Journal of Artificial Intelligence Research 74 (2022) 887–916 Submitted 12/2021; published 06/2022

FOND Planning with Explicit Fairness Assumptions

Ivan D. Rodriguez IVANDANIELRA@GMAIL.COM
Blai Bonet BONETBLAI@GMAIL.COM
Universitat Pompeu Fabra, Barcelona, Spain

Sebastian Sardina SEBASTIAN.SARDINA@RMIT.EDU.AU
RMIT University, Melbourne, Australia

Hector Geffner HECTOR.GEFFNER@UPF.EDU
Universitat Pompeu Fabra, Barcelona, Spain
Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain
Linköping University, Linköping, Sweden

Abstract

We consider the problem of reaching a propositional goal condition in fully-observable non-deterministic (FOND) planning under a general class of fairness assumptions that are given explicitly. The fairness assumptions are of the form A/B and say that state trajectories that contain infinite occurrences of an action a from A in a state s and finite occurrence of actions from B, must also contain infinite occurrences of action a in s followed by each one of its possible outcomes. The infinite trajectories that violate this condition are deemed as *unfair,* and the solutions are policies for which *all the fair trajectories reach a goal state.* We show that strong and strong-cyclic FOND planning, as well as QNP planning, a planning model introduced recently for generalized planning, are all special cases of FOND planning with fairness assumptions of this form which can also be combined. FOND$^+$ planning, as this form of planning is called, combines the syntax of FOND planning with some of the versatility of LTL for expressing fairness constraints. A sound and complete FOND$^+$ planner is implemented by reducing FOND$^+$ planning to answer set programs, and its performance is evaluated in comparison with FOND and QNP planners, and LTL synthesis tools. Two other FOND$^+$ planners are introduced as well which are more scalable but are not complete.

1. Introduction

FOND planning is planning with fully observable, non-deterministic state models specified in compact form where a goal state is to be reached (Cimatti, Pistore, Roveri, & Traverso, 2003). In its most common variant, strong-cyclic planning, one is interested in policies that reach states from which the goal can be reached following the policy (Cimatti, Roveri, & Traverso, 1998; Daniele, Traverso, & Vardi, 1999). In another common variant, strong planning (Cimatti, Roveri, & Traverso, 1998), one is interested in policies that reach a goal state in a bounded number of steps. Each form of FOND planning is adequate under a suitable *fairness* assumption; in the case of strong planning, that non-determinism is adversarial (or “unfair”); in the case of strong-cyclic planning, that non-determinism is fair, in that none of the possible outcomes of a non-deterministic action can be skipped forever.[1]

FOND planning has become increasingly important as a way of solving other types of problems such as *probabilistic (MDP) planning*, where actions have a probabilistic effect on states (Bert-

1. In Section 2, we elaborate on the distinction between adversarial and strong-cyclic planning.

sekas & Tsitsiklis, 1996; Geffner & Bonet, 2013), *LTL planning*, where goals to be reached are generalized to temporal conditions that must be satisfied possibly by plans with cycles (Calvanese, De Giacomo, & Vardi, 2002; Camacho, Bienvenu, & McIlraith, 2019; Aminof, De Giacomo, Murano, & Rubin, 2019), and *generalized planning*, where plans are not for single instances but for collections of instances (Srivastava, Immerman, & Zilberstein, 2011a; Hu & De Giacomo, 2011), and they can be obtained from suitable abstractions encoded as Qualitative Numerical Planning (QNP) problems (Srivastava, Zilberstein, Immerman, & Geffner, 2011b; Bonet & Geffner, 2020).

A critical limitation of strong, strong-cyclic, and QNP planners, is that the fairness assumptions are implicit in their models and solvers, and as a result, cannot be combined. These combinations, however, are often needed (Camacho & McIlraith, 2016; Ciolek, D'Ippolito, Pozanco, & Sardiña, 2020), and indeed, a recent FOND planner handles combinations of fair and adversarial actions in what is called *Dual FOND planning* (Geffner & Geffner, 2018). In this work, we go beyond this integration by also enabling the representation and combination of the conditional fairness assumptions that underlie QNP planning. This is achieved by extending FOND planning with a general class of fairness assumptions that are given explicitly as part of the problem. The fairness assumptions are pairs A/B of sets of actions A and B that say that state trajectories that contain infinite occurrences of actions a from A in a state s, and finite occurrences of actions from B, must also contain infinite occurrences of action a in the state s followed by each one of its possible outcomes. The infinite trajectories that violate this condition are regarded as *unfair.* The solutions of a FOND problem with conditional fairness assumptions of this type, called a FOND$^+$ problem, are *the policies for which all fair state trajectories reach the goal.*

We show that strong, strong-cyclic, and QNP planning, are all special cases of FOND$^+$ planning where the fairness assumptions underlying these models can be combined. FOND$^+$ planning extends the syntax and semantics of FOND planning with some of the versatility of the LTL language for expressing fairness constraints. The conditional fairness assumptions A/B correspond to the LTL formulas $(\Box\Diamond(s \wedge a) \wedge (\neg\Box\Diamond\bigvee_{b\in B} b)) \supset \bigwedge_i \Box\Diamond(a \wedge s \wedge \bigcirc E_i)$, one for each action $a \in A$, each state s, and each possible outcome E_i of the action a, where s stands for the conjunction of literals that s makes true.[2] However, unlike LTL synthesis and planning that are 2EXP-Complete (Pnueli & Rosner, 1989; Camacho et al., 2019; Aminof, De Giacomo, & Rubin, 2020), FOND$^+$ planning is in NEXP (non-deterministic exponential time).

A planner for FOND$^+$ is obtained by reducing FOND$^+$ planning over the explicit state space to an elegant answer set program (ASP), a convenient and high-level alternative to SAT (Brewka, Eiter, & Truszczyński, 2011; Lifschitz, 2019; Gebser, Kaminski, Kaufmann, & Schaub, 2012), using the facilities provided by the CLINGO ASP solver (Gebser, Kaminski, Kaufmann, & Schaub, 2019). The performance of this ASP-based planner is evaluated in comparison with FOND and QNP planners, and LTL synthesis tools.

This paper extends the conference paper of Rodriguez, Bonet, Sardiña, and Geffner (2021) with proofs, additional examples, and two new FOND$^+$ planners that scale up better but are incomplete. The paper is organized as follows. We review first strong and strong-cyclic FOND planning, and

2. For state s and action a in A, the LTL formula (implication) $(\Box\Diamond(a \wedge s) \wedge \neg(\Box\Diamond\bigvee_{b\in B} b)) \supset \bigwedge_i \Box\Diamond(a \wedge s \wedge \bigcirc E_i)$ holds at time (index) k in trajectory τ when either the antecedent does not hold or the consequent holds. The formula is constructed with the LTL connectives "always" ($\Box$), "eventually" ($\Diamond$), and "next" ($\bigcirc$). The antecedent does not hold when the action a is not applied at state s infinitely often (i.e., $\neg\Box\Diamond(a \wedge s)$) or when infinitely often some action in B is applied (i.e., $\Box\Diamond\bigvee_{b\in B} b$). Likewise, the consequent holds when each effect E_i of action a immediately follows the application of a in s infinitely often (i.e., $\bigwedge_i \Box\Diamond(a \wedge s \wedge \bigcirc E_i)$).

QNP planning. We introduce then FOND$^+$ planning, where the assumptions underlying these models are stated explicitly and combined, and present a description of the ASP-based FOND$^+$ planner, an empirical evaluation, some working examples, and the incomplete planners.

2. FOND Planning

A FOND model is a tuple $M = \langle S, s_0, S_G, Act, A, F\rangle$, where S is a finite set of states, $s_0 \in S$ is the initial state, $S_G \subseteq S$ is a non-empty set of goal states, Act is a set of actions, $F(a,s)$ is the set of successor states when action a is executed in state s, and $A(s) \subseteq Act$ is the set of actions applicable in state s, such that $a \in A(s)$ iff $F(a,s) \neq \emptyset$. A FOND problem P is a compact description of a FOND model $M(P)$ in terms of a finite set of atoms, so that the states s in $M(P)$ correspond to truth valuations over the atoms, represented by the set of atoms that are true. The standard syntax for FOND problems is a simple extension of the STRIPS syntax for classical planning. A FOND problem is a tuple $P = \langle At, I, Act, G\rangle$ where At is a set of atoms, $I \subseteq At$ is the set of atoms true in the initial state s_0, G is the set of goal atoms, and Act is a set of actions with atomic preconditions and effects. If E_i represents the set of positive and negative effects of an action in the classical setting, action effects in FOND planning can be deterministic of the form E_i, or non-deterministic of the form $oneof(E_1, \ldots, E_n)$.

A policy π for a FOND problem P is a partial function mapping *non-goal* states into actions. A policy π for P defines a set of, possibly infinite, compatible state trajectories $s_0, s_1, s_2, \ldots$, also called **π-trajectories**, where $s_{i+1} \in F(a_i, s_i)$ and $a_i = \pi(s_i)$ for $i \geq 0$. A trajectory τ compatible with π is *maximal* if it is infinite, or is finite of the form $\tau = s_0, \ldots, s_n$, for some $n \geq 0$, and either s_n is the first state in the sequence being a goal state, $\pi(s_n) \notin A(s_n)$ (i.e., the action prescribed at s_n is not applicable), or $\pi(s_n) = \bot$ (i.e., no action is prescribed). Likewise, the policy π reaches a state s if there is a π-trajectory $s_0, \ldots, s_n$ where $s = s_n$, and π reaches a state s' from a state s if there is a π-trajectory $s_0, \ldots, s_n$ where $s = s_i$ and $s' = s_j$ for $0 \leq i \leq j \leq n$. A state s is **recurrent** in trajectory τ if it appears an infinite number of times in τ. The strong and strong-cyclic solutions or policies are usually defined as follows:

Definition 1 (Solutions)**.** *A policy π is a* ***strong solution*** *for a FOND problem P if all the maximal π-trajectories reach a goal state, and it is a* ***strong-cyclic solution*** *if π reaches a goal state from any state reached by π.*

The strong solutions correspond also to the strong-cyclic solutions that are acyclic; namely, where the policies π do not give rise to π-trajectories that can visit a state more than once. Alternatively, strong and strong-cyclic solutions can be understood in terms suitable notions of fairness that establish which π-trajectories are deemed possible. If we say that *a policy π solves problem P when all the* ***fair*** *π-trajectories reach the goal*, then in strong planning, all π-trajectories are deemed fair, while in strong-cyclic planning, all π-trajectories are deemed fair *except* those containing a recurrent state s that is followed a finite number of times by a successor $s' \in F(\pi(s), s)$.

In order to make this alternative "folk" characterization of strong and strong-cyclic planning explicit, let us say that all the actions in strong FOND planning are **adversarial** (or "unfair"), and that all the actions in strong-cyclic FOND planning are **fair**. The state trajectories that are deemed **fair** in each setting can then be expressed as follows:

Definition 2. *If all the actions are* ***adversarial****, all π-trajectories are* ***fair****. If all the actions are* ***fair****, a π-trajectory τ is* ***fair*** *iff states s that occur an infinite number of times in τ, are followed an infinite number of times by each possible successor s' of s given π, $s' \in F(\pi(s), s)$.*

Provided with these notions of fairness, strong and strong-cyclic solutions can be characterized equivalently as:

Theorem 3. *A policy π is a strong (resp. strong-cyclic) solution of a FOND problem P iff all the maximal* ***fair*** *trajectories compatible with π in P reach the goal, under the assumption that all actions are* ***adversarial*** *(resp.* ***fair****).*

Proof. Under the assumption that all actions are adversarial, the class of π-trajectories coincide with the class of fair π-trajectories. Hence, π is a strong solution according to Def. 1 iff all the maximal fair π-trajectories reach the goal.

For strong-cyclic planning, observe that an infinite π-trajectory τ is *fair* iff τ visits each state that is reachable from any recurrent state in τ. We use this observation to show the contrapositive of the two implications involved in the iff. First, if there is a maximal fair trajectory that does not reach the goal, by the observation, there is a state s that is reachable from the initial state and that is not connected to a goal; i.e., π is not strong-cyclic according to Def. 1. Second, if π is not strong-cyclic, there is a state s that is reachable from the initial state and that is not connected to the goal, and thus there is a maximal fair π-trajectory that does not reach a goal; i.e., π does not solve P. $\square$

Methods for computing strong and strong-cyclic solutions for FOND problems have been developed based on OBDDs (Cimatti et al., 2003), explicit forms of AND/OR search (Mattmüller, Ortlieb, Helmert, & Bercher, 2010), classical planners (Muise, McIlraith, & Beck, 2012), and SAT (Chatterjee, Chmelík, & Davies, 2016). Some of these planners actually handle a **combination** of fair and adversarial actions, in what is called Dual FOND planning (Geffner & Geffner, 2018).

3. QNP Planning

Qualitative numerical planning problems (QNPs) were introduced by Srivastava et al. (2011b) as a model for generalized planning, that is, planning for multiple classical instances at once. QNPs have been used since in other works (Bonet, De Giacomo, Geffner, & Rubin, 2017; Bonet, Frances, & Geffner, 2019) and have been analyzed in depth by Bonet and Geffner (2020).

The syntax of QNPs is an extension of STRIPS problems $P = \langle At, I, O, G\rangle$ with negation where At is a set of ground (Boolean) atoms, I is a maximal consistent set of literals from At describing the initial situation, G is a set of literals describing the goal situation, and O is a set of (ground) actions with precondition and effect literals. A QNP $Q = \langle At, V, I, O, G\rangle$ extends a STRIPS problem with a set V of *numerical variables* X that can be decremented or incremented *qualitatively*; i.e., by indeterminate positive amounts, without making the variables negative. A numerical variable X can appear in action effects as $X\uparrow$ (increments) and $X\downarrow$ (decrements),while literals of the form $X = 0$ or $X > 0$ (an abbreviation of $X \neq 0$) can appear everywhere else (initial situation, preconditions, and goals). The literal $X > 0$ is a precondition of all actions with $X\downarrow$ effects.

A simple example of a QNP is $Q = \langle At, V, I, O, G\rangle$ with $At = \{p\}$, $V = \{n\}$, $I = \{\neg p, n > 0\}$, $G = \{n = 0\}$, and actions $O = \{a, b\}$ given by

$$a = \langle p, n > 0; \neg p, n\downarrow\rangle \quad \text{and} \quad b = \langle \neg p; p\rangle$$

where $\langle C;E\rangle$ denotes an action with preconditions C and effects E. Thus action a decrements n and negates p that is a precondition of a, and b restores p. This QNP represents an abstraction of the problem of clearing a block x in Blocksworld instances with stack/unstack actions that include a block x. The numerical variable n stands for the number of blocks above x, and the Boolean variable p stands for the robot gripper being empty. A policy π that solves Q can be expressed by the rules:

$$\textbf{if } p \text{ and } n > 0, \textbf{ do } a \quad \text{ and } \quad \textbf{if } \neg p \wedge n > 0, \textbf{ do } b\,.$$

A key property of QNPs is that while numerical planning is undecidable (Helmert, 2002), qualitative numerical planning is not. Indeed, a sound and complete, two-step method for solving QNPs was formulated by Srivastava et al. (2011b): the QNP Q is converted into a standard FOND problem $P = T_D(Q)$ and its (strong-cyclic) solutions are checked for termination. The QNP solutions are in correspondence with the **strong-cyclic plans** of the direct translation $P = T_D(Q)$ that **terminate.** Moreover, since the number of policies that solve P is finite, and the termination of each can be verified in finite time, plan existence for QNPs is decidable. More recent work has shown that the complexity of QNP planning is the same as that of FOND planning by introducing a **polynomial reduction** from the former into the latter, and another in the opposite direction (Bonet & Geffner, 2020).

We do not need to get into the formal details of QNPs but it is useful to review the direct translation T_D of a QNP Q into a FOND problem $P = T_D(Q)$, and the notion of termination (Srivastava et al., 2011b). Concretely, the translation T_D replaces each numerical variable n by a Boolean atom p_n that stands for the (Boolean) expression $n = 0$. Then, occurrences of the literal $n = 0$ in the initial situation, action preconditions, and goals are replaced by p_n, while occurrences of the literal $n > 0$ in the same contexts are replaced by $\neg p_n$. Likewise, effects $n\uparrow$ are replaced by effects $\neg p_n$, and effects $n\downarrow$ are replaced by non-deterministic effects $oneof(p_n, \neg p_n)$. Actions in the FOND problem $P = T_D(Q)$ with effects $\neg p_n$ (i.e., $n > 0$) are said to "increment n," while actions with effects $oneof(p_n, \neg p_n)$ (i.e., either $n > 0$ or $n = 0$) are said to "decrement n," even if there are no numerical variables in P but just Boolean variables. This information needs to be preserved in the translation $P = T_D(Q)$, as the semantics of P is not the semantics of FOND problems as assumed by strong or strong-cyclic planners.

4. Termination and SIEVE

A policy π for the FOND problem $P = T_D(Q)$ is said to **terminate** if all the state trajectories in P that are compatible with the policy π and with the fairness assumptions underlying the QNP Q, are finite. Termination is the result of the absence of cycles in the policy that can be traversed forever, under the QNP assumption that numerical values are non-negative and decrements cannot be asymptotically small and tend to zero (Bonet & Geffner, 2020). Thus a cycle that includes an action that decrements a numerical variable and none that increments it must eventually terminate.

The procedure called SIEVE (Srivastava et al., 2011b) provides a **sound and complete termination test** that runs in time that is polynomial in the number of states reached by the policy. SIEVE can be understood as an efficient implementation of the following procedure that operates on the **policy graph** $G(P,\pi)$ determined by a FOND problem P and a policy π, where the nodes are the states s that can be reached in P via the policy π, and the edges correspond to the state transitions (s,s') that are possible given the policy π; i.e., $s' \in F(\pi(s),s)$.

Starting with the graph $G = G(P, \pi)$, SIEVE iteratively removes edges from G until G becomes acyclic or does not admit further removals. In each iteration, an edge (s, s') is removed from G if $\pi(s)$ is an action that decrements a variable x that is not incremented along any path in G from s' back to s. SIEVE **accepts** the policy π iff SIEVE renders the resulting graph G acyclic. It can be shown that the resulting graph G is well defined (i.e., it is the same independently of the order in which edges are removed), and that SIEVE removes an edge (s, s') when it cannot be traversed by the policy an infinite number of times.

It is useful to capture the logic of SIEVE in terms of an **inductive definition** that considers states instead of edges:

Definition 4 (QNP Termination)**.** *Let π be a policy for the FOND problem $P = T_D(Q)$ associated with the QNP Q. The policy π **terminates** in P iff every state s that is reachable by π in P terminates, where a state s **terminates** iff:*[3]

1. *there is no cycle on node s (i.e., no path from s to itself),*
2. *every cycle on s contains a state s' that **terminates**, or*
3. *$\pi(s)$ decrements a variable x, and every cycle on s containing a state s' for which $\pi(s')$ increments x, also contains a state s'' that **terminates**.*

Theorem 5. *Let Q be a QNPs and π a policy. Then,* SIEVE ***accepts** the policy graph $G(P, \pi)$ iff policy π **terminates** in P, where $P = T_D(Q)$.*

Proof. Since the resulting graph of executing SIEVE does not depend on the order in which edges are removed, we can consider any execution of SIEVE.

Forward implication. For a variable x that will be clear from context, let us say that the state s is *forbidden* if $\pi(s)$ increments x. Let $e_0, e_1, \ldots, e_m$ be the sequence of edges removed by SIEVE along some execution. We show by induction that the state s_i in the edge $e_i = (s_i, s'_i)$ removed by SIEVE is terminating. SIEVE removes e_0 because $\pi(s_0)$ decrements a variable x and there is no forbidden state in any path from s'_0 to s_0; i.e., there is no cycle on s_0 that contains a forbidden state. Thus, condition 3 holds trivially and s_0 terminates.

Let us now assume that the claim holds for the first k iterations of SIEVE, and let us consider the $(k+1)$-st iteration. Edge e_{k+1} is removed because $\pi(s_{k+1})$ decrements a variable x and there are no forbidden states on any path from s'_{k+1} to s_{k+1} in the current graph. That is, either there is no such cycle containing s_{k+1} in the original graph, or all cycles that contain forbidden states have been "broken" by the removal of previous edges. In the first case, condition 1 applies and s_{k+1} terminates. In the second case, by inductive hypothesis, all such cycles contain a state s' that terminates, and by condition 3, s_{k+1} terminates as well.

Finally, if the graph resulting from running SIEVE is acyclic, all states that have not yet been labeled as terminating can be labeled as such using conditions 1 or 2. On the other hand, if the resulting graph is not acyclic, it is easy to see that the states in any such cycle cannot be labeled as terminating.

3. This inductive definition and the ones below imply that there is a **unique sequence** of state subsets $S_0, S_1, \ldots, S_k$ such that S_{i+1} is S_i augmented with all the states that can be added to S_i when assuming that the only terminating states are those in S_i.

Backward implication. Let us say that state s is *simple* in a graph if there is no cycle that contains s. Let $S_0, S_1, \ldots, S_k$ be the state subsets associated with the inductive definition (cf. footnote 3). We construct an execution $e_0, e_1, \ldots, e_m$ of SIEVE. If s in S_0, either there is no cycle involving s, or $\pi(s)$ decrements a variable that is not incremented on any cycle that involves s. In the former case, s is simple in the original graph, while in the latter case, SIEVE removes all edges (s, s') that originate at s and thus yields a graph in which s is simple. Hence, after processing S_0, we construct an execution of SIEVE that yields a graph G_0 in which all states in S_0 are simple. Let us assume that we have such an execution that yields a graph G_i where all states in $S_0 \cup \cdots \cup S_i$ are simple. We show that the execution can be extended such that it yields a graph G_{i+1} where all states in $S_0 \cup \cdots \cup S_{i+1}$ are simple. If s in S_{i+1}, either all cycles involving s contain a terminating state, or $\pi(s)$ decrements a variable x and any cycle that contains s and a forbidden state, also contains a state s' that terminates. By inductive hypothesis, in the former case, s is already simple in G_i and remains so. For the latter case, if there is a cycle that contains s and a forbidden state, it also contains a terminating state s', which, by inductive hypothesis, is simple in G_i. Therefore, G_i has no cycle that contains s and a forbidden state, and SIEVE removes all edges that leave s. Finally, observe that the final graph S_k rendered by the constructed execution of SIEVE admits no further edge removals. Indeed, as shown in the previous part, if SIEVE could remove a further edge (s, s'), s would be terminating.

Therefore, to finish, if all reachable states are labeled as terminating, the constructed execution of SIEVE yields a graph G_k where all reachable states are simple; i.e., an acyclic graph. On the other hand, if some reachable state s is not terminating, it is easy to see that G_k has a cycle that contains s and no further edge removal by SIEVE is possible; i.e., SIEVE does not yield an acyclic graph. □

Since solutions to QNPs Q are known to be the strong-cyclic policies of the FOND problem $P = T_D(Q)$ that are accepted by SIEVE (Srivastava et al., 2011b; Bonet & Geffner, 2020), the solutions for Q can also be expressed as:

Theorem 6. *A policy π is a solution to a QNP Q iff π is a strong-cyclic solution of $P = T_D(Q)$ that terminates.*

Proof. The result is direct given that π is a solution of Q iff π is a strong-cyclic solution for P accepted by SIEVE (Srivastava et al., 2011b; Bonet & Geffner, 2020), and the equivalence between SIEVE acceptance and the notion of policy termination in Theorem 5. □

The characterization that results from this theorem has been used to verify QNP solutions but not for *computing* them. Indeed, the only available complete QNP planner is based on a polynomial reduction of QNP planning into strong-cyclic FOND planning that avoids the termination test (Bonet & Geffner, 2020).

5. FOND$^+$ Planning

In this section, we move from strong, strong-cyclic, and QNP planning to the FOND$^+$ setting where the fairness assumptions underlying these models can be explicitly stated and combined. A **FOND$^+$ planning problem** $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of fairness assumptions:

Definition 7. *A FOND$^+$ problem $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of **(conditional) fairness assumptions** of the form A_i/B_i, $i = 1, \ldots, n$ and where each A_i is a set of **non-deterministic actions** in P, and each B_i is a set of actions in P disjoint from A_i.*

The fairness assumptions play no role in constraining the state trajectories that are possible by following a policy π, the so-called π-trajectories:

Definition 8. *A state trajectory compatible with a policy π for the FOND$^+$ problem $P_c = \langle P, C \rangle$ is a state trajectory that is compatible with π in the FOND problem P.*

However, while in strong and strong-cyclic FOND planning all actions are considered as adversarial and fair, respectively, in the FOND$^+$ setting, each action is labeled fair or unfair depending on the assumptions in C and the trajectory where the action occurs. We define what it means for an action $a = \pi(s)$ to behave "fairly" in a recurrent state s of an infinite π-trajectory as follows:

Definition 9. *The occurrence of the action $\pi(s)$ in a recurrent state s of a π-trajectory τ associated with the FOND$^+$ problem $P_c = \langle P, C \rangle$ is **fair** if for some fairness assumption $A/B \in C$, it is the case that $\pi(s) \in A$ and all the actions in B occur finitely often in τ.*

The meaning of a conditional fairness assumption A/B is that the actions $a \in A$ can be assumed to be **fair** in any recurrent state s of a π-trajectory τ, provided that the condition on B holds in τ; namely, that actions in B do *not* occur infinitely often in τ. Otherwise, if any action in B occurs infinitely often in τ, then a is said to be **unfair** or **adversarial**. Once actions $\pi(s)$ occurring in recurrent states s are "labeled" in this way, the standard notion of fair trajectories (Definition 2) extends naturally to FOND$^+$ problems:

Definition 10. *A π-**trajectory** τ for a FOND$^+$ problem $P_c = \langle P, C \rangle$ is **fair** if for every recurrent state s in τ where the action $\pi(s)$ is **fair** and every possible successor s' of s due to action $\pi(s)$ (i.e., $s' \in F(\pi(s), s)$), state s is immediately followed by state s' in τ an infinite number of times.*

The solution of FOND$^+$ problems can then be expressed in a standard way as follows:

Definition 11 (FOND$^+$ Solutions)**.** *A policy π **solves** the FOND$^+$ problem $P_c = \langle P, C \rangle$ if the maximal π-trajectories that are **fair** reach the goal.*

A number of observations can be drawn from these definitions. Let us say that one wants to model a non-deterministic action a whose behavior is **fair** in that it always displays all its possible effects infinitely often in every recurrent state s such that $\pi(s) = a$. To do so, we consider a fairness constraint A/B in C such that $a \in A$ and B is empty. On the other hand, to model an **adversarial** action b, one whose behavior is not fair (may not yield all its effects infinitely often in a recurrent state s with $\pi(s) = b$), we do not include b in any set A. This immediately suggests the way to capture standard strong and strong-cyclic planning as special forms of FOND$^+$ planning:

Theorem 12. *The **strong solutions** of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \emptyset \rangle$.*

Proof. Notice that policies are not defined on goal states, and thus there are no infinite π-trajectories that reach a goal, for any policy π. If π is a strong solution for P, there are no infinite π-trajectories and thus all maximal π-trajectories are finite and goal reaching; i.e., π solves P_c.

On the other hand, if π solves P_c, by definition, the maximal π-trajectories that are fair reach the goal. Since there are no constraints, any infinite π-trajectory is fair and non-goal reaching, and thus, if there is any such trajectory, π cannot solve P_c. Therefore, all maximal π-trajectories are finite and goal reaching; i.e., π is a strong solution for P. $\square$

Theorem 13. *The **strong-cyclic solutions** of a FOND problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \{A/\emptyset\}\rangle$, where A is the set of all the non-deterministic actions in P.*

Proof. Let π be a policy for P and let τ be an infinite π-trajectory in P. Since B is empty, any action $\pi(s)$ in A for a recurrent state s in τ is fair (cf. Definition 9). Thus, by Definition 10 and that A contains all the non-deterministic actions, τ is fair in P iff τ is fair in P_c, and by Theorem 3, π is a strong-cyclic solution for P iff π solves P_c. □

Similarly, QNP problems are reduced to FOND$^+$ problems in a direct way, using both the head A and the condition B in the fairness assumptions A/B in C:

Theorem 14. *The solutions of a **QNP problem** Q are the solutions of the FOND$^+$ problem $P_c = \langle P,C\rangle$ where $P = T_D(Q)$ and C is the set of fairness assumptions A_i/B_i, one for each numerical variable x_i in Q, such that A_i contains all the actions in P that decrement x_i, and B_i contains all the actions in P that increment x_i.*

Proof. Let Q be a QNP problem, let $P_c = \langle P,C\rangle$ be the FOND$^+$ problem associated with Q where $P = T_D(Q)$, and let π be a policy for Q (and P).

Forward direction. Let us assume that π solves Q and suppose it does not solve P_c. Then, there is a maximal fair (cf. Definition 10) π-trajectory τ that does not reach the goal. The trajectory τ must be infinite as otherwise π would not be strong-cyclic. Let R be the set of recurrent states in τ. If s in R and the occurrence of $\pi(s)$ is fair, the state s is followed infinitely often in τ by each $s' \in F(\pi(s),s)$. However, the fairness of $\pi(s)$ implies that there is a fairness assumption A/B in C such that $\pi(s) \in A$, and $\pi(s') \in B$ for no $s' \in R$. This implies that the variable x associated with the assumption A/B reaches the value of zero and τ cannot be infinite. Therefore, the occurrence of $\pi(s)$ for any state s in R must be unfair; namely, for any A/B in C, if $\pi(s) \in A$ then $\pi(s') \in B$ for some $s' \in R$. We have shown that if R contains a state where variable x is decremented, R also contains a state where x is incremented. Since there is at least one state in R where a variable is decremented, no recurrent state in τ is terminating (cf. Definition 4), and thus π does not solve Q. Therefore, π must solve P_c.

Backward direction. Let us assume that π solves P_c and suppose it does not solve Q. Observe that π must be strong-cyclic as otherwise it is easy to construct a maximal, non-goal reaching, and fair π-trajectory in P_c, contradicting the assumption that π solves P_c. Hence, π does not terminate in Q; namely, there is an infinite π-trajectory τ in P such that for the set R of its recurrent states, if $\pi(s)$ decrements a variable x for some $s \in R$, there is another state $s' \in R$ such that $\pi(s')$ increments x. On the other hand, R must contain a state s such that $\pi(s)$ decrements some variable since such actions are the only ones that can generate a cycle in P. Then, by the definition of the fairness assumptions in C, no occurrence of the action $\pi(s)$ for a state s in R is fair. Thus, τ is fair in P_c contradicting the assumption that π solves P_c. Therefore, π must solve Q. □

6. Example: Conditional Fairness

By explicitly stating the fairness assumptions underlying strong, strong-cyclic, and QNP planning, FOND$^+$ planning integrates these planning models as well. We illustrate the new possibilities with an example.

Let P be a FOND problem with state set $\{s_0, s_1, s_2, g\}$, two non-deterministic actions a and b, initial and goal states being s_0 and g, respectively. Action a can only be applied in state s_0, leading

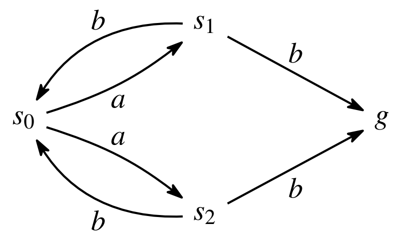
$$\begin{tikzcd}
 & s_1 \arrow[dl, bend right, "b"'] \arrow[dr, "b"] & \\
s_0 \arrow[ur, "a"'] \arrow[dr, "a"] & & g \\
 & s_2 \arrow[ul, bend left, "b"] \arrow[ur, "b"'] &
\end{tikzcd}$$

Figure 1: Example model for FOND problem P with 4 states, non-deterministic actions a and b, and goal state g.

to either s_1 or s_2, whereas action b can be applied only in s_1 and s_2, leading, in both cases, to either s_0 or g; see Figure 1. The FOND problem P admits a single policy, namely, $\pi(s_0) = a$ and $\pi(s_1) = \pi(s_2) = b$, which we analyze in the context of different FOND$^+$ problems $P_i = \langle P, C_i \rangle$ that can be built on top of P using different sets of fairness assumptions C_i. For convenience, in the sets C_i, we use a/b to denote the fairness assumption $\{a\}/\{b\}$, and a to denote the assumption $\{a\}/\emptyset$. The marks '✓' and '✗' express that the policy π solves or does not solve, resp., the FOND$^+$ problem P_i, where C_i is:

✗ $C_1 = \{\}$; a and b are adversarial.

✓ $C_2 = \{a, b\}$; a and b are fair.

✗ $C_3 = \{a\}$; a is fair and b is adversarial.

✓ $C_4 = \{b\}$; b is fair and a is adversarial.

✗ $C_5 = \{a/b\}$; a is conditionally fair on b; b adversarial.

✗ $C_6 = \{a, b/a\}$; QNP like: $a : x_1\downarrow, x_2\uparrow$ and $b : x_2\downarrow$.

✓ $C_7 = \{b, a/b\}$; QNP like: $b : x_1\downarrow, x_2\uparrow$ and $a : x_2\downarrow$.

✗ $C_8 = \{a/b, b/a\}$; QNP like: $a : x_1\downarrow, x_2\uparrow$ and $b : x_2\downarrow, x_1\uparrow$.

The subtle cases are the last four. The policy π does not solve P_5 because there are trajectories like $\tau = s_0, s_1, s_0, s_2, s_0, s_1, s_0, \ldots$ that are fair but do not reach the goal. The reason is that while $a/b \in C_5$, the occurrences of the action $a = \pi(s_0)$ in the recurrent state s_0 in τ are not fair. Thus, both a and b have an adversarial semantics in τ. The policy π does not solve P_6 either, because in the same trajectory τ, the action a is fair in s_0 as $a \in C_6$ but b is not fair in either s_1 or s_2, as the assumption b/a is in C_6 but a occurs infinitely often in τ. As a result, τ is fair but non-goal reaching in P_6. The situation is different in P_7, where b is fair and a is unfair. Here, τ is unfair, as any other trajectory in which some or all the states s_0, s_1, and s_2 occur infinitely often. This is because b being fair in s_1 and s_2 means that the transitions (s_1, g) and (s_2, g) cannot be skipped forever, and the goal must be reached eventually. Finally, in P_8, the trajectory τ becomes fair again, as both a and b are adversarial in τ.

7. Termination and SIEVE$^+$ for FOND$^+$

We now consider the computation of policies for FOND$^+$ problems. Initially, we look for a procedure to verify if a policy π solves a problem $P_c = \langle P, C\rangle$, and then transform this *verification procedure* into a *synthesis procedure*.

The solutions for FOND$^+$ problems are policies that **terminate in the goal**, a termination condition that combines and goes beyond the solution concept for QNPs that only requires goal reachability (strong-cyclicity) and termination (finite trajectories). The termination condition for FOND$^+$ planning can be expressed as follows:

Definition 15 (FOND$^+$ Termination)**.** *Let π be a policy for the FOND$^+$ problem $P_c = \langle P, C\rangle$. State s in P **terminates** (when π is run on P_c) iff*

1. *s is a goal state in P;*
2. *s is **fair** and some successor state $s' \in F(\pi(s), s)$ **terminates**; or*
3. *s is **not fair**, $F(\pi(s), s)$ is non-empty, and all states $s' \in F(\pi(s), s)$ **terminate**;*

*where state s is **fair** if $\pi(s) \in A$ for some assumption A/B in C, and every π-trajectory that connects s to itself and contains a state s' with $\pi(s') \in B$, also contains a state s'' that **terminates**.*

FOND$^+$ termination expresses a procedure similar to SIEVE, that we call SIEVE$^+$, that keeps labeling states s as terminating (the same as removing all edges from s in the policy graph) until no states are left or no more states can be labeled. The key difference with SIEVE is that the removals are done backward from the goal as captured in Definition 15. This is strictly necessary for SIEVE$^+$ to be a sound and complete procedure for FOND$^+$ problems:

Theorem 16. *A policy π solves the FOND$^+$ problem $P_c = \langle P, C\rangle$ iff all the states s that are reachable by π terminate according to Definition 15.*

Proof. Let $S_0, S_1, \ldots$ be the chain of state subsets labeled as terminating corresponding to Definition 15 (cf. footnote 3). For a state subset R, $\pi(R)$ denotes the set $\{\pi(s) : s \in R\}$.

Backward implication. For a proof by contradiction, let us assume that every reachable state terminates, and let us suppose that π does not solve P_c; i.e., there is a *maximal and fair* non-goal reaching π-trajectory τ. We consider two cases.

Case 1: $\tau = s_0, s_1, \ldots, s_n$ is finite. Since τ is maximal and s_n is not a goal, $F(\pi(s_n), s_n)$ is empty. Therefore, s_n cannot terminate contradicting the assumption.

Case 2: τ is infinite. Let R be the set of recurrent states in τ, let j be the minimum index such that $R \cap S_j \neq \emptyset$, and let s be a state in $R \cap S_j$. Clearly, $j > 0$ since S_0 is the set of goal states and τ does not reach such states. We further consider two subcases:

- s is not fair according to Definition 15. Then, every state s' in $F(\pi(s), s)$ must belong to some S_k for $k < j$. This is impossible by the choice of j.
- s is fair according to Definition 15. That is, for some constraint A/B, $\pi(s) \in A$, and if $\pi(R) \cap B \neq \emptyset$, R must contain a state $s'' \in S_k$ for $k < j$. Since the latter is impossible by the choice of j, $\pi(R) \cap B = \emptyset$. This implies that the occurrence of the action $\pi(s)$ is fair in the trajectory τ (cf. Definition 9), and thus that s is followed in τ by each of its possible successors $s' \in F(\pi(s), s)$ infinitely often; i.e., $F(\pi(s), s) \subseteq R$. However, by Definition 15, some such successor s' must terminate (i.e., $s' \in S_k$ for $k < j$), something that is impossible by the choice of j.

Forward implication. For a proof by contradiction, let us assume that π solves P_c, and let us suppose that there is a reachable state s that does not terminate. We are going to construct an infinite π-trajectory τ that is fair and does not reach a goal, contradicting the assumption.

First observe that s is not a goal state and thus it must have successor states. If s is fair (cf. Definition 15), every state $s' \in F(\pi(s),s)$ does not terminate, whereas if s is not fair, some state $s' \in F(\pi(s),s)$ does not terminate. It is then easy to see that we can construct an infinite π-trajectory τ' seeded at s such that for the set R of its recurrent states:

1. no state in R terminates,
2. if s' in R is fair, it is followed infinitely often by each $s'' \in F(\pi(s'),s')$, and
3. if s' in R is not fair, it is followed infinitely often by each $s'' \in F(\pi(s'),s')$ that does not terminate, but there is s'' in $F(\pi(s'),s')$ that does not follows s' infinitely often.

Since the state s is reachable by π, we can construct a π-trajectory τ that reaches s from the initial state s_0 of P and that then follows τ'. The set of recurrent states for τ is the set R of recurrent state for τ'; in particular, R does not contain a goal state. We finish by showing that τ is fair.

We do so using Definitions 9 and 10 for fair actions and fair trajectories respectively. Let s' be a state in R such that the action $\pi(s')$ is fair in τ; in particular, $\pi(s') \in A_i$ for some index i. If the state s' is fair (cf. Definition 15), $F(\pi(s'),s) \subseteq R$ by construction of τ. Else, if s' is not fair, we show below that the occurrence of the action $\pi(s')$ is not fair in τ. Hence, in both cases, the trajectory τ is fair.

For the last bit, by Definition 15, if s' is not fair, there is a cycle τ'' that a) passes over s', b) contains a state s'' with $\pi(s'') \in B_i$, and c) does not contain a terminating state. Hence, by construction of τ, R contains all the states in τ''. However, since there is some state in $F(\pi(s'),s')$ that is not in R, the occurrence of $\pi(s')$ in τ is not fair. □

8. FOND$^+$ and Dual FOND Planning

FOND$^+$ planning subsumes Dual FOND planning where fair and adversarial actions are combined as follows:

Definition 17 (Geffner & Geffner, 2018)**.** *A Dual FOND problem is a FOND problem where the non-deterministic actions are labeled as either* ***fair*** *or* ***adversarial****. A policy π* ***solves*** *a Dual FOND problem P iff for all reachable state s, $\pi(s) \in A(s)$, and there is a function d from* ***reachable states*** *into $\{0,\ldots,|S|\}$ such that*

1. *$d(s) = 0$ for goal states,*
2. *$d(s') < d(s)$ for* ***some*** *$s' \in F(\pi(s),s)$ if $\pi(s)$ is fair, and*
3. *$d(s') < d(s)$ for* ***all*** *$s' \in F(\pi(s),s)$ if $\pi(s)$ is adversarial.*

For showing that the Dual FOND semantics coincides with the semantics of a fragment of FOND$^+$ planning and for capturing the meaning of the $d(\cdot)$ function above, let us recast this definition as a termination procedure:

Definition 18 (Dual FOND Termination)**.** *Let π be a policy for the Dual FOND problem P. A state s in P* ***terminates*** *iff*

1. *s is a goal state,*

2. *$\pi(s)$ is **fair** and some $s' \in F(\pi(s), s)$ **terminates**, or*

3. *$\pi(s)$ is **adversarial**, all states $s' \in F(\pi(s), s)$ **terminate**, and $F(\pi(s), s)$ is non-empty.*

This termination procedure captures indeed the semantics of Dual FOND problems:

Theorem 19. *π is a solution to a Dual FOND problem P iff for every non-goal state s reachable by π, $\pi(s) \in A(s)$ and s terminates according to Definition 18.*

Proof. If π is a solution for P, there is a minimal function d that satisfies Definition 17, and that can be used to construct a chain $S_0, S_1, \ldots$ of state subsets by $S_i = \{s : s \text{ is reachable and } d(s) = i\}$. It is not difficult to check that such a chain corresponds to the unique chain of subsets entailed by Definition 18. Likewise, for a policy π that entails such a chain $S_0, S_1, \ldots$ of terminating states that cover all reachable states, the function d on states, defined by $d(s) = i$ for the minimum i such that $s \in S_i$, satisfies Definition 17. □

The only difference between the termination for Dual FOND and the one for FOND$^+$ (Def. 15) is that in the former the fair and adversarial labels are given, while in the latter they are a function of the explicit fairness assumptions and policy. The function $d(\cdot)$ above captures indeed one ordering in which the states reached by the policy can be labeled as terminating. It is easy to show that Dual FOND problems correspond to the class of FOND$^+$ problems with conditional fairness assumptions A/B with empty conditions B:

Theorem 20. *A policy π solves a Dual FOND problem P' iff π solves the FOND$^+$ problem $P_c = \langle P, C\rangle$ where P is like P' without the action labels, and $C = \{A/B\}$ where A contains all the actions labeled as fair in P', and B is empty.*

Proof. By Theorems 19 and 16, it is enough to show that for every π-reachable and non-goal state s in P', s terminates according to Definition 18 iff s terminates according to Definition 15. Yet, this is direct since for the unique constraint A/B in C, it is easy to see that a state s is fair according to Definition 15 iff $\pi(s)$ is a fair action in P'. With this observation, Definition 18 becomes exactly Definition 15. □

9. FOND-ASP: An ASP-based FOND$^+$ Planner

The characterization of FOND$^+$ planning given in Theorem 16 allows for a transparent and direct implementation of a sound and complete FOND$^+$ planner. For this, the planner *hints* a policy π and then each state reachable by π is checked for termination using Definition 15. The problem of looking for a policy that satisfies this restriction can be expressed in SAT, although we have found it more convenient to express it as an **answer set program**, a convenient and high-level alternative to SAT (Brewka et al., 2011; Lifschitz, 2019; Gebser et al., 2012), using the facilities provided by CLINGO (Gebser et al., 2019).

The code for the back-end of the ASP-based FOND$^+$ planner is shown in Figure 2. The front-end of the planner, not shown, parses an input problem $P_c = \langle P, C\rangle$ and builds a *flat representation* of P_c in terms of a number of ground atoms that are shown in capitalized predicates in the figure. The code in the figure and the facts representing the problem are fed to the ASP solver CLINGO, which either returns a (stable) model for the program or reports that no such model exists. In the former case, a policy that solves P_c is obtained from the atoms `pi(S,A)` made true by the model.[4]

The set of ground atoms providing a flat representation of the problem P_c contains the atoms `STATE(s)`, `ACTION(a)`, and `TRANSITION(s,a,s')` for each (reachable) state s, ground action a and transition $s' \in F(a,s)$ found in a reachability analysis from the initial state s_0. In addition, the set includes the atoms `INITIAL(s0)`, `GOAL(s)` for goal states s, and `ASET(i,a)` and `BSET(i,b)` for a fairness assumption A_i/B_i in C if $a \in A_i$ and $b \in B_i$ respectively.

The program for the FOND$^+$ problem P_c is denoted as $T(P_c)$, while $T(P_c,\pi)$ is used to refer to the program $T(P_c)$ but with the line 2 in Figure 2 replaced by facts `pi(s,a)` when $\pi(s) = a$ for a given policy π, and the integrity constraint in line 28 removed. A model M for $T(P_c)$ encodes a policy π_M where $\pi_M(s) = a$ iff `pi(s,a)` holds in M. The formal properties of the FOND-ASP planner are as follows:

Theorem 21. *Let $P_C = \langle P,C\rangle$ be a FOND$^+$ problem, and let π be a policy for P. Then,*

1. *There is a* ***unique stable model*** *M of $T(P_c,\pi)$, and `terminates(s)` $\in M$ iff s terminates (Definition 15).*

2. *The policy π solves P_c iff the model M for $T(P_c,\pi)$ satisfies the integrity constraint in line 28 in Figure 2.*

3. *M is a model of $T(P_c)$ iff M is the model of $T(P_c,\pi_M)$ and M satisfies the integrity constraints. Thus,* FOND-ASP *is a sound and complete planner for FOND$^+$.*

4. *Deciding if $T(P_c,\pi)$ has a model is in P; i.e.,* FOND-ASP *runs in non-deterministic exponential time.*

Proof (sketch). **Part 1.** A model M of a program T is stable *iff* it is a minimal model of the so-called reduct T_M of T, which means that M can be constructed inductively, bottom-up, assuming and verifying the stability condition; namely, that the atoms that are not in M have no well-founded support in T_M (Gelfond & Lifschitz, 1988; Lifschitz, 2019). In our program, this inductive construction yields a sequence of atom sets $M_0, M_1, \ldots$ over the atoms `connected/2`, `blocked_paths/2` (and hence `blocked_cycles/2`), `fair/1`, `terminate/1`, and `reachable/1` that is *uniquely determined* by the policy π, such that their union satisfies the stability condition. We note that the check on all cycles over a state s via a state s' in Definition 15 is achieved by `blocked_cycles/2`, which verifies that either all paths from s to s' or from s' to s include a terminating state (lines 8 and 9). This is complete as otherwise there would be a cycle not passing through any terminating state. Note that `blocked_cycle(S,T)` holds when there is no cycle on `S` passing through `T`, and that `blocked_paths(S,S)` holds iff state `S` is deemed terminating.

Parts 2 & 3. By Theorem 16, π solves P_c iff M is the unique model of $T(P_c,\pi)$ and M satisfies the integrity constraints. From this, part 3 is straightforward.

4. The way in which the atoms `fair/1` are defined to verify that all relevant cycles are “blocked” differs slightly from the one used by Rodriguez et al. (2021), which contains a subtle bug that is a result of using negative conditions in the body of conditional literals. These negative conditions are avoided in the current encoding.

```
 1 % policy; edges and connectedness compatible with policy
 2 { pi(S,A) : ACTION(A) } = 1 :- STATE(S), not GOAL(S).
 3 edge(S,T) :- pi(S,A), TRANSITION(S,A,T).
 4 connected(S,S) :- STATE(S).
 5 connected(S,T) :- connected(S,X), edge(X,T).
 6
 7 % all cycles on S via T are blocked (contain terminating states)
 8 blocked_cycles(S,T) :- blocked_paths(S,T).
 9 blocked_cycles(S,T) :- blocked_paths(T,S).
10
11 % there is no (S,T)-path or all such paths have a terminating state
12 blocked_paths(S,T) :- STATE(S), STATE(T), not connected(S,T).
13 blocked_paths(S,T) :- connected(S,T), terminates(S).
14 blocked_paths(S,T) :- connected(S,T), terminates(T).
15 blocked_paths(S,T) :- S != T, connected(S,T),
16                       blocked_paths(X,T): edge(S,X), S != X.
17
18 % for some A/B, pi(S) in A and each cycle on S that passes
19 % via T with pi(T) in B contains a terminating state
20 fair(S) :- pi(S,A), ASET(A,I), blocked_cycles(S,T): pi(T,B), BSET(B,I).
21
22 % terminating states
23 terminates(S) :- GOAL(S).
24 terminates(S) :- fair(S), edge(S,T), terminates(T).
25 terminates(S) :- not fair(S), edge(S,_), terminates(T): edge(S,T).
26
27 % reachable states must terminate
28 :- reachable(S), not terminates(S).
29 reachable(S) :- INITIAL(S).
30 reachable(S) :- reachable(X), not GOAL(X), edge(X,S).
```

Figure 2: The concise encoding of ASP-based FOND$^+$ planner FOND-ASP in CLINGO. The FOND$^+$ problem is specified through the predicates `STATE/1`, `ACTION/1`, `INITIAL/1`, `GOAL/1`, `TRANSITION/3`, `ASET/2` and `BSET/2`, where `ASET(I,A)` (resp. `BSET(I,A)`) iff action `A` belongs to A_i (resp. B_i) in the fairness assumption A_i/B_i. In CLINGO, the syntax "`P:␣<cond>`" (e.g, line 25) stands for the implicitly universally quantified conditional "if `<cond>` then `P`".

Part 4. A model for $T(P_c, \pi)$ can be constructed by calculating the terminating states for the policy graph for π. This can be done in time that is polynomial in the size of the policy graph which lower bounds the size of $T(P_c)$. FOND-ASP builds $T(P_c)$ in exponential time, then guesses a policy π in linear time in the size of $T(P_c)$, and finally checks if $T(P_c, \pi)$ has a model in time that is polynomial in the size of $T(P_c)$. □

10. Complexity

A direct consequence of Theorem 21 is that the plan-existence decision problem for FOND$^+$ is in NEXP (i.e., non-deterministic exponential time). Since FOND problems are easily reduced to FOND$^+$ problems (Theorem 13) and the plan-existence for FOND is EXP-Hard (Littman, Goldsmith, & Mundhenk, 1998; Rintanen, 2004), plan-existence for FOND$^+$ is EXP-Hard as well. We

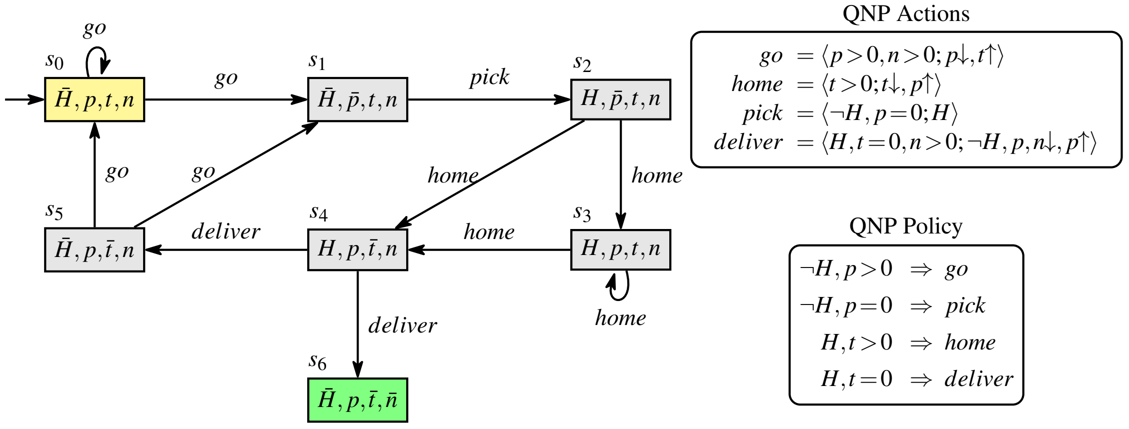
go
s_0
$\bar{H},p,t,n$
go
s_1
$\bar{H},\bar{p},t,n$
pick
s_2
$H,\bar{p},t,n$
go
go
home
home
s_5
$\bar{H},p,\bar{t},n$
deliver
s_4
$H,p,\bar{t},n$
home
s_3
H,p,t,n
home
deliver
s_6
$\bar{H},p,\bar{t},\bar{n}$
QNP Actions
$go = \langle p>0, n>0; p\downarrow, t\uparrow\rangle$
$home = \langle t>0; t\downarrow, p\uparrow\rangle$
$pick = \langle \neg H, p=0; H\rangle$
$deliver = \langle H, t=0, n>0; \neg H, p, n\downarrow, p\uparrow\rangle$
QNP Policy
$\neg H, p>0 \Rightarrow go$
$\neg H, p=0 \Rightarrow pick$
$H, t>0 \Rightarrow home$
$H, t=0 \Rightarrow deliver$

Figure 3: A policy π (bottom right) for the QNP abstraction for Delivery and the corresponding policy graph (left) on the direct translation $T_D(Q)$, where initial state shown in yellow and goal state in green. QNP actions are also shown (top right) as pairs $\langle C;E\rangle$ where C are the preconditions and E the effects. The fairness assumptions are $C_1 = go/\{home, deliver\}$, $C_2 = home/go$, and $C_3 = deliver$.

conjecture that the NEXP bound is loose and that plan-existence for FOND$^+$ is EXP-Complete. In contrast, LTL planning and synthesis is 2EXP-Complete (Pnueli & Rosner, 1989).

Theorem 22. *The plan-existence problem for FOND$^+$ problems is in NEXP and it is EXP-Hard.*

Proof. **Inclusion.** On input problem $P_c = \langle P, C\rangle$, the state space for P is constructed explicitly and a policy π for P_c is guessed in non-deterministic exponential time. Then, by Theorem 21, the logic program $T(P_c, \pi)$ has a model iff π solves P_c. Deciding whether $T(P_c, \pi)$ has a model can be done in time polynomial in the size of the program. **Hardness.** By Theorem 13, A FOND problem can be reduced in polynomial time to a FOND$^+$ problem. The claim follows by the EXP-Hardness of the plan-existence problem for FOND (Rintanen, 2004). □

11. Example: The Delivery Problem

The FOND$^+$ problem Delivery stands of the direct translation $T_D(Q)$ with the corresponding fairness assumptions for a QNP abstraction Q of all classical planning problems where packages in a grid must be picked up, one by one, and delivered to a target cell (Bonet & Geffner, 2020). The QNP Q has one Boolean feature H that captures whether the agent holds a package, and 3 numerical features p, t and n, where p is the distance to a nearest package, t is the distance to the target cell, and n counts the number of packages to be delivered. On the other hand, Q has 4 actions. The action *go* that moves the agent towards a nearest package, the action *home* that moves the agent towards the target cell, the action pickup that picks up the package that is in the same cell as the agent, and the action *deliver* that deliver the package being held by the agent to the target cell. The QNP abstraction models these actions as follows: *go* has precondition $\{p>0, n>0\}$ and effect $\{p\downarrow, t\uparrow\}$, *home* has precondition $\{t>0\}$ and effect $\{t\downarrow, p\uparrow\}$, *pickup* has precondition $\{\neg H, p=0\}$ and effect $\{H\}$, and *deliver* has precondition $\{H, t=0, n>0\}$ and effect $\{\neg H, n\downarrow, p\uparrow\}$.

The fairness assumptions that are needed for obtaining solutions for the QNP Q when solving the FOND problem $T_D(Q)$ are $C_1 = go/\{home, deliver\}$, $C_2 = home/go$, and $C_3 = delivery$, expressed with the notation used in the previous example; e.g., C_3 stands for $\{delivery\}/\emptyset$. These assumptions are obtained by the decrement/increment effects in the actions as described in Theorem 14.

The right side of Figure 3 shows the policy that makes the agent to travel between package locations and the target cell, while picking up and delivering packages, until no undelivered package remains, and the left side shows the corresponding policy graph for the direct translation $T_D(Q)$.

Rather than explaining how the solver finds the policy, we provide the intuition of how the program in Figure 2 verifies the policy, labeling the states in the graph as terminating, in order (as the resulting stable models of the program must be well-founded):

1. Initially, the goal state s_6 is labeled as terminating by the rule in line 23.

2. The state s_4 is labeled as terminating by the rule in line 24 as the condition `fair(S)` holds trivially by the assumption $C_3 = delivery$.

3. Both s_2 and s_3 are also labeled as fair because $\pi(s_2) = \pi(s_3) = home$ and all the cycles C that contain a state where go is applied, also contain the state s_4 that is terminating. Hence, s_2 is labeled as terminating because it is connected to s_4, and immediately afterwards, s_3 is labeled as terminating because it is connected to s_2.

4. It follows then that s_1 is also terminating as its single successor s_2 is terminating. Indeed, it is irrelevant whether states s with deterministic actions $\pi(s)$ are treated as fair or not when their unique successors have been already labeled as terminating.

5. Finally, s_0 and s_5 are labeled as fair as both execute the action go that appears in the set A of the assumption $C_1 = go/\{home, deliver\}$, and the two actions $home$ and $deliver$ are only executed in states already labeled as terminating. Hence, since s_1 is terminating and is successor of both, both s_0 and s_5 become terminating.

12. Experiments

We tested FOND-ASP on three classes of problems:[5] FOND problems, QNPs, and more expressive FOND$^+$ problems that do not fit in either class and that can only be addressed using LTL engines. On each class, we compare FOND-ASP with the FOND solvers FOND-SAT (Geffner & Geffner, 2018) and PRP (Muise et al., 2012), the QNP solver QNP2FOND (Bonet & Geffner, 2020) using FOND-SAT and PRP as the underlying FOND solver, and the LTL-synthesis tool STRIX (Luttenberger, Meyer, & Sickert, 2020). The pure (strong and strong-cyclic) FOND problems are those in the FOND-SAT distribution, the QNPs are those by Bonet and Geffner (2020), and two new families of instances that grow in size with a parameter. For more expressive FOND$^+$ planning problems, four new families of problems are introduced that extend the new QNPs with fair and adversarial actions, with only some being solvable. The domain and goals of these problems are encoded in LTL in the usual way, while the fairness assumptions A/B are encoded as described in the introduction. In all the experiments, time and memory bounds of 1/2 hour and 8GB are enforced.

5. Planner and problems available at `https://github.com/idrave/fond-asp`

The results are detailed below. In summary, we observe the following. For pure FOND benchmarks, FOND-ASP does not compete with specialized planners like PRP or FOND-SAT as these problems span (reachable) state spaces that are just too large. For QNPs, on the other hand, FOND-ASP does better than FOND-SAT but worse than PRP on the FOND translations. For expressive FOND$^+$ problems, where these planners cannot be used at all, FOND-ASP performs much better than STRIX on both solvable and unsolvable problems.

12.1 FOND Benchmarks

FOND-ASP managed to solve a tiny fraction of the benchmarks used for strong and strong-cyclic planning in the FOND-SAT distribution. The number of reachable states in these problems is large (tens of thousands of states, or more) and the size of the grounded ASP program is quadratic in that number. In general, this seems to limit the scope of FOND-ASP to problems with no more than one thousand states approximately, as suggested by the results in Table 1. We have observed however that sometimes FOND-ASP manages to solve strong planning problems with more than 100,000 states. This may have to do with CLINGO's grounder or with the state space topology; we do not know the exact reason yet.

12.2 QNP Problems

The two families of QNPs involve the numerical variables $\{x_i\}_{i=1}^n$ that have all positive values in the initial state. The goal is to achieve $x_n = 0$. Problems in the QNP1 family are solved by means of n sequential simple loops, while problems in the QNP2 family are solved using n nested loops. The actions for problems in QNP1 are $b = \langle \neg p; p \rangle$, $a_1 = \langle p; \neg p, x_1{\downarrow} \rangle$, and $a_i = \langle p, x_{i-1}=0; \neg p, x_i{\downarrow} \rangle$ for $1 < i \leq n$, while those for QNP2 are $b = \langle \neg p; p \rangle$, $a_1 = \langle p; \neg p, x_1{\downarrow} \rangle$, and $a_i = \langle p, x_{i-1}=0; \neg p, x_{i-1}{\uparrow}, x_i{\downarrow} \rangle$, $1 < i \leq n$.

Table 1 shows the results for values of n in $\{2, 3, \ldots, 10\}$ and different planners, along with the number of reachable states in each problem. As it can be seen, QNP2FOND/PRP is the planner that scales best, followed by FOND-ASP, QNP2FOND/FOND-SAT, and STRIX at the end. As mentioned, the performance of FOND-ASP is harmed by a large number of reachable states. While the number of states for the FOND translation produced by QNP2FOND is much larger, as the translation involves extra propositions, this number does not necessarily affect the performance of FOND planners like FOND-SAT and PRP that can compute compact policies. It is also interesting to see how quickly the performance of the LTL engine STRIX degrades; it cannot even solve qnp1-06 which has 14 states. The table also shows results for QNP problems that capture abstractions for 4 generalized planning problems, all of which involve small state spaces (Bonet & Geffner, 2020).

12.3 More Expressive FOND$^+$ Problems

The third class of instances consists of four families of problems obtained from the two QNP families above. The new problems are not "pure" QNPs, as they also involve actions with non-deterministic effects on Boolean variables that can be adversarial or fair. Thus, these problems cannot be translated into FOND problems for the use of planners such as PRP or FOND-SAT. For each family QNP1 and QNP2, two new families $f01$ and $f11$ of problems are obtained by replacing the action $b = \langle \neg p; p \rangle$ by the non-deterministic action $b' = \langle \neg p; oneof\{p, \neg p\} \rangle$, leaving the actions a_i untouched. Since the action b' does not appear in any fairness assumption, it is adversarial and

		QNP2FOND			
problem	#states	FOND-SAT	PRP	STRIX	FOND-ASP
qnp1-02	6	0.08	0.09	3.35	0.00
qnp1-03	8	0.13	0.11	2.63	0.00
qnp1-04	10	0.28	0.14	5.21	0.00
qnp1-05	12	0.60	0.14	98.34	0.01
qnp1-06	14	1.27	0.15	—	0.01
qnp1-07	16	2.54	0.15	—	0.01
qnp1-08	18	5.54	0.17	—	0.01
qnp1-09	20	12.96	0.21	—	0.02
qnp1-10	22	26.70	0.19	—	0.02
qnp2-02	8	0.20	0.18	2.33	0.00
qnp2-03	16	1.77	0.30	2.31	0.01
qnp2-04	32	10.00	0.58	14.25	0.04
qnp2-05	64	50.24	1.15	885.37	0.20
qnp2-06	128	302.80	2.53	—	1.26
qnp2-07	256	1,969.35	4.02	—	7.14
qnp2-08	512	—	6.96	—	54.37
qnp2-09	1,024	—	13.22	—	***
qnp2-10	2,048	—	21.94	—	***
Clear	4	0.23	0.13	1.53	0.00
On	16	3.69	0.20	3.01	0.01
Delivery	12	4.07	0.27	1.50	0.01
Gripper	12	15.43	1.61	2.47	0.02

Table 1: Results for three families of QNPs for QNP2FOND paired with the FOND solvers FOND-SAT and PRP, STRIX (QNP translated to LTL), and FOND-ASP. Entries '—' and '***' denote out of time and memory, respectively. Time is in seconds.

thus no problem in the class $f01$ has a solution as the "adversary" may always choose to leave p false. The family $f11$ is obtained on top of $f01$ by adding two additional Booleans q and r, and two actions $c = \langle \neg q; r, oneof\,\{q, \neg q\}\rangle$ and $d = \langle r; q, \neg r\rangle$ such that: 1) the actions a_i are modified by adding q as precondition and $\neg q$ as effect, and 2) the fairness assumption A/B with $A = \{b'\}$ and B empty is added. The problems in $f11$ thus involve the QNP-like actions a_i, the fair action b', and the adversarial action c, and they all have a solution.

Table 2 shows the result for FOND-ASP and STRIX as these are the only solvers able to handle the combination of fairness assumptions. As it can be seen, FOND-ASP scales better than STRIX on all of these problems, the solvable ones (families $f11$) and the unsolvable ones (families $f01$).

We finally tested FOND-ASP over the 7 problems considered in a recent approach to program synthesis over unbounded data structures (Bonet, De Giacomo, Geffner, Patrizi, & Rubin, 2020). Although the original specifications are in LTL, these can be all expressed in FOND$^+$ using different types of conditional fairness assumptions. The problems are solved easily by both FOND-ASP and STRIX as their reachable state spaces are very small.

	$f01$ (unsolvable)			$f11$ (solvable)		
problem	#states	STRIX	FOND-ASP	#states	STRIX	FOND-ASP
qnp1-$f$xx-02	6	3.44	0.00	24	6.07	0.03
qnp1-$f$xx-03	8	2.42	0.01	32	6.20	0.04
qnp1-$f$xx-04	10	4.13	0.01	40	98.58	0.07
qnp1-$f$xx-05	12	93.85	0.01	48	—	0.11
qnp1-$f$xx-06	14	—	0.01	56	—	0.14
qnp1-$f$xx-07	16	—	0.01	64	—	0.19
qnp1-$f$xx-08	18	—	0.01	72	—	0.25
qnp1-$f$xx-09	20	—	0.02	80	—	0.39
qnp1-$f$xx-10	22	—	0.02	88	—	0.34
qnp2-$f$xx-02	8	3.22	0.00	32	5.85	0.04
qnp2-$f$xx-03	16	2.25	0.01	64	8.16	0.21
qnp2-$f$xx-04	32	11.38	0.04	128	236.89	1.55
qnp2-$f$xx-05	64	873.09	0.21	256	—	15.45
qnp2-$f$xx-06	128	—	1.25	512	—	46.67
qnp2-$f$xx-07	256	—	12.13	1,024	—	***
qnp2-$f$xx-08	512	—	39.56	2,048	—	***
qnp2-$f$xx-09	1,024	—	***	4,096	—	***
qnp2-$f$xx-10	2,048	—	***	8,192	—	***

Table 2: Results for four families of solvable/unsolvable FOND$^+$ problems obtained from the QNPs in Table 1 by adding non-deterministic actions and conditional fairness assumptions. These problems are handled only by STRIX and FOND-ASP. Entries '—' and '***' denote out of time and memory, respectively. Time is in seconds.

13. Two Incomplete FOND$^+$ Planners

The main limitation of the FOND-ASP planner is the size of the grounded encoding (i.e., the size of the program after grounding), which grows as $O(|S|^2 \times B)$ where $|S|$ is the number of states, and B is the average number of possible state successors; i.e., the number of states s' that may follow a state s after an action. This is indeed the number of rules that follow the grounding of the rule in line 15 in Figure 2. This quadratic growth on the size of the state space explains why the planner solves all instances with no more than 512 states but none of the instances with 1,024 states or more. In order to deal with larger state spaces we introduce two small variations of the base FOND$^+$ planner that replace the quadratic factor $|S|^2$ by a linear factor $|S|$. The new planners are sound but no longer complete, and we illustrate this difference with an example. On the other hand, the two incomplete planners solve all the FOND$^+$ problems considered so far, thus scaling up to larger instances.

The quadratic growth in the encoding of FOND-ASP results from lines 4–20 in the code of FOND-ASP (Figure 2) that check if a state s is fair when $\pi(s) \in A$ for an assumption A/B. This is done by checking if the policy cycles involving the state s and states s' where $\pi(s') \in B$ also include a terminating state (i.e., atom `blocked_cycles(s,s')` is true). The two incomplete planners FOND-

```
 1 % fairc/1: occurrence of actions in A_i are fairc for assumption A_i/B_i
 2 % if B_i is empty, or actions in B_i only applied at terminating states
 3 fairc(I) :- ASET(I,_), { BSET(I,B) } = 0.
 4 fairc(I) :- BSET(I,_), terminates(S): pi(S,B), BSET(I,B).
 5
 6 % fair(S) if pi(S) in A for some A/B and all occurences of actions in A
 7 % are fairc
 8 fair(S) :- pi(S,A), ASET(I,A), fairc(I).
 9
10 % termination on states with an action in set A for some assumption A/B
11 terminates(S) :- pi(S,A), ASET(_,A), edge(S,U), S != U,
12                  terminates(T): edge(S,T), T != S.
```

Figure 4: Planner FOND-ASP2. Lines shown replace lines 4–20 in FOND-ASP (Figure 2). The size of the resulting grounded encoding is $O(|S| \times B)$, dominated by the rule in line 11, unlike the size of the encoding generated by FOND-ASP, which is $O(|S|^2 \times B)$, where B is the average number of states s' that may follow a state s.

ASP2 and FOND-ASP3 provide a sound but incomplete approximation of this condition that avoids the consideration of either cycles or paths.

The FOND-ASP2 planner replaces lines 4–20 in FOND-ASP by the lines shown in Figure 4. In the new code, a state s where $\pi(s) \in A$ for some fairness assumption A/B is deemed fair, if the states s' where $\pi(s') \in B$ are all terminating, *even if there are no cycles involving s and s'*. In addition, however, FOND-ASP2 makes a second change in the code: a state s is deemed as terminating if $\pi(s) \in A$ for some assumption A/B and all its successors s' *different than* s are terminating. This change makes a difference only when s is a possible successor of itself (i.e., self-loops). The new rule is sound because if $\pi(s) \in A$ for some assumption A/B, any cycle involving a state s' where $\pi(s') \in B$ must involve one of the children s'' of s different than s itself, as $\pi(s) \in A$ and A and B are assumed to be disjoint. Thus, if all such children are terminating (traversed a finite number of times only), the state s is fair, according to the definitions, and hence terminating as well, in spite of the self-loop. Indeed, the new rule for `terminates/1` (line 11 in Figure 4) is implied by the original FOND-ASP code (Figure 2), and the new rule for `fair/1` in FOND-ASP2 is just a restriction of the rules for `fair/1` in the original FOND-ASP code.

Theorem 23. *The FOND$^+$ planner* FOND-ASP2 *is sound.*

Proof. First observe that soundness easily follows if one can show that the set of states labeled as fair (resp. terminating) by FOND-ASP2 is a subset of the states labeled as fair (resp. terminating) by FOND-ASP. Indeed, FOND-ASP2 is unsound iff it labels a state as terminating when it is not (according to FOND-ASP). Yet, this is impossible if the above inclusions hold since the requirements for non-fair states to be labeled as terminating are more demanding than those for fair states.

The inclusion of the fair/terminating states is shown inductively. Initially, only states s with $\pi(s) \in A$ for some assumption $A/\emptyset$ are labeled as fair, while only goal states are labeled as terminating. In both cases, it is easy to see that such states attain the same labels by FOND-ASP.

Consider now fixed subsets F and T of fair and terminating states, respectively, calculated by FOND-ASP2. Let s be a state that becomes fair using F and T. It must be via the rule in line 4. Yet then, by induction, it is easy to see that s would have been labeled as fair too by FOND-

ASP. Likewise, let s be a state that becomes terminating using F and T. If s becomes labeled as terminating by the rules in FOND-ASP, it is also labeled as terminating by FOND-ASP, given the inductive hypothesis and the very first observation. Finally, if s becomes labeled as terminating by the rule in line 11, it is also labeled as terminating by FOND-ASP. Indeed, there are two cases: either s is fair or not. In the first case, FOND-ASP labels s as terminating if it has a successor s' that is terminating, something that is granted since s has a successor s' that is in T. In the second case, all successors of s must be terminating. This is guaranteed for all successors of s that are different from s. Hence, we are only left to consider the case when s is a successor of itself and s is not fair as per FOND-ASP. This case however is impossible since then there is an assumption A/B such that $\pi(s) \in A$, and all successors of s different from s are terminating. Therefore, by FOND-ASP, all cycles that go through a successor $s' \neq s$ are "blocked," that is, include a terminating state. Thus, s is not fair only because $\pi(s) \in B$, which contradicts the assumption that A and B are disjoint. □

The second incomplete planner FOND-ASP3 replaces the same lines 4–20 in Figure 2 by the lines shown in Figure 5. The resulting planner is more subtle than the previous ones, as it considers the propositional structure of the states; namely, the Boolean variables of the problem and their truth values in the different states. Technically, in addition to the rules for determining when a state s is terminating (i.e., s cannot be traversed an infinite number of times), FOND-ASP3 includes rules for determining when a variable (atom) K is *terminating*, in that K cannot change its truth value an infinite number of times under the policy. More precisely, a variable K is deemed *terminating* (line 11) if the policy never "flips" the truth value of K or it does so only in states that are terminating. A variable K flips its truth value in a state s (lines 7 and 8) if there is a state transition (s,s') compatible with the action $\pi(s)$ where K changes its truth value (from true to false, or vice-versa). A state s is then deemed fair by FOND-ASP3 when $\pi(s) \in A$ for an assumption A/B, and the states s' where an action in B is performed (i.e., $\pi(s') \in B$) are either terminating (as in FOND-ASP2) or differ from s in the value of a terminating variable K (lines 28 and 29, together with lines 20–23, in Figure 5). This last case is new and rules out the possibility of infinite cycles comprising states s and s' that differ in the value of a terminating variable.

Theorem 24. *The FOND$^+$ planner* FOND-ASP3 *is sound.*

Proof. In FOND-ASP3, a variable K flips in a state s if there is a state transition (s,s') compatible with the action $\pi(s)$ where K changes its value, and it is terminating if there are no actions that flip K, or such actions only occur in terminating states (line 11 in Figure 5). A state s is regarded as fair when $\pi(s) \in A$ for an assumption A/B, and the states s' for which $\pi(s') \in B$, are terminating (lines 20–23), or such states s' differ from s in the value of a terminating variable K (lines 28–29). In the induction, we need to show that the intuitive meaning of terminating states s and variables K is captured; namely, that terminating states are traversed a finite number of times only, and that terminating variables flip a finite number of times too. Initially, this is clearly true for the goal states that are terminating. Then, given a set of terminating states and variables, we need to show that the rules in the program preserve this meaning. It can be shown that `fairc(i)` is true if there is an assumption A_i/B_i such that all the states s' where $\pi(s') \in B_i$ are terminating. Then, if `fairc(i)` holds, `fair(s)` is true for $\pi(s) \in A_i$. This is sound as the general definition says that s is fair if $\pi(s) \in A_i$ and every cycle involving s and states s' where $\pi(s') \in B_i$ must include a terminating state, which in this case, are the s' states themselves. It is thus left to show that lines 28 and 29 are sound as well. The first rule says that s is fair if $\pi(s) \in A_i$, there is a terminating variable K that is

```
 1 % definition of eff/4
 2 eff(S,K,A,0) :- TRANSITION(S,A,T), not VAL(T,K).
 3 eff(S,K,A,1) :- TRANSITION(S,A,T),     VAL(T,K).
 4
 5 % flips/3: policy action in S may flip  value of variable K to true/false
 6 atom(K)      :- VAL(_,K).
 7 flips(S,K,1) :- pi(S,A), not VAL(S,K), eff(S,K,A,1).
 8 flips(S,K,0) :- pi(S,A), VAL(S,K), eff(S,K,A,0).
 9
10 % terminates_var/1: var K can only flip a finite number of times
11 terminates_var(K) :- X=0..1, atom(K), terminates(S): flips(S,K,X).
12
13 % fairc/1: B_i is empty or all states where an action in B_i is applied
14 % can only be traversed finitely often (actions in A_i are fair)
15 fairc(I) :- ASET(I,_), { BSET(I,B) } = 0.
16 fairc(I) :- BSET(I,_), terminates(S): pi(S,B), BSET(I,B).
17
18 % fairc/3: all states with value V (true/false) of variable K where an
19 % action in B_j is applied can only be traversed finitely often
20 fairc(I,K,1) :- atom(K), BSET(I,_),
21                 terminates(S): pi(S,B), BSET(I,B), VAL(S,K).
22 fairc(I,K,0) :- atom(K), BSET(I,_),
23                 terminates(S): pi(S,B), BSET(I,B), not VAL(S,K).
24
25 % fair(S) if for some A/B, pi(S) in A and each cycle over S that passes
26 % over X such that pi(X) in B contains a terminate state
27 fair(S) :- pi(S,A), ASET(I,A), fairc(I).
28 fair(S) :- pi(S,A), ASET(I,A), terminates_var(K), VAL(S,K), fairc(I,K,1).
29 fair(S) :- pi(S,A), ASET(I,A), terminates_var(K), not VAL(S,K),
30            fairc(I,K,0).
```

Figure 5: Planner FOND-ASP3. Lines shown replace lines 4–20 in FOND-ASP (Figure 2). In this code, `VAL(S,K)` is true if atom K is true in state S, while `eff(S,K,A,0)` (resp., `eff(S,K,A,1)`) is true if some state transitions at s associated with action A makes K false (resp., true). `VAL/2` atoms are part of the input, while `eff/4` atoms are derived from the input in lines 2–3. The code allows to conclude that a state s is fair by reasoning over the variables, without having to consider either cycles or paths that include s. The size of the grounded encoding generated by FOND-ASP3 is $O(|S| \times |At|)$, dominated, among others, by the rules in lines 20 and 28 (recall that At is the set of domain atoms in the planning problem).

true in s, and through the rule for `fairc/3`, that all the states s' such that $\pi(s') \in B_i$ and where K is true, are terminating. This does not rule out non-terminating states s'' where $\pi(s'') \in B_i$ and K is false; yet such states s'' differ from s in the value of the terminating variable K, therefore any cycle involving s and any such s'' must include a state that is terminating. As a result, the rule captured by line 28 is sound. The soundness of the rule captured by line 29 is established in the same way with the sole difference that K is false in the states s and s''. □

Experimental results of the two incomplete planners FOND-ASP2 and FOND-ASP3 are shown in comparison with complete planner FOND-ASP in Tables 3 and 4. In the these tables we can see that the two incomplete planners manage to solve all the problems considered, while scaling much

		FOND-ASP		FOND-ASP2		FOND-ASP3	
problem	#states	time	mem	time	mem	time	mem
qnp1-02	6	0.00	39.60	0.02	39.57	0.02	39.20
qnp1-03	8	0.00	39.72	0.04	39.68	0.02	39.34
qnp1-04	10	0.00	39.83	0.02	39.80	0.04	39.47
qnp1-05	12	0.01	39.88	0.03	39.92	0.03	39.67
qnp1-06	14	0.01	40.10	0.04	40.08	0.04	39.84
qnp1-07	16	0.01	40.17	0.03	40.14	0.04	39.84
qnp1-08	18	0.01	40.18	0.03	40.16	0.06	39.89
qnp1-09	20	0.02	40.24	0.05	40.22	0.05	40.14
qnp1-10	22	0.02	40.48	0.05	40.45	0.07	40.42
qnp2-02	8	0.00	39.45	0.02	39.70	0.02	39.14
qnp2-03	16	0.01	39.65	0.04	39.90	0.03	39.40
qnp2-04	32	0.04	40.04	0.06	40.30	0.08	39.86
qnp2-05	64	0.20	40.38	0.18	40.65	0.21	40.80
qnp2-06	128	1.26	68.09	0.53	42.68	0.49	43.44
qnp2-07	256	7.14	248.46	1.18	47.48	1.37	48.90
qnp2-08	512	54.37	1047.1	4.10	51.07	4.67	57.42
qnp2-09	1,024	***	***	15.12	65.07	302.74	97.83
qnp2-10	2,048	***	***	58.17	97.40	134.83	219.96
Clear	4	0.00	39.56	0.02	39.50	0.01	39.58
On	16	0.01	39.76	0.03	39.69	0.04	39.88
Delivery	12	0.01	39.94	0.03	39.86	0.04	40.13
Gripper	12	0.02	40.15	0.04	40.08	0.03	40.30

Table 3: Comparison of FOND-ASP against the two incomplete variants on the same problems from Table 1. Entries '—' and '***' denote out of time and memory, respectively. Time is in seconds and memory in MB.

better in terms of memory. Indeed, FOND-ASP solves all problems with at most 512 states and then yields memory outs, while FOND-ASP2 and FOND-ASP3 manage to solve problems with up to $2,048$ states, producing then time outs.

14. Example: Incomplete Planners

In order to give an intuition of the power and limitations of the two new planners, we first describe how FOND-ASP2 and FOND-ASP3 manage to label all reachable states in the Delivery problem from Section 11, and then given an example on which the new planners do not find a solution, while FOND-ASP does.

		FOND-ASP		FOND-ASP2		FOND-ASP3	
problem	#states	time	mem	time	mem	time	mem
qnp1-f11-02	24	0.03	39.62	0.08	39.96	0.07	39.58
qnp1-f11-03	32	0.04	39.75	0.07	40.24	0.10	39.83
qnp1-f11-04	40	0.07	40.21	0.10	40.77	0.12	40.57
qnp1-f11-05	48	0.11	40.34	0.15	40.95	0.14	41.50
qnp1-f11-06	56	0.14	41.16	0.16	41.73	0.18	42.61
qnp1-f11-07	64	0.19	42.12	0.16	42.69	0.14	43.92
qnp1-f11-08	72	0.25	43.22	0.18	43.79	0.18	45.35
qnp1-f11-09	80	0.39	44.44	0.23	45.03	0.22	46.98
qnp1-f11-10	88	0.34	45.84	0.26	46.41	0.29	47.54
qnp2-f11-02	32	0.04	39.64	0.11	40.01	0.07	39.72
qnp2-f11-03	64	0.21	40.13	0.20	40.66	0.19	40.51
qnp2-f11-04	128	1.55	63.32	0.68	42.86	0.52	43.38
qnp2-f11-05	256	15.45	232.96	1.88	47.60	1.67	48.54
qnp2-f11-06	512	46.67	950.44	10.35	50.09	6.49	57.42
qnp2-f11-07	1,024	***	***	448.04	61.89	22.77	78.28
qnp2-f11-08	2,048	***	***	—	87.21	—	184.34
qnp2-f11-09	4,096	***	***	—	139.55	—	424.16
qnp2-f11-10	8,192	***	***	—	249.13	—	978.36

Table 4: Comparison of FOND-ASP against the two incomplete variants FOND-ASP2 and FOND-ASP3 over instances of the solvable family qnp2-f11 from Table 2. Entries '—' and '***' denote out of time and memory, respectively. Time is in seconds and memory in MB.

14.1 Delivery

In Section 11, we discussed how FOND-ASP labels all states reachable with the policy as terminating by reasoning about cycles in the context of the fairness assumptions $C_1 = go/\{home, deliver\}$, $C_2 = home/go$, and $C_3 = deliver$ FOND-ASP2 and FOND-ASP3 achieve the same result but without reasoning about cycles or paths, something that makes them more scalable but incomplete.

FOND-ASP2 labels states s_6 and s_4 in the same way as FOND-ASP: state s_6 because it is a goal state, and s_4 because, according to FOND-ASP2, it is fair (due to assumption C_3 and $\pi(s_4) = delivery$) and connected to s_6. On the other hand, the reasoning for states s_2 and s_3 is now different. The action $\pi(s_2) = \pi(s_3) = home$ yields the set of successors $\{s_3, s_4\}$ for both s_2 and s_3. Hence, the new rule for termination (line 11 in Figure 4) applies at s_3, since s_4 is already terminating and $\pi(s_3) \in A_3$ (note the rule ignores the self-loop at s_3). Afterwards, as the successors of s_2 (i.e., s_3 and s_4) have been labeled as terminating, s_2 is then labeled terminating as well. The only successor of state s_1, via deterministic action $\pi(s_1) = pick$, is the terminating state s_2, so s_1 will also be deemed terminating. The termination of s_0 follows, once again, from the new rule for `terminates/1` in FOND-ASP2, since $go \in A_1$ and s_1 is terminating (self-loop ignored). Finally, s_5 becomes terminating since all its successor states (i.e., s_0 and s_1) are terminating.

The reasoning of FOND-ASP3 is different in that it deals with state variables in addition to states. As before, the goal state s_6 is terminating while s_4 with $\pi(s_4) = deliver$ is fair and terminating because of the assumption C_3. With that, we conclude that variable H, which tells whether the agent holds a package, will be deemed as a terminating variable, as it changes from true to false only in terminating state s_4 (line 11 in Figure 5). This means that H can "flip" its value only a finite number of times. With H is marked as terminating, states s_2 and s_3 where $\pi(s_2) = \pi(s_3) = \mathit{home}$ are labeled as *fair* (line 28 in Figure 5) because of the assumption $C_2 = \mathit{home}/\mathit{go}$ and the fact that the *go* action is only executed at states where $\neg H$ holds (i.e., `fairc(I,K,1)` holds). Since s_2 and s_3 are fair and connected to s_4 that is terminating, both are labeled as terminating (line 24 in Figure 2). This triggers the labeling of s_1 as terminating, as the only successor s_2 is terminating, regardless whether s_1 is fair or not. Finally, states s_0 and s_5, in which *go* is executed, are labeled as fair via rule in line 29 of Figure 5, because of the assumption $C_1 = \mathit{go}/\{\mathit{home}, \mathit{deliver}\}$, the fact that variable H is terminating, and the fact that actions *home* and *deliver* are only executed in states where the value of H is different from that in s_0 and s_5. Right afterwards, both states are also then deemed terminating since they have a successor, namely, state s_1, that is terminating.

Putting it all together, all reachable states are deemed terminating and integrity constraint in line 28 of Figure 2 is met in both FOND-ASP2 and FOND-ASP3.

14.2 Incompleteness

We illustrate the incompleteness of the planners with a counterexample. For simplicity, there is just one applicable action at each state, and hence, a single policy to consider. The problem, shown in Figure 6, consists of the variables x and y, the actions $a = \langle \neg x, \neg y; \mathit{oneof}\{\neg x, x\}\rangle$, $b = \langle x, \neg y; \mathit{oneof}\{\neg x, x\}, y\rangle$ and $c = \langle x, y; \neg y\rangle$, and the fairness assumptions $C_1 = a/b$ and $C_2 = b/a$. The initial and goal states are $s_0 = \{\neg x, \neg y\}$ and $s_2 = \{\neg x, y\}$.

It is easy to see that FOND-ASP labels all states as terminating: s_2 since it is a goal state; s_1 since $\pi(s_1) = b$ but in no state reachable from s_1, a is applied; s_3 since $\pi(s_3) = c$ is deterministic and its unique successor is s_1; and s_0 since $\pi(s_0) = a$ and in no cycle involving s_0 the action b is applied. On the other hand, FOND-ASP2 is unable to label s_1 nor s_3 as terminating as neither of them is fair (s_1 because `fairc` does not hold for $C_2 = b/a$, and s_3 because $\pi(s_3) = c$ does not appear in any assumption), and each such state has the other as successor. FOND-ASP3 exhibits a similar limitation; it cannot establish that either x or y are terminating variables because there are state transitions from s_0, s_1, and s_3 that flip the value of the variables, and none of these states can be shown to be terminating.

The fact that FOND-ASP2 and FOND-ASP3 manage to solve all the problems used to evaluate FOND-ASP illustrates that these planners are powerful, despite their incompleteness. It is easy to come up with examples that one planner can solve and the other cannot but it also simple to combine the two planners into one, still incomplete, and unable to solve the last example. The practical suggestion is to try the complete planner first, and if it fails due to memory, to try the incomplete planners, that appears to do fairly well in the variety of FOND$^+$ problems that we have tried, while consuming an amount of memory that grows linearly with the size of the state space rather than quadratic. The problem of designing a complete FOND$^+$ planner that does not expand the full state space first, as it is done by more scalable FOND planners, is left as an open challenge that is beyond the scope of this work. Such a complete planner would establish the EXP-Completeness of FOND$^+$ planning.

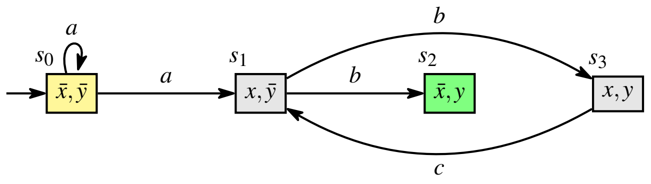

Figure 6: A simple problem solved by FOND-ASP but not by FOND-ASP2 or FOND-ASP3. FOND-ASP2 is unable to label $\{s_1, s_3\}$ as terminating since none of them is fair and each one if a successor of the other, while FOND-ASP3 is unable to establish that either of the variables is terminating.

15. Related Work

This work is related to three threads: SAT-based FOND planning, QNPs, and LTL synthesis. The SAT-based FOND planner by Chatterjee et al. (2016) expands the state space in full, like FOND-ASP, but a more recent version computes compact policies and provides support for Dual FOND planning (Geffner & Geffner, 2018). We have used answer set programs as opposed to CNF encodings exploiting their high-level modeling language, the natural support for inductive definitions, and the competitive performance of CLINGO (Gebser et al., 2019). FOND-ASP is also a novel QNP planner which can handle non-deterministic effects on Boolean variables. The formulation actually brings QNP planning into the realm of standard FOND planning by dealing with the underlying fairness assumptions explicitly.

The use of fairness assumptions connects also to works on LTL planning and synthesis (Camacho et al., 2019; Aminof et al., 2019), and to works addressing temporally extended goals (De Giacomo & Vardi, 1999; Patrizi, Lipovetzky, & Geffner, 2013; Camacho, Triantafillou, Muise, Baier, & McIlraith, 2017; Camacho et al., 2019; Aminof et al., 2020). Our work can be seen as a special case of planning under LTL assumptions (Aminof et al., 2019) that targets an LTL fragment that is relevant for FOND planning and is computationally simpler. While it is possible to express FOND$^+$ tasks as LTL syntheses problems, and we have shown how to do that, it remains to be seen whether the task can be expressed in a *restricted* LTL fragment that admits more efficient techniques. While the *strong* fairness assumption on action effects that is required cannot be *directly* encoded in GR(1) (Bloem, Jobstmann, Piterman, Pnueli, & Sa'ar, 2012), strong-cyclic FOND planning has been encoded in Büchi Games (D'Ippolito, Rodríguez, & Sardiña, 2018), a special case of GR(1). It remains to be investigated whether that encoding can be extended to deal with *conditional* fairness.

The conditional fairness assumptions handled in FOND$^+$ planning could be used to extend the power of LTL planning over finite traces (De Giacomo & Vardi, 2015). Fairness assumptions are constraints on infinite trajectories and thus cannot be expressed in LTL$_f$, but at the same time, the fairness assumptions considered in this work do not appear to require the full complexity of LTL planning over infinite traces either. In principle, it would be of interest to consider LTL planning over finite traces under the explicit fairness assumptions handled in FOND$^+$ planning.

16. Summary

We have formulated an extension of FOND planning that makes use of explicit fairness assumptions of the form A/B where A and B are disjoints sets of actions. While in Dual FOND planning actions are labeled as fair or unfair, in FOND$^+$ planning these labels are a function of the trajectories and the fairness assumptions: an action $a \in A$ is deemed fair in a recurrent state if a suitable condition on B holds. In this way, FOND$^+$ generalizes strong, strong-cyclic, Dual FOND planning, and also QNP planning, which is actually the only planning setting, excluding LTL planning, that makes use of the conditions B. We have implemented an effective FOND$^+$ planner by reducing the problem to answer set programs using CLINGO, and evaluated its performance in relation to FOND and QNP planners, which handle less expressive problems, and LTL synthesis tools, which handle more expressive ones. At the same time, we have introduced two incomplete variants of the planner that are sufficiently powerful to solve all the problems considered in this work by scaling up much better in terms of memory. Finally, we have shown that FOND$^+$ is in NEXP but leave as an open problem whether it is in EXP like FOND and QNP planning.

Acknowledgments

We thank the CLINGO team for the tool and documentation. The work is partially supported by an ERC Advanced Grant (No 885107), the project TAILOR, funded by an EU Horizon 2020 Grant (No 952215), and by the Knut and Alice Wallenberg (KAW) Foundation under the WASP program.

References

Aminof, B., De Giacomo, G., Murano, A., & Rubin, S. (2019). Planning under LTL environment specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 31–39.

Aminof, B., De Giacomo, G., & Rubin, S. (2020). Stochastic fairness and language-theoretic fairness in planning in nondeterministic domains. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 20–28.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., & Sa'ar, Y. (2012). Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, *78*(3), 911–938.

Bonet, B., De Giacomo, G., Geffner, H., & Rubin, S. (2017). Generalized planning: Non-deterministic abstractions and trajectory constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* , pp. 873–879.

Bonet, B., Frances, G., & Geffner, H. (2019). Learning features and abstract actions for computing generalized plans. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 2703–2710.

Bonet, B., De Giacomo, G., Geffner, H., Patrizi, F., & Rubin, S. (2020). High-level programming via generalized planning and LTL synthesis. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)* , pp. 152–161.

Bonet, B., & Geffner, H. (2020). Qualitative Numeric Planning: Reductions and complexity. *Journal of Artificial Intelligence Research (JAIR)*, *69*, 923–961.

Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92–103.

Calvanese, D., De Giacomo, G., & Vardi, M. Y. (2002). Reasoning about actions and planning in ltl action theories. *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)* , *2*, 593–602.

Camacho, A., Bienvenu, M., & McIlraith, S. A. (2019). Towards a unified view of AI planning and reactive synthesis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 58–67.

Camacho, A., & McIlraith, S. A. (2016). Strong-cyclic planning when fairness is not a valid assumption. In *Proceedings of Knowledge-based Techniques for Problem Solving and Reasoning Workshop (KnowProS)*.

Camacho, A., Triantafillou, E., Muise, C. J., Baier, J. A., & McIlraith, S. A. (2017). Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3716–3724.

Chatterjee, K., Chmelík, M., & Davies, J. (2016). A symbolic sat-based algorithm for almost-sure reachability with small strategies in pomdps. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3225–3232.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 875–881.

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

Cimatti, A., Roveri, M., & Traverso, P. (1998). Strong planning in non-deterministic domains via model checking. In *Proceedings of the International Conference on AI Planning & Scheduling (AIPS)*, pp. 36–43.

Ciolek, D. A., D'Ippolito, N., Pozanco, A., & Sardiña, S. (2020). Multi-tier automated planning for adaptive behavior. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 66–74.

Daniele, M., Traverso, P., & Vardi, M. Y. (1999). Strong cyclic planning revisited. In *Proceedings of the European Conference on Planning (ECP)* , Vol. 1809 of *LNCS*, pp. 35–48. Springer.

De Giacomo, G., & Vardi, M. (2015). Synthesis for LTL and LDL on finite traces. In *Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1558–1564.

De Giacomo, G., & Vardi, M. Y. (1999). Automata-theoretic approach to planning for temporally extended goals. In *Proceedings of the European Conference on Planning (ECP)* , Vol. 1809 of *LNCS*, pp. 226–238. Springer.

D'Ippolito, N., Rodríguez, N., & Sardiña, S. (2018). Fully observable non-deterministic planning as assumption-based reactive synthesis. *Journal of Artificial Intelligence Research (JAIR)*, *61*, 593–621.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *6*(3), 1–238.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2019). Multi-shot ASP solving with Clingo. *Theory and Practice of Logic Programming*, *19*(1), 27–82.

Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Morgan-Claypool.

Geffner, T., & Geffner, H. (2018). Compact policies for non-deterministic fully observable planning as SAT. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 88–96.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. A., & Bowen, K. (Eds.), *Proc. of the Fifth International Conference on Logic Programming*, pp. 1070–1080. The MIT Press.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *Proceedings of the International Conference on AI Planning & Scheduling (AIPS)*, pp. 44–53.

Hu, Y., & De Giacomo, G. (2011). Generalized planning: Synthesizing plans that work for multiple environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* , pp. 918–923.

Lifschitz, V. (2019). *Answer set programming*. Springer.

Littman, M. L., Goldsmith, J., & Mundhenk, M. (1998). The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research (JAIR)*, *9*, 1–36.

Luttenberger, M., Meyer, P. J., & Sickert, S. (2020). Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, *57*(1-2), 3–36.

Mattmüller, R., Ortlieb, M., Helmert, M., & Bercher, P. (2010). Pattern database heuristics for fully observable nondeterministic planning.. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 105–112.

Muise, C., McIlraith, S. A., & Beck, J. C. (2012). Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 172–180.

Patrizi, F., Lipovetzky, N., & Geffner, H. (2013). Fair LTL synthesis for non-determinsistic systems using strong cyclic planners. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* , pp. 2343–2349.

Pnueli, A., & Rosner, R. (1989). On the synthesis of an asynchronous reactive module. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 652–671.

Rintanen, J. (2004). Complexity of planning with partial observability.. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 345–354.

Rodriguez, I. D., Bonet, B., Sardiña, S., & Geffner, H. (2021). Flexible fond planning with explicit fairness assumptions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 290–298.

Srivastava, S., Immerman, N., & Zilberstein, S. (2011a). A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, *175*(2), 615–647.

Srivastava, S., Zilberstein, S., Immerman, N., & Geffner, H. (2011b). Qualitative numeric planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 1010–1016.