Strong-Cyclic Planning when Fairness is Not a Valid Assumption

Alberto Camacho
Department of Computer Science
University of Toronto, Canada
acamacho@cs.toronto.edu

Sheila A. McIlraith
Department of Computer Science
University of Toronto, Canada
sheila@cs.toronto.edu

Abstract

We address the class of fully-observable non-deterministic (FOND) planning problems where non-deterministic actions are not guaranteed to display fair behaviour. Typical solutions to FOND planning problems either guarantee goal reachability with a bounded number of transitions (so-called strong acyclic solutions), or are predicated on a fairness assumption that presumes each action effect occurs infinitely often when the action is applied infinite times in the same state (so-called strong-cyclic solutions). We introduce the FOND$^+$ planning model, that extends the FOND model with a more informed description of the action transitions. Solutions to FOND$^+$ problems guarantee goal reachability, even when the classical fairness assumption is not valid. We characterize different topologies, or classes, of solutions showing that typical strong acyclic, and fair strong-cyclic planning are particular cases. Moreover, we characterize a class of FOND$^+$ solutions that also solve 1-primary normative fault-tolerant problems, and are robust to any finite number of faults during execution. We present algorithms to solve each class of problems. Finally, we present the results of one of our algorithms in the search for so-called normative solutions for a selection of *blocksworld* problems where the classical fairness assumption is not valid.

1 Introduction

Most real-world dynamical systems are best modeled using non-deterministic action effects, reflecting our inability to characterize, with certainty, the outcome of actions, and in some cases to additionally model the effects of exogenous events. A number of different frameworks have been proposed to model and plan with non-deterministic transition systems. For the purposes of this paper, we limit our discussion to fully-observable systems. *Markov Decision Processes* (MDPs) [Puterman, 1994], for example, model the effect of an action in each state as a probability distribution over the successor states. In contrast, many *probabilistic planning* systems (e.g. [Kolobov *et al.*, 2009]) exploit a compact symbolic description of the actions and their effects that associates a non-zero probability with alternative action outcomes. While less expressive than MDPs, such systems are often easier to solve and show better scalability.

In cases where the stochasticity of a system is not well understood, or deemed unnecessary, *Fully Observable Non-Deterministic* (FOND) planning [Cimatti *et al.*, 2003] provides a framework, similar to probabilistic planning, but with the exception that the effects of actions are non-deterministic without commitment to the probability of alternative action effects. Solutions to FOND planning problems are policies that map states into actions. Cimatti *et al.* [2003] identified three different classes of solutions to FOND problems: *weak, strong, and strong-cyclic*. Weak solutions are plans that achieve the goal, but without guarantees. Strong solutions guarantee goal achievement in *all* executions. This condition is too restrictive, as FOND problems often do not have strong solutions. The class of strong-cyclic solutions is an alternative, and guarantees goal achievement provided that *all* executions are *fair*. When the fairness assumption is not valid, policy executions may not achieve the goal.

In 2004, Jensen *et al.* introduced the notion of *fault-tolerant* planning, motivated by two observations. First, they interpreted non-determinism in many real-world problems as the consequence of infrequent errors (so-called *faults*) during execution. Second, they claimed that in general, no action is ever guaranteed to achieve non-faulty behaviour. The resultant framework for *fault-tolerant* planning includes complete knowledge about which state transitions are *normative* and which are otherwise *faulty*. Solutions attempt to reach a prescribed goal under the assumption that no more than κ faults will occur during execution.

In this paper we recognize that there exist non-deterministic transition systems where the fairness assumption does not hold, and the challenge we address is to find solutions that guarantee goal achievement when strong FOND solutions do not exist. To this end, we introduce the FOND$^+$ planning framework as an extension to the FOND framework that includes a more informed description of transitions. The FOND$^+$ framework makes it possible to find policies that guarantee goal achievement when the classical fairness assumption is not valid. Alternatively, the FOND$^+$ framework can be informed with a description of the normative behaviour of the system. In this case, solutions to certain

FOND$^+$ problems are also solutions to 1-primary normative fault-tolerant planning problems. We define different classes of solutions to FOND$^+$ problems, and introduce algorithms to find solutions for each class. We identify a continuum of FOND$^+$ solutions that includes strong, strong-cyclic, and fault-tolerant planning solutions in extreme cases. This result is perhaps most related to the work done by Trevizan *et al.* [2007], which defines a framework to deal simultaneously with risk and uncertainty in an attempt to unify probabilistic and non-deterministic planning. The results presented in this paper open the door to further advancements in the unification of FOND, fault-tolerant, and probabilistic planning frameworks.

2 Background

A *non-deterministic planning domain* is a tuple $\mathcal{D} = \langle \mathcal{F}, S, \mathcal{A}, T \rangle$, where $\mathcal{F}$ is a set of propositions; $S \subseteq 2^{\mathcal{F}}$ is a finite set of states, typically represented by the set of propositions that hold; $\mathcal{A}$ is a finite set of actions; and $T : S \times \mathcal{A} \to 2^S$ is a *transition function*.

An actions $a \in \mathcal{A}$ is defined in terms of its *preconditions* Pre_a and *effects* Eff_a. $Pre_a \subseteq \mathcal{F}$ is the set of propositions that need to hold true in a state s for a to be applicable in s, i.e., $Pre_a \subseteq s$. $Eff_a = \langle Eff_a^1, \ldots, Eff_a^n \rangle$ is a finite set of possible *effects* of a. Each effect $e \in Eff_a$ is a tuple of the form $e = \langle Add_e, Del_e \rangle$, where Add_e and Del_e are sets of propositions to be added and deleted, respectively, when an action transition is applied. The *progression* of s with respect to action a and effect e is the updated state $Prog(s, a, e) = (s \setminus Del_e) \cup Add_e$. Finally, the result of applying a in s is one of the states in $T(s, a) = \bigcup_{e \in Eff_a} Prog(s, a, e)$. A state $s' \in T(s, a)$ is said to be an *outcome* of a in s, and the triplet (s, a, s') is called the *transition*. An *execution* of domain $\mathcal{D}$ in state s_0 is a state-action sequence $s_0, a_0, s_1, a_1 \ldots$ such that a_i is applicable in s_i for each $i = 0, 1 \ldots$, and s_{i+1} is one of the outcomes of a_i in s_i chosen non-deterministically by the system. When $s' = Prog(s, a, e)$, the *regression* of s' with respect to action a and effect e is the *partial state* $Regr(s', a, e) = (s' \setminus Add_e) \cup Pre_a$. Conceptually, $Regr(s', a, e)$ is the minimal subset of propositions that a state s needs to entail so that s' is an outcome of a in s. Actions can be applied in partial states, whose progression and regression is defined as above (cf. [Muise *et al.*, 2012]).

2.1 Fully Observable Non-Deterministic Planning

A *fully-observable non-deterministic* (FOND) planning problem is a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, S_G \rangle$, where $\mathcal{D} = \langle \mathcal{F}, S, \mathcal{A}, T \rangle$ is a non-deterministic planning domain, $s_0 \in S$ is the *initial state*, and $S_G \subseteq S$ is a set of *goal states*. Solutions to FOND planning problems are policies that map states into actions. We say that π is *well-defined* when $\pi(s)$ is defined in every state reachable by π. *Executions* of a well-defined policy π (also called *plan executions*) are executions $s_0, a_0, s_1, a_1 \ldots$ where, for each $i = 0, 1 \ldots$, $a_i = \pi(s_i)$. When the execution finishes in a goal state $s_n \in S_G$, the sequence of actions $P = a_0, a_1, \ldots, a_{n-1}$ is called a *plan*.

Definition 1 (adapted from [Cimatti *et al.*, 2003])**.** *We say that an execution σ is* unfair *when a state-action tuple s, a appears infinitely often in σ, but the transition (s, a, s') occurs a finite number of times for an outcome $s' \in T(s, a)$. Executions that are not unfair are said to be* fair.

Definition 2 (adapted from [Cimatti *et al.*, 2003])**.** *A solution to a FOND problem is strong cyclic when all executions result in either finite sequences of states that terminate in a goal state, or are infinite and unfair.*

Cimatti *et al.* [2003] distinguish three classes of solutions to a FOND planning problem: *weak*, *strong*, and *strong cyclic*. Weak solutions are plans that lead the agent from the initial state to a goal state, but with no guarantees due to the non-determinism of the actions. Strong solutions are plans that guarantee goal achievement. Necessarily, strong solutions are acyclic, and achieve the goal in a bounded number of steps. Finally, strong cyclic solutions are plans that are guaranteed to achieve the goal, but predicated on the fairness assumption given by Definition 1. The fairness assumption presumes that, for each state s and each action a applicable in s, every outcome of a occurs infinitely many times if a is applied in s infinitely often. Strong solutions are also strong-cyclic, but the opposite is not always true.

2.2 Fault-Tolerant Planning

A *fault-tolerant* planning problem is a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, F, \kappa \rangle$, where $\mathcal{D} = \langle \mathcal{F}, S, \mathcal{A}, T \rangle$ is a non-deterministic planning domain, $s_0 \in S$ is the *initial state*, $S_G \subseteq S$ is a set of *goal states*, F is an *exception model* as described in Definition 3, and κ is a constant [Jensen *et al.*, 2004; Domshlak, 2013]. Informally, a fault-tolerant planning task $\mathcal{P}$ requires planning for goal achievement under the assumption that no more than κ *exceptions* will occur during plan execution.

Definition 3 (adapted from [Domshlak, 2013])**.** *For a non-deterministic planning domain $\mathcal{D}$, an* exception model *is a function $F : \bigcup_{a \in \mathcal{A}} Eff_a \to \mathbb{N}$, computable in time polynomial in the size of $\mathcal{P}$. If, for each action $a \in \mathcal{A}$, $|\{e \mid e \in Eff_a, F(e) = 0\}| \leq \alpha$, then F is called* α-primary. *Likewise, if, for each action $a \in \mathcal{A}$, $|\{e \mid e \in Eff_a, F(e) = 0\}| > 0$, then F is called* normative.

In a fault-tolerant planning problem, actions have two types of effects. *Primary* effects model the normative behaviour of the system, while *faulty* effects model one or more failures (faults) of the system. Primary effects, e, have $F(e) = 0$, whereas $F(e) > 0$ when e is faulty. Intuitively, the exception model F maps each effect to the *number of exceptions*, or faults, associated with it.

Solutions to a fault-tolerant planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, F, \kappa \rangle$ are κ-plans, or policies that achieve the goal under the assumption that no more than κ faults will occur during execution. Formally, a plan P is κ-*admissible* when its execution generates the state-effect sequence $(s_0, e_0, \ldots, s_i, e_i, \ldots)$ with $\Sigma_i F(e_i) \leq \kappa$. A policy is a κ-plan when all plan executions are κ-admissible and reach the goal. Solutions to fault-tolerant planning problems do not assume fairness as done in strong-cyclic FOND planning, and are robust to occurrence of up to κ faults. Note that the parameter κ that sets the maximum number of exceptions in a solution is part of the problem description, and needs to be known, computed, or otherwise guessed in advance.

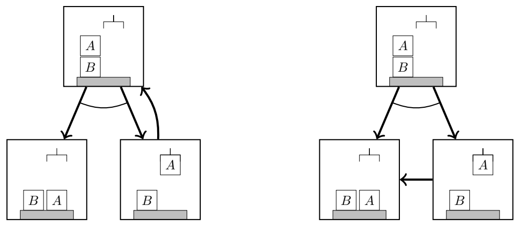
Jensen *et al.* [2004] introduced the model for fault-tolerant planning, and addressed 1-primary normative models with the rationale that these satisfactorily address non-determinism originating from many real-world physical system models. Fault-tolerant planning has found applications in the robot community (e.g. [Lussier *et al.*, 2007]). More recently, fault-tolerant planning has been studied by Domshlak [2013] and Delgrande and Levesque [2013]. Recent work by Pineda *et al.* [2013] addressed the problem of fault-tolerant planning under uncertainty, tackling the problem from the perspective of a stochastic shortest-path (SSP) problem.

3 Planning with Unfair Non-Determinism

We seek to generate solutions to goal-oriented planning problems with non-deterministic action effects where the fairness assumption does not necessarily hold. Different notions of fairness over executions have been discussed in the literature. In particular, D'Ippolito *et al.* [2011] defined *strong independent fairness*, a notion that presumes it is known, for each action, which outcomes are *good* and which ones are *failures* (similar to the distinction made in fault-tolerant planning), and requires failures and assumptions – normally, temporally extended constraints – on the environment behaviour to be independent. Sardina and D'Ippolito [2015] defined different logical characterizations of fairness based on the notions introduced by D'Ippolito *et al.* [2011]. Kupferman and Vardi [1996] proved that verification of fair transition systems is PSPACE-complete.

Our premise is that in many transition systems, and in concrete non-deterministic planning domains, it *is known* which state-action pairs s, a are guaranteed to produce, eventually, certain outcomes s' if a would be applied in s infinitely often. While not true in general, it is a valid assumption for many real-world transition systems. For example, it is known that pressing the elevator's button on the wall repeatedly causes the elevator's doors to open. On the other hand, it is known there is no guarantee that a fixed number will be awarded in a lottery, even if the lottery is played infinitely many times under identical conditions. When fairness does not hold, solutions need to be robust to non-determinism, and goal achievement in plan executions should not rely on a particular transition for which there is no guarantees of occurrence. A similar argument can be made in non-deterministic domains that have known normative behaviour. In the scope of this paper, we focus our attention on 1-primary normative fault-tolerant planning problems. In these problems, solutions need to be robust to failures, and the goal needs to be achievable whenever the system manifests its normative behaviour.

The following example motivates the need for policy generation mechanisms that guarantee goal achievement in those non-deterministic domains where the fairness assumption is not valid. Further, it illustrates a parallelism between solutions to goal-oriented planning problems where fairness governs the non-determinism of the actions, and those where the non-deterministic actions have primary and faulty effects.



(a) Strong-Cyclic Solution (b) Normative Solution

Figure 1: Different policies for the *blocks-world* domain. The *pick-up* action is non-deterministic, and may pick-up a certain block or drop it on the table. The *put-on-block* and *put-block-on-table* actions are deterministic.

Illustrative Example

Consider the *blocks-world* domain, where block A is initially on top of block B, and the goal is to put A on the table. The agent can pick up blocks and put them on top of other blocks, or on the table. The *pickup-block* action is non-deterministic and the gripper may drop the block onto the table, but is guaranteed to eventually pick-up the block if tried repeated times. The *put-on-block* (resp. *put-block-on-table*) action is deterministic and puts the block held by the gripper on top of another block (resp. on the table). Strong-cyclic FOND solutions that assume fairness may rely on the eventual occurrence of the faulty effect of *pickup-block* in order to reach the goal. Figure 1a shows the strong-cyclic solution that picks up block A and puts it on top of B repeatedly until block A is dropped on the table. If it is the case that the effect of the *pickup-block* action that drops the block is not guaranteed to occur during execution, the solution described above does not guarantee goal achievement. In contrast, the solution illustrated in Figure 1b is strong, and goal achievement is guaranteed.

A similar pattern appears when we consider that the block being picked up is the normative behaviour of the *pickup-block* action. The solution described in Figure 1a loops until the *pick-up* action is faulty, which is certainly not a good quality solution because the normative behaviour of the system can lead to the goal, regardless of whether the faulty effect is guaranteed to occur or not. Note that the solution described in Figure 1b is of good quality because executions lead to the goal when the behaviour of the system is normative, and the solution is robust to non-determinism.

3.1 The FOND$^+$ Model

A *Fully Observable Non-Deterministic* Planning problem with *Labeled UnfairnessS* (for short, FOND$^+$ problem) is a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L \rangle$, where $\mathcal{D} = \langle \mathcal{F}, S, \mathcal{A}, T \rangle$ is a non-deterministic planning domain, $s_0 \in S$ is the *initial state*, $S_G \subseteq S$ is a set of *goal states*, and $L : S \times \mathcal{A} \times S \rightarrow \{\mathtt{F}, \mathtt{U}\}$ is a *labeling function* that maps each triplet $(s, a, s') \in S \times \mathcal{A} \times S$ into one of the symbols $\mathtt{F}$ or $\mathtt{U}$. The semantics of L supports different interpretations. Before elaborating, we first define what constitutes an *L-fair* execution. For a non-deterministic planning domain $\mathcal{D} = \langle \mathcal{F}, S, \mathcal{A}, T \rangle$, and labeling function $L : S \times \mathcal{A} \times S \rightarrow \{\mathtt{F}, \mathtt{U}\}$, we say that an execution in state s_0 is *L-unfair* when there exists a state-action tuple (s, a) such that (i) (s, a) appears infinitely often, and (ii) there exists a transition (s, a, s') such that $L(s, a, s') = \mathtt{F}$ and (s, a, s') occurs a finite number of times. Executions that are not L-unfair are said to be *L-fair*. Note that fairness, as defined by Cimatti *et al.* [2003], is a particular case of L-fair that occurs when L assigns $\mathtt{F}$ to all transitions. Solutions to a FOND$^+$ problem are policies that guarantee goal achievement, predicated on the assumption that all executions of $\mathcal{D}$ in s_0 are L-fair.

Definition 4. *A* FOND$^+$ problem *is a tuple* $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L \rangle$*, where* $\mathcal{D} = \langle \mathcal{F}, S, \mathcal{A}, T \rangle$ *is a non-deterministic planning domain,* $s_0 \in S$ *is the* initial state*,* $S_G \subseteq S$ *is a set of* goal states*, and* $L : S \times \mathcal{A} \times S \to \{\mathtt{F}, \mathtt{U}\}$ *is a* labeling function*.*

Definition 5. *Solutions to a FOND$^+$ problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L \rangle$ are policies that guarantee goal achievement, predicated on the assumption that all executions of $\mathcal{D}$ in s_0 are L-fair.*

In the scope of this paper, we focus our attention on two different interpretations of the labeling function. The first interpretation assumes it *is known*, for each state-action pair (s, a), which transitions (s, a, s') are *not* guaranteed to occur infinitely often in executions where the pair (s, a) appears an infinite number of times. In those transitions, we assign $L(s, a, s') = \mathtt{U}$. Otherwise, we assign $L(s, a, s') = \mathtt{F}$. With this interpretation, all plan executions are L-fair and FOND$^+$ solutions guarantee goal achievement during execution. In other words, FOND$^+$ solutions are robust to non-deterministic transitions during plan execution, and goal achievement does *not* rely on any specific transition for which there is no guarantees of eventual occurrence. The FOND$^+$ model overcomes difficulties of the FOND model when strong solutions do not exist, as strong-cyclic solutions do not guarantee goal achievement, in general, when Cimatti *et al.*'s fairness assumption is not valid.

The second interpretation we examine describes the normative behaviour of the system, and assigns $L(s, a, s') = \mathtt{F}$ (resp. $L(s, a, s') = \mathtt{U}$) when s' follows from a primary (resp. faulty) effect of a in s. With this interpretation, an execution is L-fair when it is finite, or when it is infinite and for each state-action pair (s, a) that appears infinitely often, the system responds with each primary effect of a in s infinitely often. As we will see later, this interpretation is particularly interesting in certain FOND$^+$ planning problems, and makes it possible to find policies that guarantee goal achievement under the assumption that the number of transitions produced by faulty effects during plan execution is finite.

In the blocksworld example introduced above, and with both interpretations, all transitions t have $L(t) = \mathtt{F}$ except the transitions produced by the effect of the *pick-up* action that drops the block on the table, which have $L(t) = \mathtt{U}$. With this labeling, the policy depicted in Figure 1b is a FOND$^+$ solution, whereas the policy depicted in Figure 1b is not.

3.2 Classes of Solutions

We characterize different classes of solutions to FOND$^+$ problems according to the fairness of plan executions. Namely, we distinguish between *strictly fair*, *strictly unfair*, and *mixed* solutions. Furthermore, we characterize the

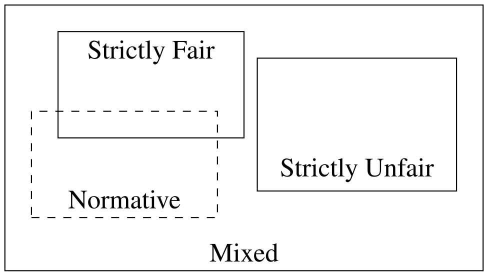


Figure 2: Distribution of different classes of solutions to FOND$^+$ planning problems.

class of *normative* solutions that is particularly interesting for its practical applications and proximity with solutions to 1-primary normative fault-tolerant planning problems.

A solution π to a FOND$^+$ problem is *strictly fair* when all transitions t produced by L-fair plan executions have $L(t) = \mathtt{F}$. Strictly fair solutions to a FOND$^+$ problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L \rangle$ are strong-cyclic solutions to the FOND problem $\mathcal{P}' = \langle \mathcal{D}, s_0, S_G \rangle$ that results from ignoring the labeling function in $\mathcal{P}$. The opposite is not always true, and strong-cyclic solutions to $\mathcal{P}$' may or may not be strictly fair solutions to $\mathcal{P}$ – only when all transitions t produced by plan executions have $L(t) = \mathtt{F}$. In general, a strong cyclic solution can belong to any class of FOND$^+$ solutions. Like strong-cyclic solutions, strictly fair solutions can contain cycles.

Definition 6. *A solution π to a FOND$^+$ problem is* strictly fair *when all transitions t produced by L-fair plan executions have* $L(t) = \mathtt{F}$.

The class of *strictly unfair* solutions is analogous to the class of strictly fair solutions, this time requiring transitions t produced by L-fair plan executions have $L(t) = \mathtt{U}$. Strictly unfair solutions to a FOND$^+$ problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L \rangle$ are strong solutions to the FOND problem $\mathcal{P}' = \langle \mathcal{D}, s_0, S_G \rangle$ that results from ignoring the labeling function in $\mathcal{P}$. The opposite is not always true either, and strong solutions to $\mathcal{P}$' are strictly unfair solutions to $\mathcal{P}$ only when all transitions t produced by plan executions have $L(t) = \mathtt{U}$. In general, a strong solution can belong to any class of FOND$^+$ solutions. Like strong solutions, strictly unfair solutions are acyclic.

Definition 7. *A solution π to a FOND$^+$ problem is* strictly unfair *when all transitions t produced by L-fair plan executions have* $L(t) = \mathtt{U}$.

The class of *mixed* solutions completes the space of solutions to a FOND$^+$ planning problem. Mixed solutions are those that are neither strictly fair nor strictly unfair. Plan executions $s_0, a_0, s_1, a_1 \ldots s_n$ are such that s_n is a goal state, and may contain cycles. More precisely, there may exist $k < m < n$ such that $s_k = s_m$. As all infinite plan executions need to be L-unfair (cf. Definition 5), at least one of the outcomes of s_i by a_i, for a certain $k \le i \le m$, must yield a transition (s_i, a_i, s'_{i+1}) with $L(s_i, a_i, s'_{i+1}) = \mathtt{F}$.

Definition 8. *A solution π to a FOND$^+$ problem is* mixed *when it is neither strictly fair nor strictly unfair.*

Finally, we characterize the class of *normative* solutions, which are particularly interesting when the labeling function describes the normative behaviour of the system – i.e., when transitions (s, a, s') with label $L(s, a, s') = \texttt{F}$ are exactly those produced by a primary effect of a in s. A solution π is normative when, in each state s, reachable by π: (i) there exists a plan execution in s that reaches the goal and such that all transitions t have $L(t) = \texttt{F}$, and (ii) exactly one outcome of s by $\pi(s)$ produces a transition t with $L(t) = \texttt{F}$. The class of normative solutions intersects with the class of strictly fair solutions in those solutions that are deterministic plans (cf. Figure 2).

Definition 9. *A FOND$^+$ solution π is* normative *when, in each state s, reachable by π: (i) there exists a plan execution in s that reaches the goal and such that all transitions t have $L(t) = \texttt{F}$, and (ii) exactly one outcome of s by $\pi(s)$ produces a transition t with $L(t) = \texttt{F}$.*

With the interpretation of the labeling function that describes the normative behaviour of the system, policy actions of normative solutions are required to have, by definition, one primary effect. Solutions need to be robust to faulty transitions and guarantee goal achievement, at any point during execution, when the system follows its normative behaviour. In order to facilitate comparison with the fault-tolerant planning model, we assume the value $L(s, a, s')$ depends only on the *effect* that produces the outcome s', but not on the state s itself. Formally, we assume $L(s, a, Prog(s, a, e)) = L(s', a, Prog(s', a, e))$ for all actions $a \in \mathcal{A}$, all effects e of a, and all pairs of states $s, s' \in S$ in which a is applicable. When this property holds, the labeling function can be described compactly as a function of the actions effects.

With the restrictions described above, normative solutions to a FOND$^+$ problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L\rangle$ are also 1-primary normative solutions [1] to fault-tolerant planning problems $\mathcal{P}' = \langle \mathcal{D}, s_0, S_G, F, \kappa\rangle$ with an exception model F that assigns faults $F(e) > 0$ to effects e that produce transitions $(s, a, Prog(s, a, e))$ such that $L(s, a, Prog(s, a, e)) = \texttt{U}$. Note that normative FOND$^+$ solutions are robust to occurrence of *any* possible number of faults during execution, as opposed to standard fault-tolerant solutions, whose executions are guaranteed to be robust up to a bounded number of faults κ specified by the model.

Theorem 1. *Let $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L\rangle$ be a FOND$^+$ problem, and let $\mathcal{P}' = \langle \mathcal{D}, s_0, S_G, F, \kappa\rangle$ be a* 1*-primary normative fault-tolerant planning problem with exception model F that assigns faults $F(e) > 0$ to effects e that produce transitions $(s, a, Prog(s, a, e))$ such that $L(s, a, Prog(s, a, e)) = \texttt{U}$. Then, FOND$^+$ normative solutions to $\mathcal{P}$ are* 1*-primary normative solutions to $\mathcal{P}$' for any $\kappa < \infty$.*

The practicality of 1-primary normative fault-tolerant planning to model and solve many real-world physical problems has been discussed by Jensen *et al.* [2004] and Domshlak [2013]. Normative FOND$^+$ solutions overcome certain limitations of the 1-primary normative fault-tolerant planning model and its solutions. More precisely, normative FOND$^+$ solutions are robust to *any* number of faulty transitions during execution. This is an advantage with respect to solutions to fault-tolerant planning, which are robust to a limited number of κ faults during execution. Besides, κ is a parameter of the model that has to be fixed prior search, even if a solution robust to *any* number of faults exists.

[1]For convenience, we informally say that a solution π is α-primary (resp. normative) when $|\{e \mid e \in \mathit{Eff}_a, F(e) = 0\}| \leq \alpha$ (resp. $|\{e \mid e \in \mathit{Eff}_a, F(e) = 0\}| > 0$) in all effects from plan executions of π. Technically, the α-primary normative model introduced by Domshlak [2013] requires F to be α-primary and normative globally, which is a harder constraint.

4 Search Algorithms for FOND$^+$

This section presents algorithms to search for strictly fair, strictly unfair, and normative solutions to FOND$^+$ planning problems. This is followed by an algorithm that is, in principle, less efficient, but can search for general solutions to FOND$^+$ problems. Notwithstanding, the algorithms presented below are intended to provide a baseline for future improvements, and their main purpose is to demonstrate the existence of methods to find solutions to FOND$^+$. We decided to follow a branch-and-bound search scheme, motivated by its efficiency in the search for strong-cyclic solutions to FOND problems [Muise *et al.*, 2012]. The similarity between the FOND$^+$ and FOND planning models suggests that a similar technique may be effective in the search for FOND$^+$ solutions.

4.1 Search for Strictly Fair Solutions

By definition, all transitions t produced by plan executions of strictly fair solutions are such that $L(t) = \texttt{F}$. It is straightforward to see that strictly fair solutions to a FOND$^+$ problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L\rangle$ are all and only those strong-cyclic solutions to the FOND problem $\mathcal{P}' = \langle \mathcal{D}', s_0, S_G\rangle$. Intuitively, $\mathcal{D}'$ is like $\mathcal{D}$, but the actions applicable in a given state s are restricted to those a's that only yield transitions (s, a, s') labeled with $L(s, a, s') = \texttt{F}$. This constraint can be implemented by means of extra dummy fluents in $\mathcal{D}'$. Alternatively, when L accepts a compact description such that its value depends only on the effects of the actions that yield each transition, we can safely prune an action a from $\mathcal{D}'$ when not all its effects yield transitions t with $L(t) = \texttt{F}$.

For a FOND$^+$ problem $\mathcal{P}$, the algorithm consists of two steps:

1. $\mathcal{P}$ is relaxed into a FOND problem $\mathcal{P}'$ as described above.
2. A sound and complete strong-cyclic FOND planner – e.g. PRP [Muise *et al.*, 2012] – is used to search for a strong-cyclic solution to $\mathcal{P}'$, which is returned as a strictly fair solution to $\mathcal{P}$.

Since strictly fair solutions to $\mathcal{P}$ are the strong-cyclic solutions to the FOND problem $\mathcal{P}'$, soundness and completeness of the algorithm derives from soundness and completeness of the strong-cyclic FOND planner.

4.2 Search for Strictly Unfair Solutions

By definition, all transitions t produced by plan executions of strictly unfair solutions are such that $L(t) = \texttt{U}$. The algorithm to find strictly unfair solutions is somewhat analogous to the algorithm used to find strictly fair solutions. This time, however, we exploit the correspondence between strictly unfair solutions to a FOND$^+$ problem $\mathcal{P} = \langle \mathcal{D}, s_0, S_G, L\rangle$

and strong FOND solutions to the FOND problem $\mathcal{P}' = \langle \mathcal{D}', s_0, S_G \rangle$. Intuitively, $\mathcal{D}'$ is like $\mathcal{D}$, but the actions applicable in a given state s are restricted to those a's that only yield transitions (s, a, s') labeled wth $L(s, a, s') = \mathtt{U}$. This constraint can also be implemented by means of extra dummy fluents in $\mathcal{D}'$. Alternatively, when L accepts a compact description such that its value depends only on the effects of the actions that yield each transition, we can safely prune an action a when not all its effects yield transitions t with $L(t) = \mathtt{U}$.

For a FOND$^+$ problem $\mathcal{P}$, the algorithm consists of two steps:

1. $\mathcal{P}$ is relaxed into a FOND problem $\mathcal{P}'$ as described above.
2. A sound and complete strong FOND planer – e.g. [Jaramillo *et al.*, 2014] – is used to search for a strictly unfair solution to $\mathcal{P}'$, which is returned as a strictly unfair solution to $\mathcal{P}$.

Since strictly unfair solutions to $\mathcal{P}$ are the strong solutions to the FOND problem $\mathcal{P}'$, soundness and completeness of the algorithm derives from soundness and completeness of the strong FOND planner.

4.3 Search for Normative Solutions

Normative solutions are those where, for every reachable state s, there exists a plan for s for which all transitions t have $L(t) = \mathtt{F}$. Algorithm 1 exploits this property. Intuitively, the algorithm explores plans that exclusively produce transitions t such that $L(t) = \mathtt{F}$, which are extended into robust policies that also consider occurrence of other transitions.

Without loss of generality, we assume the problem has the property that all actions produce, in each state s in which a is applicable, exactly one transition t such that $L(t) = \mathtt{F}$. If this property is not true, and the value of $L(s, a, s')$ in any state s depends only on the effect of a that generates outcome s', actions that do not have exactly one effect that produces a transition t such that $L(t) = \mathtt{F}$ can be safely pruned in a preprocessing stage. Alternatively, a constraint during search can be added, so that a state s can be expanded by action a only when exactly one outcome produces a transition t such that $L(t) = \mathtt{F}$. If this check is not performed, Algorithm 1 still finds solutions (perhaps non-normative mixed solutions) that guarantee goal achievement when the interpretation of the labeling function describes fairness, but goal achievement during execution is not guaranteed anymore when the labeling function describes the normative behaviour of the system.

Procedure POLICYSEARCH in Algorithm 1 maintains three separate stacks: *Open*, *Seen*, and *Deferred*. The following search procedure is performed until policy π does not change with respect to the previous policy found. First, *Open* is initialised with the initial state s_0 and *Seen*, *Deferred*, and π are initialised to empty sets. For each state s in *Open* being processed, for which π is undefined, procedure GENFAIRPLAN searches a plan P for s that exclusively produces transitions t such that $L(t) = \mathtt{F}$. Then, policy π is extended with information extracted from P, and the successors of s by $\pi(s)$ are stacked. This is done by procedure STACKSUCCESSORS, which stacks each successor s' into *Open* when

Algorithm 1 Search for Normative Solutions

```
1:  procedure PolicySearch($\mathcal{V}, s_0, s_\star, \mathcal{A}$)
2:      while $\pi$ changes do
3:          $\pi \leftarrow \emptyset$
4:          $Open \leftarrow s_0$
5:          $Seen \leftarrow \{\}$
6:          $Deferred \leftarrow \{\}$
7:          while $Open \neq \emptyset \vee Deferred \neq \emptyset$ do
8:              $s =$ Unstack()
9:              if $s \not\models s_\star \wedge s \notin Seen$ then
10:                 $Seen.add(s)$
11:                 if $\pi(s)$ is undefined then
12:                     GenFairPlan($\mathcal{V}, s, s_\star, \mathcal{A}, \pi$)
13:                 end if
14:                 if $\pi(s)$ is defined then
15:                     $\langle p, a \rangle = \pi(s)$
16:                     StackSuccessors($s, a$)
17:                 else break
18:                 end if
19:             end if
20:         end while
21:         ProcessDeadends()
22:     end while
23:     return $\pi$
24: end procedure
```

Add Successors to Corresponding Stack

```
25: function StackSuccessors($s, a$)
26:     for $e \in \text{Eff}_a$ do
27:         $s' = Prog(s, a, e)$
28:         if $L(s, a, s') = \mathtt{F}$ then
29:             $Open.add(s')$
30:         else
31:             $Deferred.add(s')$
32:         end if
33:     end for
34: end function
```

Get Next Node to Explore

```
35: procedure Unstack
36:     if $Open_F \neq \emptyset$ then
37:         $s = Open.pop()$
38:     else
39:         $s = Deferred.pop()$
40:     end if
41:     return $s$
42: end procedure
```

$L(s, \pi(s), s') = \mathtt{F}$, and stacks s' into *Deferred* otherwise. The construction of the policy is performed by, iteratively, unstacking a new state from *Open*, or *Deferred* if *Open* is empty, and finding a plan with GENFAIRPLAN search. This iterative process is run until the stacks *Open* and *Deferred* are empty, or GENFAIRPLAN search for plans fails.

Procedures GENFAIRPLAN and PROCESSDEADENDS are inspired by procedures GENPLANPAIRS and PROCESSDEADENDS in the FOND planner PRP [Muise *et al.*, 2012]. GENFAIRPLAN does two things. First, it searches for a plan P for s such that all transitions t have $L(t) = \mathtt{F}$. For efficiency, such a plan may end in a goal state or in a state for which a policy action has already been computed. Second, like PRP's GENPLANPAIRS, it extends π with new state-action pairs. More precisely, for each pair $\langle s, a\rangle$ in the plan, GENFAIRPLAN computes, via regression, the relevant part p of s so that the goal is still reachable (e.g. [Waldinger, 1977; Reiter, 2001]). Finally, it extends π with the (partial) state-action pair $\langle p, a\rangle$. PRP maintains a list of forbidden state-action pairs (FSAPs) that recognisably lead to a state where no strong-cyclic policy exists. FSAPs allow to prune the search space effectively, and reduce search times considerably, in practice. We apply the same ideas in our algorithm. Whenever GENFAIRPLAN fails to find a plan for state s (i.e., s is a dead end), procedure PROCESSDEADENDS computes the reasons why s is a dead end and performs regression to create a set of forbidden (partial) state-action pairs that need to be avoided in future searches.

Algorithm 1 finds normative solutions to a FOND$^+$ problem, as proved in Theorem 2.

Theorem 2. *Algorithm 1 is sound and complete in the search for normative solutions to a FOND$^+$ problem.*

Proof sketch. Line 16 of Algorithm 1 stacks all states reachable by π, Every state a reachable by π is considered in line 8 of Algorithm 1. If no fair plan for s has been computed yet, procedure GENFAIRPLAN finds a fair plan for it. This proves soundness of the algorithm. Procedure POLICYSEARCH explores all the states space, except those forbidden state-action pairs that are pruned from search by PROCESSDEADENDS. Therefore, the algorithm is complete. □

Algorithm 1 maintains soundness and completeness when procedure STACKSUCCESSORS stacks all successors into the *Open* stack – i.e. the use of *Deferred* stack is not necessary. However, there is an advantage to employing the *Deferred* stack. The rationale appears with the interpretation that transitions with $L(t) = \mathtt{F}$ model the normative behaviour of the system. Intuitively, we want faulty transitions to be *repaired* so the system moves back to a normative state. When handling of states originated by faulty transitions ($L(t) = \mathtt{U}$) is deferred, it is more likely that plans found by procedure GENFAIRPLAN plan to a state that is already handled by the policy rather than yielding a niew plan all the way to the goal. In other words, it is more likely that those plans are repairs rather than alternative plans. This is a desired property of solutions that justifies, in principle, the use of the *Deferred* stack.

4.4 Search of FOND$^+$ Solutions

We presented algorithms to obtain strictly fair, strictly unfair, and normative solutions to FOND$^+$ problems. In this section, we present a search algorithm to obtain general solutions to FOND$^+$ problems.

For a FOND$^+$ problem $\mathcal{P}$, our algorithm is an iterated call to Algorithm 2 with parameter $\kappa = 0, 1, \ldots$ until a solution is found. Similar to normative fault-tolerant planning, each iteration fixes the maximum number κ of transitions with $L(t) = \mathtt{U}$ allowed to occur in any plan execution. However, solutions do not presume that no more than κ faulty effects will occur. The algorithm makes use of an augmented

Algorithm 2 Search of FOND$^+$ Solutions

```
1:  procedure POLICYSEARCH($\kappa, \mathcal{V}, s_0, s_\star, \mathcal{A}$)
2:      while $\pi$ changes do
3:          $\pi \leftarrow \emptyset$
4:          $Open \leftarrow s_0$
5:          $Seen \leftarrow \{\}$
6:          while $Open \neq \emptyset$ do
7:              $s = Open.pop()$
8:              if $s.ucount > \kappa$ then break
9:              end if
10:             if $s \not\models s_\star \wedge s \notin Seen$ then
11:                 $Seen.add(s)$
12:                 if $\pi(s)$ is undefined then
13:                     GENBOUNDPLAN($\kappa, \mathcal{V}, s, s_\star, \mathcal{A}, \pi$)
14:                 end if
15:                 if $\pi(s)$ is defined then
16:                     $\langle p, a\rangle = \pi(s)$
17:                     STACKSUCCESSORS($s, a$)
18:                 else break
19:                 end if
20:             end if
21:         end while
22:         PROCESSDEADENDS()
23:     end while
24:     return $\pi$
25: end procedure
```

Add Successors to Corresponding Stack

```
26: function STACKSUCCESSORS($s, a$)
27:     for $e \in \mathrm{Eff}_a$ do
28:         $s' = Prog(s, a, e)$
29:         if $L(s, a, s') = \mathtt{F}$ then
30:             $s'.ucount \leftarrow s.ucount$
31:         else
32:             $s'.ucount \leftarrow s.ucount + 1$
33:         end if
34:         $Open.add(s')$
35:     end for
36: end function
```

representation of states that includes, for each planning state s explored, a counter $s.ucount$ that records the number of transitions with $L(t) = \mathtt{U}$ produced to visit state s. The initial state has $s.ucount = 0$. For a transition (s, a, s'), the counter $s'.ucount$ equals $s.ucount$ when $L(s, a, s') = \mathtt{F}$, and $s.ucount + 1$ when $L(s, a, s') = \mathtt{U}$.

We abuse notation and say that s entails s' when $s \models s'$ and $s.ucount \leq s'.ucount$. For a state s in the *Open* stack being processed, procedure GENBOUNDPLAN searches for plans $P = s, a_0, s_1, a_1, \ldots a_n, s_n$ that either: (i) exclusively produce transitions t with $L(t) = \mathtt{F}$ and s_n is a goal state, (ii) exclusively produce transitions t with $L(t) = \mathtt{F}$ and s_n entails a partial state for which the policy action is defined. or (iii) exclusively produce transitions t with $L(t) = \mathtt{F}$ except (s_{n-1}, a_n, s_n), and all transitions produced by the progression of s_{n-1} by a_n have $L(t) = \mathtt{U}$. Intuitively, conditions (i),(ii),(iii) ensure that all L-fair plan executions achieve the goal.

In GENBOUNDPLAN, plans are regressed as in procedure

GENFAIRPLAN of Algorithm 1. Regressed states keep the counter $s.ucount$ in each state s of the plan, which is not altered by plan regression. Procedure PROCESSDEADENDS – analogous to that of Algorithm 1 – keeps track of the state-action pairs that recognisably lead to a dead end or a state for which procedure GENBOUNDPLAN fails to find a plan, where representation of states s include the counter $s.ucount$. The list of forbidden state-action pairs is initialised in each call to POLICYSEARCH. We leave further optimizations of the algorithm for future work.

Strictly speaking, solutions found by Algorithm 2 are not policies. The reason is that the action performed in an *augmented* state s, in general, may depend on the value of the counter $s.ucount$. Certainly, the action returned by a policy should depend only on the planning state, not on the counter. We have two options to mitigate for this. One, is to construct the policy that selects, for each planning state s, the action among all (augmented partial) state-action pairs $\{p, a\}$ such that $s \models p$ and $p.ucount$ is maximal. The second option is to represent the solution found by Algorithm 2 as a FSC π that keeps track, implicitly, of occurrence of transitions with $L(t) = \mathtt{U}$ along execution.

Theorem 3. *Algorithm 2 is sound and complete in the search of FOND$^+$ solutions whose plan executions produce a number of transitions t with $L(t) = \mathtt{U}$ that is bounded by κ.*

Corollary 1. *The iterated call to Algorithm 2 with parameters $\kappa = 0, 1, \ldots$ finds a solution to a FOND$^+$ problem $\mathcal{P}$, if such one exists. If a solution is found in the κ-th iteration, the number of transitions t with $L(t) = \mathtt{U}$ in plan execution is bounded by κ.*

5 Experiments

We conducted a series tests with the objective of demonstrating the need for FOND$^+$ solutions, that differ from strong-cyclic FOND solutions and that guarantee goal achievement when the classical fairness assumption is not valid. We selected a non-deterministic planning domain, the *blocksworld-new*, and a set of FOND problems. For each FOND problem, we constructed FOND$^+$ problem by labeling transitions. Finally, we compared FOND$^+$ normative solutions with strong-cyclic FOND solutions to each of the problems. We used PRP as the strong-cyclic planner; Algorithm 1 is implemented on top of PRP. In all cases (also in Algorithm 1), we configured PRP search with dead end detection disabled, and strong-cyclic-detection optimization disabled. Experiments were run on an Intel Xeon E5-2430 CPU @2.2GHz Linux server, limiting processes to 2GB of memory usage.

The *blocksworld-new* problems used in our tests have been extracted from Muise *et al.* [2012]. They are based on the *blocksworld* benchmark used in past International Probabilistic Planning Competitions (IPPCs), and include some fixes and larger problem sizes with respect to the original instances. In the *blocksworld-new* domain, the gripper can pick up a block (or two blocks at once) and put it on top of other block, or on the table. The goal is to set the blocks in a certain order. The action that puts blocks on the table is deterministic. However, the *pick-up* and *put-on-block* actions have non-deterministic effects. More precisely, the primary effect of

	Strong-Cyclic		Normative	
problem	run-time	size	run-time	size
p2	0	3	0	3
p3	0.002	5	0.016	5
p4	0.020	11	0.048	11
p5	0.070	27	0.178	27
p6	0.110	39	0.296	39
p7	0.114	32	0.270	32
p8	0.150	26	0.356	26
p9	0.278	46	0.664	46
p10	0.336	49	0.782	49
p11	0.522	120	1.936	97
p12	0.626	97	1.840	119.5
p13	0.682	57	1.810	57
p14	3.794	1117	37.10	1123
p15	1.500	278	7.814	278

Table 1: Average run-time (sec.) and policy size of FOND and FOND$^+$ solutions to *blocksworld-new* problems. Strong-cyclic solutions presume fairness, whereas computation of normative solutions is informed by the labeling function.

pick-up is to pick-up the chosen block (or the tower of two blocks). Likewise, the primary effect of *put-on-block* is to put a block (or a tower of two blocks) that is held by the gripper on top of another block. The faulty effect for both actions drops the block (or the tower of two blocks) on the table. Alternatively, we can interpret the gripper is guaranteed to succeed in picking-up the block eventually, and so on.

Table 1 summarizes the results of our tests. For each problem, we report the run-time and policy size, averaged over 10 runs per problem. Normative policies have, sometimes, equal size to strong-cyclic policies. We conjecture that in these problems strong-cyclic policies are also robust to unfairness (i.e. are also normative solutions to the FOND$^+$ problems). On the other hand, in some FOND$^+$ problems – like *p11*, *p12*, and *p14*– the sizes of the normative policy varies with respect to the size of the strong-cyclic policy of the corresponding FOND problem. In those cases, normative policies can be of greater or lesser size relative to strong-cyclic policies. As normative FOND$^+$ solutions are also solutions to the FOND problem, this illustrates that policies found by PRP are not necessarily of minimal size. Run-times of Algorithm 1 are, in general, two times greater than those of PRP in those problems where the size of the policies generated are the same. Comparing the run times of our algorithm with those of PRP is not fair, as the algorithms find solutions to different problems. However, general run-times suggest that the branch-and-bound technique used in Algorithm 2 is, in principle, efficient in the search for normative solutions. As the implementation of Algorithm 2 is preliminary and not optimal, we believe there is room for further improvements in its performance.

6 Summary and Future Work

There exist a diversity of models for planning in the face of non-deterministic action outcomes including probabilistic planning, MDPs, and fault-tolerant planning. We focus our attention on FOND planning. Solutions to FOND planning problems are required to be robust to non-deterministic action effects, and to guarantee goal achievement during execution. The class of *strong* solutions provides such guarantees, but the existence of such solutions is restricted in general. On the other hand, the soundness of strong-cyclic solutions is predicated on fairness, and therefore its execution does not guarantee goal achievement, in general, when the fairness assumption is not valid.

To mitigate for this, we introduced the model for *Fully Observable Non-Deterministic* Planning with *Labeled Unfairness* (FOND$^+$), where fairness of action transitions can be described explicitly by means of the labeling function L, and solutions guarantee goal achievement provided that executions are L-fair. We leave complexity analysis of our model to future work.

We distinguished four classes of solutions to FOND$^+$ planning problems. Namely, *strictly fair*, *strictly unfair*, *mixed*, and *normative*. Strictly fair (resp. strictly unfair) solutions are strong-cyclic (resp. strong) solutions to FOND relaxations of the problem. This results in a continuum between strong planning and strong-cyclic planning. Additionally, we found a connection between *normative* solutions and fault-tolerant planning. In particular, when primary action transitions have $L(s, a, s') = \texttt{F}$, normative solutions are solutions to 1-primary normative models of fault-tolerant planning. Notably, our model does not impose a bound on the number of faults during policy execution, and solutions are robust to any finite number of faults during execution. This overcomes the limitation of fault-tolerant planning, where that the number of faults κ that a plan can tolerate must be designated prior to policy generation and execution. We leave further exploration of the connection between the FOND$^+$ model and fault-tolerant planning to future work.

Finally, we presented algorithms to find different classes of solutions to FOND$^+$ planning problems, and an algorithm to find general solutions to FOND$^+$. Our algorithms aim to provide a baseline for future enhancements, although the performance of Algorithm 1 suggests that a branch-and-bound search scheme that explores deterministic plans in a certain order is effective. We leave the design and evaluation of better algorithms to future work.

References

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paulo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147:35–84, 2003.

[Delgrande and Levesque, 2013] James P Delgrande and Hector J Levesque. A formal account of nondeterministic and failed actions. In *Proc. of the 23rd Int'l Joint Conference on Artificial Intelligence (IJCAI)*. Citeseer, 2013.

[D'Ippolito *et al.*, 2011] Nicolás D'Ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesis of live behaviour models for fallible domains. In *33rd Int'l Conf. on Software Engineering (ICSE)*, pages 211–220. IEEE, 2011.

[Domshlak, 2013] Carmel Domshlak. Fault tolerant planning: Complexity and compilation. In *Proc. of the 23th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

[Jaramillo *et al.*, 2014] Andres Calderon Jaramillo, Jicheng Fu, Vincent Ng, Farokh B Bastani, and I-Ling Yen. Fast strong planning for fond problems with multi-root directed acyclic graphs. *International Journal on Artificial Intelligence Tools*, 23(06):1460028, 2014.

[Jensen *et al.*, 2004] Rune M Jensen, Manuela M Veloso, and Randal E Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *Proc. of the 14th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, pages 335–344, 2004.

[Kolobov *et al.*, 2009] Andrey Kolobov, Mausam, and Daniel S Weld. Retrase: Integrating paradigms for approximate probabilistic planning. In *Proc. of the 21st Int'l Joint Conference on Artificial Intelligence (IJCAI)*, pages 1746–1753, 2009.

[Kupferman and Vardi, 1996] Orna Kupferman and Moshe Y Vardi. Verification of fair transition systems. In *Computer Aided Verification*, pages 372–382. Springer, 1996.

[Lussier *et al.*, 2007] Benjamin Lussier, Matthieu Gallien, Jérémie Guiochet, Félix Ingrand, Marc-Olivier Killijian, and David Powell. Fault tolerant planning for critical robots. In *7th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN)*, pages 144–153. IEEE, 2007.

[Muise *et al.*, 2012] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved Non-deterministic Planning by Exploiting State Relevance. In *Proc. of the 22th Int'l Conference on Automated Planning and Scheduling (ICAPS)*, pages 172–180, 2012.

[Pineda *et al.*, 2013] Luis Enrique Pineda, Yi Lu, Shlomo Zilberstein, and Claudia V Goldman. Fault-tolerant planning under uncertainty. In *Proc. of the 23rd Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Puterman, 1994] Martin Puterman. *Markov Decision Processes: Discrete Dynamic Programming*. Wiley, New York, 1994.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.

[Sardina and D'Ippolito, 2015] Sebastian Sardina and Nicolas D'Ippolito. Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In *Proc. of the 24th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, pages 3200–3206. AAAI Press, 2015.

[Trevizan *et al.*, 2007] Felipe W Trevizan, Fabio Gagliardi Cozman, and Leliane Nunes de Barros. Planning under risk and knightian uncertainty. *Proc. of the 20th Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 2007:2023–2028, 2007.

[Waldinger, 1977] Richard Waldinger. Achieving several goals simultaneously. In *Machine Intelligence 8*, pages 94–136. Ellis Horwood, Edinburgh, Scotland, 1977.