumptions—to the reactive synthesis field.

It turns out that to make such link evident, one needs to formalize the assumptions on the environments in which strong cyclic plans are guaranteed to eventually achieve their goals. Surprisingly, this aspect has merely been discussed at an informal level, by calling for “fair” environments, that is, those in which “all actions’ outcomes will ensue infinitely often, if tried infinitely often.” To settle that formally, we provide a logical characterization of the environment which complements that of “good” plans found in [Pistore and Traverso, 2001]. We show, with an example, that a naive interpretation of the fairness assumption does not yield the expected results, and provide a characterization that accounts for “failure independence.” We prove that strong cyclic plans are indeed sound and complete solution concepts under such characterization. Finally, we argue that, by using the characterization developed, reactive synthesis [Pnueli and Rosner, 1989] can be directly used to solve FOND planning tasks, and show that, for the special case in which actions have “intended effects,” existing effective synthesis techniques can be exploited.
2 Preliminaries

2.1 Fully Observable Non–Deterministic Planning

We mostly follow the characterisation of non-deterministic planning given in [Rintanen, 2008], as it provides a more formal framework than others to work on. However, such account is indeed equivalent to the usual “oneof” clauses in PDDL based characterisations [Gerevini et al., 2006].

A fully observable non-deterministic (FOND) planning problem is a tuple \( P = (P, O, s_1, o_{goal}) \) consisting of a set of Boolean state propositions \( P \) (atoms), an initial state \( s_1 \), a goal \( o_{goal} \) as a conjunction of literals (i.e., atoms or negated atoms), and an operator set \( O \) (see below). We use \( l \) to denote the complement of literal \( l \).

A state \( s \) is a consistent set (or conjunction) of literals such that \( |s| = |P| \)—every atom is either true or false. We use \( S \) to denote the set of all states of task \( P \).

An operator is a pair \( o = (Pre_o, Eff_o) \), where \( Pre_o \) is a condition describing the preconditions of operator \( o \), and \( Eff_o = e_1 \mid \cdots \mid e_n \) the (non-deterministic) effects of \( o \), where each \( e_i \) is a (deterministic effect) condition and \( n \geq 1 \). The intended meaning is that one of the \( e_i \) events ensure non-deterministically, by the environment’s choice. Operator \( o \) is executable on a state \( s \) if \( s \models Pre_o \). The successor states resulting from executing operator \( o \) in state \( s \) is defined as \( next(o, s) = [Eff_o]s \), where:

\[
[e_1 \mid \cdots \mid e_n] = \bigcup_{i=1}^{n} \left\{ (s \setminus \{ l \mid e_i = l \}) \cup \{ l \mid e_i = \bar{l} \} \right\}.
\]

In general, \([Eff_o]s\) yields a set of states (one per effect of \( o \)); if \( o \) is deterministic (i.e., \( Eff_o = \epsilon \)), then \([Eff_o]s = 1\).

A policy (or conditional plan) is a function \( \pi : S \rightarrow 2^O \) mapping states \( s \in S \) onto the set of executable operators \( \pi(s) \) such that if \( o \in \pi(s) \), then \( s \models Pre_o \). The universal policy for a FOND problem \( P \) is \( \pi(s) = \{ o \mid o \in O, s \models Pre_o \} \). A policy \( \pi \) executed from state \( s \) defines a set of possible executions \( \Delta(s) \) made up of executions \( \lambda = s_0o_0s_1 \cdots s_i o_is_{i+1} \cdots \), where \( s_0 = s \), \( o_1 \in \pi(s_1) \), \( s_1 \models Pre_{o_1} \), and \( s_{i+1} \in next(o_is_i) \), for all \( i \geq 0 \). The set of states relevant to a policy \( \pi \) from state \( s \) is defined as \( S_{\pi}(s) = \bigcup_{\lambda \in \Delta(s)} \{ s_i \mid s_i \in \lambda \} \) (we abuse notation and write \( s \in \lambda \) to say that execution \( \lambda \) mentions state \( s \)). A policy \( \pi \) is closed iff \( \bigcup_{\lambda \in \Delta(s)} next(o_is_i) \subseteq S_{\pi}(s) \) for all states \( s_i \in S_{\pi}(s) \), that is, \( \pi \) always returns an action for every non-goal state it reaches. In turn, a state \( s \) is reachable by a policy \( \pi \) from state \( s' \) if there is a chance that following \( \pi \) leads the agent to \( s \); formally, there exists \( \lambda \in S_{\pi}(s') \) such that \( s \in \lambda \).

When it comes to FOND planning, the usual solution concept in the literature is that of strong cyclic plans.

Definition 1 (Strong Cyclic Plan [Cimatti et al., 2003]). A strong cyclic plan is a closed policy such that a goal state is reachable from every reachable state using the policy.

Strong policies are a special case of strong cyclic policies, for which all executions are finite and acyclic: they solve the planning problem in a bounded number of steps. We close by noting that non-deterministic planning with full observability is EXPTIME-complete [Rintanen, 2004].

2.2 Labelled Transition Systems

Labelled Transition Systems (LTS) [Keller, 1976] are operational models that describe behaviour in terms of the occurrence of actions and their effect on the system’s state. LTSs are widely used in many domains, particularly, as description language for event-based controllers and controller synthesis.

Definition 2 (Labelled Transition Systems). A labelled transition system (LTS) is a tuple \( E = (S, A, \Delta, s_0) \), where \( S \) is a finite set of states, \( A \) is its communicating alphabet, \( \Delta \subseteq (S \times A \times S) \) is a transition relation, and \( s_0 \in S \) is the initial state. We use \( \Delta(s) = \{ \ell \mid \langle s, \ell, s' \rangle \in \Delta \} \) to denote the set of enabled actions in state \( s \). A sequence \( t = \ell_0\ell_1 \cdots \) is a trace in \( E \) if there exists a sequence \( s_0\ell_0s_1\ell_1 \cdots \) such that \( (s_i, \ell_i, s_{i+1}) \in \Delta \) for all \( i \geq 0 \).

The parallel composition of two LTSs \( E_1 = (S_1, A_1, \Delta_1, s_{10}) \) and \( E_2 = (S_2, A_2, \Delta_2, s_{20}) \) is obtained from the synchronous product of \( E_1 \) and \( E_2 \), and encodes the concurrent execution of \( E_1 \) and \( E_2 \) by interleaving non-shared actions but forcing synchronisation on shared actions [Hoare, 1978].

2.3 Temporal Logics

Temporal logics are logic formalisms tailored for statements and reasoning which involve the notion of order in time. They allow the formal specification of behavioral properties of systems [Pnueli, 1977] and are the basis for model checking and synthesis. Some widely known and used such logics are CTL* and its fragments CTL and LTL [Emerson and Halpern, 1986; Clarke and Emerson, 1982]. In particular, CTL has been proved to be a convenient framework for describing the notion of strong cyclic plans in a precise and concise manner. The syntax of CTL* is defined with the following grammar:

\[
\Psi ::= p \mid \Psi_1 \wedge \Psi_2 \mid \neg \Psi \mid A\varphi \mid E\varphi; \\
\varphi ::= \Psi \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 U\varphi_2 \mid \varphi_1 W\varphi_2,
\]

where \( p \) ranges over the set of propositions. Formulas of the form \( \Psi \) are called state formulas, whereas those of the form \( \varphi \) are said to be path formulas. Formula \( A\Psi \) (\( E\Psi \)) states that all executions (some execution) from the current state satisfy property \( \Psi \). Then, path formulas \( X\varphi \), \( F\varphi \), and \( G\varphi \) state that \( \varphi \) is true in the next state of the path, eventually in the path, or always along the path, resp. Finally, \( U\varphi_1\varphi_2 \) says that \( \varphi_1 \) holds along the path until \( \varphi_2 \) becomes true and \( \varphi_1 \) is eventually true; \( W\varphi_2 \) is its weak version where \( \varphi_2 \) is not required to eventually come true in the path. Common combinations of path and state quantifiers allow us to say things like all or some next states satisfy \( \varphi \) (\( AX\varphi \) and \( EX\varphi \)), and \( \varphi \) holds in all or some executions (\( AG\varphi \) or \( EG\varphi \)).

The meaning of such CTL* formulas is given over the states and paths of a transition system, with a branching-time interpretation of time. Concretely, a CTL* transition system over a set of propositions \( P \), also called Kripke structure, is a tuple \( K = (W, R, P) \), where:

- \( W \subseteq 2^P \) is the set of states of \( K \); and
- \( R \subseteq W \times W \) is the transition relation of \( K \) such that for every \( w \in W \), there exists \( w' \in W \) such that \( R(w, w') \).

When \( R(w_1, w_2) \) holds, it means that state \( w_2 \) is a possible successor of state \( w_1 \).
A run in $K$ from state $w_0$ is an infinite sequence $\pi = w_0w_1w_2\cdots$ such that $R(w_i, w_{i+1})$, for all $i \geq 0$. Given a state formula $\Psi$ and a state $w \in W$, we can define whether $\Psi$ holds true in structure $K$ at state $w$, denoted $K, w \models \Psi$. Finally, LTL is the fragment obtained from CTL* by withholding the $A$ and $E$ quantifiers and assuming universal path quantification. CTL, in turn, is the CTL* fragment obtained by requiring that each temporal path quantifiers (e.g., $X$, $F$, etc.) be under the immediate scope of an $A$ or $E$ quantifier.

A variant of LTL used in the Software Engineering context (from where we will import certain results) is FLTL (fluent-LTL) [Giannakopoulou and Magee, 2003], which is more tailored towards reasoning about fluents via automata-like structures with emphasis on transitions’ labels rather than states’ labels. A fluent $f = \langle I_f, T_f, init_f \rangle$ is defined by sets of disjoint initiating and terminating actions $I_f$ and $T_f$ from a universal set of actions $Act$, resp., and an initial value $init_f$.

Given a set of fluents, the truth of an FLTL temporal formula is defined relative to an infinite trace $t = a_0a_1\cdots$ over the universal set of actions $Act$. If $t$ is an infinite trace, the satisfaction of an FLTL formula $\varphi$ at position $i$, denoted $t, i \models \varphi$, is standard and omitted. However, we just point out that since FLTL traces focus on action/events (i.e., transitions labels, rather than states’ labels), the truth of a fluent $f$ at a certain prefix $t^i$ of trace $t = a_0a_1\cdots$ is defined as: $t, i \models f$ if and only if one of the following conditions holds: (i) $\text{init}_f \land (\forall j.0 \leq j < i \implies a_j \not\in T_f)$; and (ii) $\exists j.(j \leq i \land a_j \in I_f) \land (\forall k.0 < k \leq i \implies a_k \not\in T_f)$.

We say that $\varphi$ holds in $t$, denoted $t, 0 \models \varphi$, if $t, 0 \models \varphi$. Finally, a FLTL formula $\varphi$ holds in an LTS $E$ (denoted $E, 0 \models \varphi$) iff $t, 0 \models \varphi$, for all infinite traces $t$ of $E$.

3 Strong-cyclic Plans and Fair Environments

In this section, we revisit the semantics of FOND planning, with special focus on the “environment” where plans are to be executed. In particular, whereas the notion of strong cyclic plans have been vastly studied and formally defined, the type of environments in which such plans will succeed are often discussed at an informal level.

The first precise analysis of what a solution is for FOND planning was given by Daniele et al. [1999; 2000]. There, the authors formally defined strong cyclic plans through a CTL formula on their executions. To capture all executions of a policy, they defined what is basically the projection of the underlying state model for a planning task onto the policy.

Definition 3 ($\langle \mathcal{P}, \pi \rangle$-structure). Let $\mathcal{P} = \langle P, O, s_I, \phi_o \rangle$ be a FOND planning problem over the set of propositions $P$ and $\pi$ a policy over $\mathcal{P}$. The induced structure of $\langle \mathcal{P}, \pi \rangle$ is a Kripke structure $K_\mathcal{P} = \langle W, R, P \rangle$, where:

- $W = \{ \langle s, o \rangle \mid s \in 2^P, o \in O, s \models Pre_o \}$. Intuitively, $\langle s, o \rangle$ represents the execution of operator $o$ in state $s$;
- $R(\langle s, o \rangle, \langle s', o' \rangle)$ iff $o \in \pi(s)$ and $s' \in next(o, s)$.

Structure $K_\mathcal{P}$ represents all evolutions of policy $\pi$ when executed in planning domain $\mathcal{P}$. It is straightforward to define a structure representing all possible executions of the planning domain, by considering the universal policy.

Definition 4 ($\mathcal{P}$-structure). The structure induced by a FOND planning problem $\mathcal{P}$ is defined as the Kripke structure $K_\mathcal{P} = K_\mathcal{P}^\pi$, where $\pi$ stands for the universal policy.

So, Daniele et al. [1999; 2000] defined a policy $\pi$ as a strong cyclic solution for a planning problem $\mathcal{P}$ iff $K_\mathcal{P}^\pi, (s_I, o_I) \models AG\varphi_{goal}$, for all $o_I \in \pi(s_I)$. In words, starting from the initial state, whatever actions we choose to execute (from the policy) and whatever their outcomes are, we always (AG) have a way of reaching the goal ($\varphi_{goal}$).

Because goals are achievement goals in planning, what happens after the goal is achieved is irrelevant. Hence, Daniele et al.’s definition was then further refined as follows:

Definition 5 ([Pistore and Traverso, 2001]). A policy $\pi$ is a strong cyclic plan solution for a FOND planning problem $\mathcal{P}$ iff $K_\mathcal{P}^\pi, (s_I, o_I) \models A(\varphi_{goal}W\varphi_{goal})$, for all $o_I \in \pi(s_I)$.

In words, in all possible executions of the policy, the goal is always eventually reachable, at least until the goal is indeed reached. Observe that this definition precisely captures the notion of closeness and goal-reachable (as in Definition 1).

The availability of a precise notion for strong-cyclic plans has facilitated the development of various techniques capable of synthesising such type of plans. Planning systems like MBP [Cimatti et al., 2003], PRP [Muir et al., 2012], FIP [Fu et al., 2011], and GRENDEL [Ramirez and Sardina, 2014] all search for policies that are closed and goal-reachable.

Nevertheless, besides understanding what a strong-cyclic plan amounts to, we argue here that it is also important to understand and formalize the contexts under which these type of plans will indeed achieve the objectives, namely, bringing about the goal. This has not received much attention and has mostly been discussed informally.

Clearly, under non-determinism, there is in principle the possibility of never achieving the goal, as “strong cyclic solutions can produce executions that loop forever” (without ever reaching the goal) [Cimatti et al., 2003]. However, the common understanding in the literature is that strong cyclic plans are adequate solutions under the assumption that the underlying environment described by the planning domain is “fair.”

Unlike strong cyclic plans, which have been formally characterized (see above), this fairness assumption is always stated at an informal level. We shall provide a precise characterization of such fairness in LTL and show that strong cyclic plans are sound and complete plan types for fair domains.

A usual (informal) understanding is that fair domains are those for which “if an action is executed infinitely many times, every non-deterministic outcome will occur infinitely often.” Formally, this can be stated as the following strong-fairness LTL constraint (on the induced structure $K_\mathcal{P}$):

$$\Phi_{\mathcal{P}}^{\mathcal{P}} \triangleq \bigwedge_{o \in O, e \in Eff_o} (GFo \rightarrow GF\varphi).$$

Note that while strong cyclic plans have been characterized in CTL (Definition 5), $\Phi_{\mathcal{P}}^{\mathcal{P}}$ above is an LTL formula not expressible in CTL. While simple, D’Ippolito et al. [2011] argued, in the context of reactive synthesis for Software Engineering, that such strong-fairness assumptions are not enough to guarantee success of controllers that are meant to “re-try.” Consider the following counter example.
Example 1. Imagine the problem of opening a door that has some operational defects, by inserting the key and then turning the knob. However, both actions may fail non-deterministically and not achieve their objectives. When that happens, the agent has to take the key out (which can only be done with the knob turned) and start all over again. Thus, consider planning problem $P_{safe} = \{P,O,s_1,\phi_{goal}\}$, where:

- $P$ includes the operators:
  - INSERT = \{-kn, k| kkn \land kStuck\},
  - TURN = \{kn \land \neg turned, open \land turned | turned\},
  - REMOVE = turned, \neg kn \land \neg kStuck \land open \land \neg turned.
- $O$ includes the operators:

Taking next the following (partial) policy $\pi$: (i) if $s = \neg kn$, then perform INSERT; (ii) if $s = \neg turned \land kn$, then perform TURN; and (iii) if $s = turned$, then perform REMOVE.

It is not hard to see that $\pi$ is a strong cyclic solution: it continuously tries to first insert and use the key, and then turn the knob and remove the key, until success is achieved.

Now, the formula $\Phi_{\pi_{safe}}^s$, as defined above, implies that if the agent tries to use the key infinitely often, then she will succeed infinitely often. Similarly, if the agent keeps turning the knob, then she will keep succeeding getting the door open. It turns out, though, that:

$$K_{\pi_{safe}}^s(s_1, INSERT) \not\models A[\Phi_{\pi_{safe}}^s \rightarrow F(open \land \neg kStuck)].$$

That is, there is an execution of $\pi$ in $P$ in which all outcomes of INSERT and TURN arise infinitely often, but it is always the case that either the door is closed or the key stuck. In fact, consider the following states of $K_{\pi_{safe}}^s$: (i) $w_0 = (s_1, INSERT)$; (ii) $w_1 = (kn, kStuck, TURN)$; (iii) $w_2 = (kn, turned, REMOVE)$; and (iv) $w_3 = (kn, TURN)$. Then, run $\lambda = (w_0, w_1, w_2, w_3, w_4)$ does satisfy the assumption formula $\Phi_{\pi_{safe}}^s$, but it never reaches the goal. Intuitively, the failure of the action TURN is not independent of failure of previous action INSERT—their failures are “synchronized.”

The example above shows that the failures and successes of two non-deterministic actions can be “coordinated in a way that will preclude goal achievability, despite the fact that all outcomes of both actions ensue infinitely often.

To rule out coordinated outcomes D’Ippolito et al. [2011] provided the stronger notion of t-strong fair environments, but their framework assumes that it is known which action outcomes are “good” (and which ones are considered “failures”). Still, the intuition there is to have action-fairness in every state, as captured by the following definition.

Definition 6 (State Strong Fairness; t-strong fair). Let $P$ be a planning problem and $K_P = \langle W,R,P \rangle$ its corresponding Kripke structure. The state strong, or simply t-strong, fairness constraint is defined as the following LTL formula:

$$\gamma_P^{\text{sfair}} = \bigwedge_{\{s,o,e\}} (GF(s \land o) \rightarrow GF[\phi]_s).$$

A run of $\lambda$ of $K_P$ is t-strong fair if $\lambda \models \gamma_P^{\text{sfair}}$. In words, $\gamma_P^{\text{sfair}}$ states that if an operator is executed infinitely often in the same state, then all its effects ought to arise infinitely often. This LTL constraint formalizes Cimatti et al., 2003’s claim that unfair executions are those where “some actions are executed infinitely often in given states, but some of its outcomes never occur.” and corresponds to the semantically expressed requirement used in various FOND planning works (e.g., [Patrizi et al., 2013; Ramirez and Saracino, 2014; Geffner and Bonet, 2013]).

Returning to our example above, the run $\lambda$ does not satisfy $\gamma_P^{\text{sfair}}$ and $K_P \models A[\gamma_P^{\text{sfair}} \rightarrow F(\phi_{goal})]$ holds.

We now have all the machinery to state the main contribution of this section, namely, linking state strong fairness with Pistore and Traverso [2001]’s account of strong cyclic plans (see Definition 5). First we show that all such plans do achieve the goal in state strong fair environments.

Theorem 1. Let $\pi$ be a strong cyclic plan for a FOND planning problem $\mathcal{P}$. Then, $K_{\pi}^s, (s_1, o_1) \models A[\gamma_P^{\text{sfair}} \rightarrow F(\phi_{goal})]$, for all (initial) operators $o_1 \in \pi(s_1)$.

This theorem provides a “soundness” result for state strong fair environments. More importantly and less obvious, if a plan guarantees the goal in a state strong fair environment, then it has to be strongly cyclic.

Theorem 2. Let $\pi$ be a policy and $\mathcal{P}$ a FOND problem such that $K_{\pi}^s, (s_1, o_1) \models A[\gamma_P^{\text{sfair}} \rightarrow F(\phi_{goal})]$, for all (initial) operators $o_1 \in \pi(s_1)$. Then, $\pi$ is a strong cyclic plan.

The above two theorems say that state strong fairness is a complete characterization of the type of environment for which strong cyclic plans are successful. We note that, together, the above theorems are closely related to Theorem 2 in [Patrizi et al., 2013], though ours crystalize the assumption-solution relationship directly within Traverso et al.’s FOND planning foundational framework. Having a precise characterization of what adequate environments are opens the door for applying (reactive) synthesis techniques to solve FOND problems. That is the topic of the next section.

4 FOND Planning via Reactive Synthesis

Reactive synthesis or controller synthesis studies techniques that given a model of the assumed behaviour of the environment ($E$) and a system goal ($G$), produces an operational behaviour model for a component $M$ that when, executed in an environment consistent with the assumptions, results in a system that is guaranteed to satisfy the goal, namely, $M \models G$.

Arguably, the application of controller synthesis as means for solving planning problems has been discouraged due to the high complexity of the general approach. Indeed, controller synthesis is, in general, computationally expensive: synthesising for LTL goals is 2EXPTIME complete and known techniques have resisted practical implementations due to complementation [Pnueli and Rosner, 1989; Kupferman et al., 2006]. However, recent advances (e.g., [Bloem et al., 2012; D’Ippolito et al., 2011]) may, under appropriate assumptions, provide efficient solutions to planning problems. The fact is that restricting the form of the specification system goal allows for more effective solutions and im-
implementable systems. For instance, goal specifications consisting uniquely of safety requirements can be solved in polynomial time, and so can particular, yet expressive, subsets of liveness properties such as GR(1) [Piterman et al., 2006].

As stated in Section 3, FOND strong cyclic plans are guaranteed to achieve the goals in t-strong fair environments. Following [Bloem et al., 2012] and relying on Theorems 3 and 4, one can solve a FOND planning task \( P \) with goal \( \phi_{goal} \) by checking LTL realizability of specification formula

\[
\varphi_P \equiv \varphi_P^1 \land [\varphi_P^2 \rightarrow (\gamma_P \rightarrow F \phi_{goal})],
\]

where \( \varphi_P^1 \) and \( \varphi_P^2 \) are simple (safety) LTL formulas encoding structure \( K_P \). Unfortunately, \( \varphi_P \) results in an expression that does not allow for efficient synthesis techniques, as it does not fall into known efficient LTL fragments like GR(1). Hence, in general, there is still an efficiency challenge when solving FOND planning via controller synthesis.

In some cases, however, it is possible to reduce the planning problem to an equivalent efficient controller synthesis one. For example, De Giacomo et al. [2010a] do so for standard conditional planning (i.e., planning for strong solutions). Here, we discuss a special such case for FOND planning, namely, when every action of a non-deterministic planning problem can be annotated with its “expected” or “intended” effect—that is, actions have a clear clear fail/success semantics. For instance, in the example in Section 3, the expected effect of turning the knob is to successfully open the door. All other effects can be considered failures. Arguably, many meaningful FOND planning problems come with a clear success/failure semantics of actions.

In what follows, we show how efficient reactive synthesis techniques developed in the area of Software Engineering can be imported to solve this type of planning problems. Specifically, we rely on D’Ippolito et al. [2011]’s polynomial approach to building LTS controllers for control problems with explicit description of failures against GR(1)-like specifications. Failures are defined by introducing try-response sets that relate the attempt to perform a controlled action and the corresponding environment’s reaction. Technically, a try-response triple is a tuple \( t = (a, suc_a, fail_a) \) where \( a \) and \( suc_a \) are actions in the environment model representing an action and its intended result, respectively, and \( fail_a \) is a set of actions representing failures of \( a \)’s execution. In planning terms, a try would be an operator, while the responses would its effects. A key aspect of the approach is the Strong Independent Fairness (SIF) condition, which provides restrictions on how failures behave [D’Ippolito et al., 2011, Def. 4.3]. Intuitively, SIF requires a windows of opportunity (“gaps”) in which all expected effects are executed without any failures. It turns out that the SIF assumption allows to reduce the problem of having gaps with no failures to one in which failures are assumed to occur finitely.

When interpreting failures as operator’s effects, most of the required restrictions for try-response triples are inherently guaranteed (e.g., re-tries cannot occur before a response), except for one restriction that needs to be lifted. While [D’Ippolito et al., 2011] assumes simple action representation for failures and successes, here we need failures to be a set of actions—the set of non-intended effects. However, since their approach reduces to solving a control problem where failures occur only finitely often, having multiple failures for the same try action does not affect key fact that effective controllers (re)try actions knowing that success will eventually occur. For lack of space we omit the technical details of such generalization to set of failures.

Next, we formally define try-response sets for a given LTS restricting attention to the requirements described above.

**Definition 7 (Try-Response).** Let \( E = (S, A, \Delta, s_0) \) be an LTS and \( A_c \subseteq A \) be the set of controllable actions. Set \( T \) is a try-response set for \( E \) if:

1. \( (a, suc_a, fail_a) \in T \) iff \( a \in A_c, suc_a, fail_a \in A \setminus A_c, and suc_a \sqsubseteq \Delta, suc_a, fail_a \in A \setminus A_c, for some k \geq 1. \)
2. For all \( t_a = (a, suc_a, fail_a), t_b = (b, suc_b, fail_b) \in T, if a = b then \( t_a = t_b \).
3. \( E \models \mathcal{G}[\bigwedge_{1 \leq i \leq k}\{\text{fail}_i \lor suc_a\}] \leftrightarrow a \), for all \( a \in A_c \).
4. For all \( s \in S \) and \( try_a \in A_c, fail_a \) is enabled from \( s \) iff \( suc_a \) is enabled from \( s \).

We note that this definition is a special case of that in [D’Ippolito et al., 2011], tailored to the fact that outcomes ensue immediately after operators’ executions. In their work, they also need to deal with delayed failures/successes.

Now, the specific control problem we are interested in is defined with an LTS environment model, a GR(1)-like specification, a set of controllable actions, and a try-response set.

**Definition 8 (Recurrent Success LTS Control).** A recurrent success LTS control (RSGR) problem is a tuple \( E = (E, A_c, \varphi, T) \) where \( E \) is an LTS, \( A_c \) is a set of controllable actions, \( \varphi \) is a GR(1) FLTL formula, and \( T \) is a try-response set for \( E \).

A solution for \( E \) is an LTS \( M \) with controlled actions \( A_c \) such that \( E \) only disables controllable actions, \( E|M \) is deadlock free, and every trace \( \pi \) in \( M|E \) is such that \( \pi \models \varphi \).

That is, a solution to an RSGR control problem is an LTS controller that, in parallel composition with the environment model, meets the system specification goal. Thanks to the assumed failure semantics, such controller can be computed in polynomial time via a reduction to a GR(1) synthesis task [D’Ippolito et al., 2011].

So, let us next show how to reduce a FOND planning problem with intended effects to an RSGR control problem.

**Definition 9 (FOND to RSGR).** Let \( P = (P, O, s_f, \phi_{goal}) \) be a FOND planning problem, such that each operator \( o \in O \) is defined as \( o = (Pre_o, Eff_o, Int_o) \) where \( Int_o \in Eff_o \) stands for the intended effect of operator \( o \). The corresponding RSGR problem \( E_P = (E, A_c, \varphi, T) \) is defined as follows (here \( Eff(O) = \{ e \mid o \in O, e \in Eff_o \} \) is all the effects of \( O \)):

- \( A_c = O; \)
- \( T = \{ (o, Int_o, E_o) \mid o \in O and E_o = Eff_o \setminus Int_o \}; \) and
- \( \varphi = F(\land \in \theta_{goal} f_1) \), where:

\[
f_1 = \{ (e \in Eff(O) \mid e \models t_1), \{ e \in Eff(O) \mid e \models \bot \};
\]
Proof Sketch. A trace of $E$ is an interleaving of sequences of operators and effects (from where one can reconstruct the events that hold true at any point). Observe that a controller for an RSGR control problem only restricts controllable actions (in our case, planning operators). Also, as we deal with reachability goals, it does not require additional memory: effectively, it is a subgraph of the environment model. This last property enables us to produce a strong cyclic policy from it.

**Definition 10 (LTS to Policy).** Given a planning problem $\mathcal{P} = \langle P, O, s_I, \phi_{goal} \rangle$, and an LTS $M = (S, A, \Delta, s_0)$ solution for the RSGR control problem $\mathcal{E}_P$, and $\pi_{M}$ the policy obtained from $M$. Then, $K_{\mathcal{P}}^{\pi_{M}}, \langle s_I, o_1 \rangle \models A(\mathcal{E}\phi_{goal}W\phi_{goal})$, for all $o_1 \in \pi_{M}(s_I)$.

**Proof Sketch.** Suppose $K_{\mathcal{P}}^{\pi_{M}}, \langle s_I, o_1 \rangle \not\models A(\mathcal{E}\phi_{goal}W\phi_{goal})$. Hence, there is a state $s$ in $K_{\mathcal{P}}^{\pi_{M}}$ such that every execution $\lambda$ in $K_{\mathcal{P}}^{\pi_{M}}$ that visits $s$ (i.e., $s \in \lambda$) is such that $\lambda \not\models \phi_{goal}$. By construction, $s$ also belongs to $M$. Further, all traces in $M$ that visit $s$ must also avoid the goal—i.e., never satisfy the FLTL formula $(\bigwedge_{t \in C(\phi_{goal})} F_t)$. In particular, every trace $t$ satisfying $\text{SIF}$ condition that visits $s$ will not satisfy the goal (i.e. $t \not\models F(\bigwedge_{t \in C(\phi_{goal})} F_t)$), which contradicts our hypothesis.

A direct consequence of Theorems 1 and 3 is that $K_{\mathcal{P}}^{\pi} \models A[\gamma_{\mathcal{P}} \rightarrow F\phi_{goal}]$. Secondly, if the FOND problem has a solution that is based on the operators’ intended effects, then it is accounted in the solution to the corresponding RSGR problem. Intuitively, a policy is intention-based if it can reach the goal relying (eventually) only on the intended effects of operators—i.e., it achieves the goal “under no failures.”

**Definition 11 (Intention-Based Policy).** Let $\mathcal{P}$ be a FOND planning problem with goal $\phi_{goal}$ and operators of the form $o = (P_{\phi_{eff}}, E_{\phi_{eff}}, I_{\phi_{int}})$, where $I_{\phi_{int}} \in E_{\phi_{eff}}$ stands for the intended effect of $o$. A strong cyclic policy $\pi$ for $\mathcal{P}$ is intention-based iff for all $w_1 \in \{s_I\} \times \pi(s_I)$:

$$K_{\mathcal{P}}, w_1 \models A[F(\bigwedge_{s_0 \in O} G(s \land o) \rightarrow \lceil I_{\phi_{int}} \rceil_s) \rightarrow F\phi_{goal}]$$

**Theorem 4 (Completeness).** Let $\mathcal{P}$ be a problem, and $\mathcal{E}_P$ its corresponding RSGR control problem. If an intention-based strong cyclic policy $\pi$ for $\mathcal{P}$ exists, then there exists a solution $M$ for $\mathcal{P}$ such that $\pi \subseteq \pi_{M}$.

**Proof Sketch.** We construct an LTS $M$ from policy $\pi$ and show that $M$ is a solution to $\mathcal{E}_P$. Let $M = (S, A, \Delta_M, s_0)$ be an LTS defined as follows: $s_0 = s_I$; $S = 2^P \cup (2^P \times O)$; $A = \{e | O, e \in Eff_o \} \cup O$; and $\Delta = \{ (s, o, (s, o)) | o = \pi(s) \} \cup \{ (s, o, s') | e_o \in Eff_o, s' \in \lceil e_o \rceil_s \}$. Now, the proof amounts to proving that for all traces $t$ of $M \models E_{P}$, if $t$ is SIF, then $t \models F(\bigwedge_{t \in C(\phi_{goal})} F_t)$. Since $t$ is SIF, we know that $t$ exhibits long enough gaps showing all intended effects of operators and no failures. Consider the prefix $t'$ of $t$ that includes the first such a gap. There exists an execution $\lambda_{t'}$ of policy $\pi$ corresponding to prefix $t'$, which is “intended base” and hence achieves the goal. Therefore, prefix $t'$ also achieves the goal specification and $t \models F(\bigwedge_{t \in C(\phi_{goal})} F_t)$ applies.

Now, observe that [D’Ippolito et al., 2011] proved that solving a RSGR control problem can be done polynomially, by reducing it to a GR(1) synthesis task [Bloem et al., 2012]. Since the encoding of a FOND problem into a RSGR control problem (Definition 9) is exponential, we get that we can use reactive synthesis to obtain intended-based strong cyclic solution in exponential time (which is the complexity of conditional planning). In addition, unlike standard solutions via specialized FOND algorithms, the policy is “maximal,” in that it accounts for every initial state for which a solution exists.

**5 Conclusions**

In this paper we formally studied the kind of environments in which strong cyclic plans are adequate, thus complementing the existing formal characterization of such plans in CTL and contributing to bridging the gap between automated planning and reactive synthesis. We showed that an adequate fairness assumption should encode the independence of different nondeterministic steps, and provided a characterization of such environments in CTL∗/LTL. We argued that doing so allows for direct specification of the FOND planning task into a reactive synthesis one. While the general synthesis task is not amenable for efficient computation, we showed a special case of FOND problems—those in which actions have intended effects—for which low complexity techniques are available.

We believe that making the connection between planning and controller synthesis more evident can benefit both areas. Automated planning can exploit powerful techniques—like synthesis of GR(1) or safety specifications—and open for more general planning settings. As pointed out in [De Giacomo et al., 2010b], one could be interested in planning under “selective” and/or “conditional” fairness assumptions (e.g., the dice is fair unless an action has “loaded” it), something that specialized planners cannot handle. Indeed, the special case of actions with intended effects is one such case. For reactive synthesis, planning settings can provide concrete applications and inform what types of system goal specifications are meaningful and worth studying.

In future work, we plan to investigate other solution concepts for non-deterministic planning. In particular, we are interested in fault-tolerant planning [Domshlak, 2013].

**Acknowledgments**

We thank the anonymous reviewers for their very helpful comments. We acknowledge the support of the Australian Research Council under a Discovery Project (DP120100332), and of projects ANPCYT PICT 2012-
References


