Learning Generalized Policies for Fully Observable Non-Deterministic Planning Domains

Till Hofmann and **Hector Geffner**
RWTH Aachen University
till.hofmann@ml.rwth-aachen.de, hector.geffner@ml.rwth-aachen.de

Abstract

General policies represent reactive strategies for solving large families of planning problems like the infinite collection of solvable instances from a given domain. Methods for learning such policies from a collection of small training instances have been developed successfully for classical domains. In this work, we extend the formulations and the resulting combinatorial methods for learning general policies over fully observable, non-deterministic (FOND) domains. We also evaluate the resulting approach experimentally over a number of benchmark domains in FOND planning, present the general policies that result in some of these domains, and prove their correctness. The method for learning general policies for FOND planning can actually be seen as an alternative FOND planning method that searches for solutions, not in the given state space but in an abstract space defined by features that must be learned as well.

1 Introduction

General policies express reactive strategies for solving large families of planning problems such as all Blocks world problems [Srivastava *et al.*, 2008; Hu and De Giacomo, 2011; Belle and Levesque, 2016; Bonet and Geffner, 2018; Illanes and McIlraith, 2019; Celorrio *et al.*, 2019]. Methods for learning such policies have been developed successfully for classical domains appealing to either combinatorial or deep learning approaches [Rivlin *et al.*, 2020; Bonet *et al.*, 2019; Ståhlberg *et al.*, 2022a]. While the learning methods do not guarantee that the resulting general policies are correct and will solve all the problems in the target class, the policies obtained from combinatorial methods are more transparent and can be analyzed and shown to be correct on an individual basis [Francès *et al.*, 2021; Drexler *et al.*, 2022b].

Methods for learning general policies for Markov Decision Problems (MDPs) have also been developed [Toyer *et al.*, 2020; Bajpai *et al.*, 2018; Groshev *et al.*, 2018; Chevalier-Boisvert *et al.*, 2019], in most cases relying on deep learning and deep reinforcement learning (DRL) techniques [Goodfellow *et al.*, 2016; Sutton and Barto, 1998; François-Lavet *et al.*, 2018], but the performance of the learned policies is evaluated experimentally as their correctness cannot be assessed.

The goal of this work is to extend the combinatorial approaches developed for learning general policies for classical domains to non-deterministic, fully observable (FOND) domains [Cimatti *et al.*, 2003]. The motivations are twofold. On the one hand, FOND planning is closely related to both classical and MDP planning. Indeed, the FOND planners that scale up best are those relying on classical planners [Muise *et al.*, 2012; Yoon *et al.*, 2007; Muise *et al.*, 2024], and the policies that reach the goal states of an MDP with probability 1 are precisely the policies that solve the FOND problem underlying the MDP; i.e., where the possible transitions are the ones that have positive probabilities [Geffner and Bonet, 2013; Ghallab *et al.*, 2016]. This means that FOND models capture the qualitative structure of Goal MDPs, and that general policies that solve classes of FOND problems will also solve correctly a larger class of Goal MDPs.

On the other hand, while the best FOND planners rely on classical planners, FOND planning is harder, requiring not just exponential time but exponential space.[1] So the formal relation between the two planning tasks is not so clear. Interestingly, this relation becomes clearer in the generalized setting, where, as we will see, generalized FOND planning reduces to generalized *classical* planning plus FOND dead-end detection. In other words, a general policy for a class $\mathcal{Q}$ of FOND problems can be obtained from a general policy for a class $\mathcal{Q}_D$ of classical problems obtained by the outcome relaxation from those in $\mathcal{Q}$ [Yoon *et al.*, 2007; Muise *et al.*, 2012], along with a description of the dead-end states to be avoided. The resulting method for learning general policies for FOND planning can also be seen as an alternative FOND planning method that solves a FOND problem by solving a number of classical problems, not in the given state space but in an abstract space defined by features that must be learned as well.

The rest of the paper is organized as follows. We review related work and background first, and then introduce general FOND policies and a method for learning them, followed by an evaluation and analysis of the results.

[1] Classical planning is PSPACE-hard [Bylander, 1994], while FOND planning is EXP-hard [Littman *et al.*, 1998; Rintanen, 2004].

2 Related Work

General Policies for Classical Domains. The problem of learning general policies for classical domains has a long history [Khardon, 1999; Martín and Geffner, 2004; Fern *et al.*, 2006], and general policies have been formulated in terms of logic [Srivastava *et al.*, 2011a; Illanes and McIlraith, 2019], and more recently in terms of features and rules [Bonet and Geffner, 2018; Bonet *et al.*, 2019] that can be learned using combinatorial methods [Francès *et al.*, 2021]. The rule language has also been used to express problem decompositions or sketches [Drexler *et al.*, 2021; Drexler *et al.*, 2022b], and in this work it will be used to express general policies for FOND problems.

General Policies for MDPs. Deep learning (DL) and deep reinforcement learning (DRL) methods have also been used to learn general policies for classical domains [Groshev *et al.*, 2018; Chevalier-Boisvert *et al.*, 2019; Rivlin *et al.*, 2020; Ståhlberg *et al.*, 2022b; Ståhlberg *et al.*, 2023] and MDPs [Boutilier *et al.*, 2001; Wang *et al.*, 2008; van Otterlo, 2012; Toyer *et al.*, 2020; Bajpai *et al.*, 2018; Rivlin *et al.*, 2020; Sanner and Boutilier, 2009]. DL and DRL methods scale up better than combinatorial methods and do not need to assume an existing pool of features, but the resulting policies are not transparent and cannot be understood or shown to be correct.

FOND Planning. FOND planning has become increasingly important as a way of solving other types of problems, including MDPs [Teichteil-Königsbuch *et al.*, 2010; Camacho *et al.*, 2016], problems with extended temporal goals [Patrizi *et al.*, 2013; Camacho *et al.*, 2019; Bonassi *et al.*, 2023] and generalized planning problems [Srivastava *et al.*, 2011b; Bonet *et al.*, 2017]. FOND planners rely on different techniques like OBDDs [Cimatti *et al.*, 2003; Kissmann and Edelkamp, 2009], SAT [Geffner and Geffner, 2018], graph search [Mattmüller *et al.*, 2010; Ramírez and Sardina, 2014; Pereira *et al.*, 2022], and classical planning algorithms [Kuter *et al.*, 2008; Fu *et al.*, 2011; Muise *et al.*, 2012; Muise *et al.*, 2024], but problems are solved individually from scratch.

Dead-Ends. Dead-ends in planning refer to states from which there is no solution. There has been work in learning to identify dead-ends in classical planning [Lipovetzky *et al.*, 2016; Steinmetz and Hoffmann, 2017], and in FOND and MDP planning [Kolobov *et al.*, 2010; Camacho *et al.*, 2016]. Closer to this work is the learning of *general* dead-end representations [Ståhlberg *et al.*, 2021]. While dead-ends in the all-outcome relaxation of FOND problems [Yoon *et al.*, 2007] are dead-ends of the FOND problem, the reverse is not true.

3 Background

We review classical, generalized, and FOND planning.

3.1 Classical Planning

A classical planning problem is a pair $P = \langle D, I \rangle$, where D is a first-order *domain* and I contains information about a domain *instance* [Geffner and Bonet, 2013; Ghallab *et al.*, 2016; Haslum *et al.*, 2019]. The domain D is a set of action schemas involving a number of domain predicates. The action schemas have preconditions and positive effects expressed by atoms $p(x_1, \ldots, x_k)$ and the negative (delete) effects are negations of such atoms, where p is a predicate symbol of arity k, and each term x_i is a schema argument.

The instance information is a tuple $I = \langle O, s_0, G \rangle$ where O is a set of objects (constants); s_0 is the initial state, and G is the goal. The ground atoms $p(o_1, \ldots, o_k)$ in the problem instance $P = \langle D, I \rangle$ are the atoms $p(x_1, \ldots, x_k)$ that result from replacing the terms x_i by objects $o_i \in O$, and the ground actions result from grounding the action schemas in a similar way. The states s are sets of ground atoms; those which are true in the state. The initial state s_0 is a set of ground atoms, while G is a set of ground goal atoms.

A classical planning problem $P = \langle D, I \rangle$ defines a state model $M = \langle S, s_0, S_G, Act, A, f \rangle$ where S is the set of states, $s_0 \in S$ is the initial state, $S_G \subseteq S$ is the set of goal states, Act is a set of (ground) actions, $A(s) \subseteq Act$ is the set of actions applicable in the state s, and $f(a, s)$ for $a \in A(s)$ is a deterministic state transition function. In the model $M(P)$ determined by P, the states $s \in S$ are collection of ground atoms from P, s_0 is given, S_G contains the states that include G, Act is the set of ground actions, $a \in A(s)$ if the preconditions of a are true in s, and $s' = f(a, s)$ if $a \in A(s)$ and s' contains the positive effects of a and the atoms in s except those deleted by a.

It is convenient to consider *non-deterministic policies* for classical planning problems instead of (open loop) plans. A policy π for a problem P is a partial function mapping states s of P into *sets* $\pi(s)$ of actions from P, possibly empty. A π-trajectory in P is a sequence of states $s_0, \ldots, s_n$ that starts in the initial state of P such that $s_{i+1} = f(a_i, s_i)$ if $a_i \in A(s_i)$ and $a_i \in \pi(s_i)$. The trajectory is *cyclic* if it contains the same state infinitely often and *maximal* if (1) s_n is the first goal state of the sequence, (2) it is cyclic and does not contain goal states, (3) there is no action a_n in both $\pi(s_n)$ and $A(s_n)$, or (4) $\pi(s_n)$ is undefined. The policy π solves P if the maximal π-trajectories all reach a goal state of P.

3.2 Generalized Classical Planning

Departing slightly from previous work, a *general policy* π for a class $\mathcal{Q}$ of classical instances over the same domain is taken to be a mapping that assigns a (concrete) policy π_P to each problem P in $\mathcal{Q}$. The general policy π *solves* $\mathcal{Q}$ if π_P solves P for each P in $\mathcal{Q}$.

A general policy π can be represented in many forms from formulas or rules to value functions. Following [Bonet and Geffner, 2018; Bonet *et al.*, 2019], we consider general policies π for classes of problems $\mathcal{Q}$ expressed by sets of rules $C \mapsto E$ in terms of a collection Φ of Boolean features p and numerical features n that take value in the non-negative integers. The condition C is a set (conjunction) of Boolean feature conditions and the effect description E is a set (conjunction) of feature value changes. A Boolean feature condition is of the form p, $\neg p$, $n = 0$, and $n > 0$ for Boolean and numerical features p and n in Φ, and feature value changes are of the form p, $\neg p$, $p?$ for Boolean p, and $n\downarrow$, $n\uparrow$, and $n?$ for numerical n.

The general policy π for a class of problems $\mathcal{Q}$ defined by a set R of rules $C \mapsto E$ determines for each problem P in

$\mathcal{Q}$ the policy π_P that maps a reachable state s in P into the set of actions $\pi_P(s)$, where $a \in \pi_P(s)$ iff a is applicable in s, $a \in A(s)$, and the successor state $s' = f(a, s)$ is such that the transition (s, s') *satisfies* a rule in R. The transition (s, s') satisfies a rule $C \mapsto E$ if all feature conditions in C are true in s, and the values of the features change from s to s' according to E; i.e., if p (resp. $\neg p$) is in E, then $p(s') = 1$ (resp. $p(s') = 0$), if $n\downarrow$ (resp. $n\uparrow$) is in E, $n(s) > n(s')$ (resp. $n(s) < n(s')$), if p (resp. n) is not mentioned at all in E, $p(s) = p(s')$ (resp. $n(s) = n(s')$), and if $n = 0$ (resp. $n > 0$) is in E, $n(s') = 0$ (resp. $n(s') > 0$). The transition (s, s') satisfying rule R is also said to be *compatible with rule* R and if (s, s') is compatible with some rule R of policy π, it is called *compatible with the policy* π.

Methods for learning rule-based general policies for classical planning from small training instances have been developed [Bonet *et al.*, 2019; Francès *et al.*, 2021]. For this, a set of rules involving a set of features of minimum complexity is obtained by finding a satisfying assignment to a propositional theory $T(\mathcal{S}, \mathcal{F})$ of minimum cost, where $\mathcal{S}$ is the collection of state transitions appearing in the training instances, and $\mathcal{F}$ is a large pool of features obtained from the domain predicates in a domain-independent manner using a description logic grammar [Baader *et al.*, 2008]. The complexity of feature f in $\mathcal{F}$ is given by the number of grammar rules needed to generate the unary predicate $p(x)$ associated with f. Such unary predicate gives rise to the numerical feature n_p whose value in a state s is given by the number of objects o for which $p(o)$ is true in s, and the Boolean feature b_p that is true in s if n_p is positive in s. Since problems P in the target class $\mathcal{Q}$ often have different goals, it is assumed that the states s in P are extended with a suitable "copy" of the goal atoms; for each goal atom $p(o_1, \ldots, o_k)$, the states s in P are extended with the atom $p_G(o_1, \ldots, o_k)$ where p_G is a new predicate [Martín and Geffner, 2004].

3.3 FOND Planning

A FOND model is a tuple $M = \langle S, s_0, S_G, Act, A, F \rangle$ similar to the one underlying classical planning except that the state transition function F is non-deterministic and maps an action a applicable in a state s into a non-empty set of successor states $s' \in F(a, s)$. The syntax for FOND problems is an extension of the syntax for classical planning where the actions $a \in A$ are *sets* $a = \{b_1, \ldots, b_k\}$ of classical, deterministic actions b_i, all sharing the same preconditions. The application of a results in the random application of one of the actions b_i so that if $a \in A(s)$, $F(a, s) = \{f(b_1, s), \ldots, f(b_k, s)\}$. A (non-deterministic) policy π for a FOND problem P is a partial function that maps states into sets of actions of P. The π-trajectories $s_0, \ldots, s_n$ for FOND problems P are defined in the same way as for classical problems except that for each $a_i \in \pi(s_i)$, the condition $s_{i+1} = f(a_i, s_i)$ is replaced by $s_{i+1} \in F(a_i, s_i)$. In addition, a notion of *fairness* is needed in FOND planning that can be specified by considering π-trajectories that include the actions as $s_0, a_0, s_1, a_1, \ldots, s_n$ where $a_i \in \pi(s_i)$. One such trajectory is deemed *fair* if it is finite, or if it is infinite, and infinite occurrences of states s_i followed by the same action a_i are in turn followed by each of the possible successor states $s_{i+1} \in F(a_i, s_i)$ an infinite number of times. A policy π is a *strong cyclic solution* or simply a *solution* of P if the maximal π-trajectories that are *fair* all reach the goal.

3.4 Dead-Ends and Deterministic Relaxations

A state s is reachable in a classical or FOND problem P if there is a trajectory $s_0, \ldots, s_n$ that reaches s, where $s = s_n$ and $s_{i+1} = f(a_i, s_i)$ or $s_{i+1} \in F(a_i, s_i)$ for $i = 0, \ldots, n-1$ and suitable actions a_i in P. For a reachable state s in P, $P[s]$ defines the problem that is like P but with initial state s. A reachable state s in P is *alive* if $P[s]$ has a solution and a *dead-end* otherwise. Since a general policy π is often aimed at solving all solvable instances $\mathcal{Q}$ in a given domain, it is natural to ask for the class $\mathcal{Q}$ to be closed, in the sense that if P is in $\mathcal{Q}$, then $P[s]$ is in $\mathcal{Q}$ if s is not a dead-end. The set of dead-ends in a FOND problem P is related to the set of dead-end states in the classical problem P_D that results from P when each non-deterministic action $a = \{b_1, \ldots, b_m\}$ is replaced by the set of deterministic actions $b_1, \ldots, b_m$. The classical problem P_D is the so-called deterministic relaxation or all-outcome relaxation [Yoon *et al.*, 2007] and it plays an important role in FOND planners that rely on classical planning algorithms [Muise *et al.*, 2012]. Clearly, if s is a dead-end state in P_D, s will be a dead-end state in the FOND problem P, but the inverse implication is not true.

4 General Policies for FOND Planning

We consider the semantics of general FOND policies and the language to describe them.

4.1 Semantical Considerations

The *semantics* of general policies for classes $\mathcal{Q}$ of FOND problems is clear and direct: a general policy π for $\mathcal{Q}$ must determine a policy π_P for each problem P in $\mathcal{Q}$, and π solves $\mathcal{Q}$ if each problem P in $\mathcal{Q}$ is solved by π_P; i.e., if π_P is a strong cyclic policy for P. The *language* for representing general policies for classes of FOND problems, however, is a bit more subtle than in the case of classical planning. Nonetheless, a tight relation between general policies for FOND problems and general policies for classical problems can be established that will serve to motivate the language for expressing and then learning general FOND policies.

Let $\mathcal{Q}$ be a collection of solvable FOND problems P that is *closed* in the following sense: if P is in $\mathcal{Q}$ and s is an *alive state* reachable in P, then $P[s]$ is also in $\mathcal{Q}$. Let $\mathcal{Q}_D$ stand for the *determinization* of $\mathcal{Q}$; namely, the collection of classical problems P_D obtained from the deterministic (all-outcome) relaxation of the FOND problems P in $\mathcal{Q}$. Let us also say that a general policy π_D for the determinization $\mathcal{Q}_D$ of $\mathcal{Q}$ is *safe* in P_D if for every reachable state s of P_D and every (deterministic) action $b_i \in \pi_D(s)$, there is a (non-deterministic) action a such that $b_i \in a$ and no $s' \in F(a, s)$ is a dead-end. Finally, π_D is *safe* in $\mathcal{Q}_D$ if it is safe in every $P_D \in \mathcal{Q}_D$. We can show the following relation between the general policies that solve the class of FOND problems $\mathcal{Q}$ and the general policies that solve the class of classical problems $\mathcal{Q}_D$:[2]

[2]Proofs can be found in the appendix.

Theorem 1. *Let $\mathcal{Q}$ be a collection of solvable FOND problems P that is closed, and let $\mathcal{Q}_D$ be determinization of $\mathcal{Q}$.*

(1) If π is a general policy that solves the FOND problems $\mathcal{Q}$, a general safe policy π^D can be constructed from π that solves the class of classical problems $\mathcal{Q}_D$.

(2) If π^D is a general safe policy that solves the classical problems $\mathcal{Q}_D$, a general policy π that solves the FOND problems $\mathcal{Q}$ can be constructed from π^D.

This result expresses a basic intuition and the conditions that make the intuition valid; namely, that the uncertainty in the action effects of FOND problems can be "pushed" as uncertainty in the set of possible initial states, resulting in a collection of classical problems, and hence, a generalized classical planning problem. This suggests that one way to get a general policy for a class $\mathcal{Q}$ of FOND problems is by finding a general policy for the classical problems in the determinization $\mathcal{Q}_D$. The theorem qualifies this intuition by requiring that the policy that solves $\mathcal{Q}_D$ must be safe and not visit a dead-end state of P, because a state may be a dead-end in P but not in its determinization P_D. The intuition that FOND planning can leverage classical planning in this way is present in a slightly different form in one of the most powerful FOND planners [Muise *et al.*, 2012]. The correspondence between FOND and classical planning can be captured more explicitly in the generalized planning setting as a FOND problem does not map into a single classical planning problem but into a collection of them.

4.2 Expressing General FOND Policies

The correspondence captured by Theorem 1 implies that general policies π for a class of FOND problems $\mathcal{Q}$ can be obtained from the general policies π' for $\mathcal{Q}_D$ that are safe, i.e., those policies that avoid dead-ends in the "original" FOND problem P.

This observation suggests that a suitable language for defining general FOND policies can be obtained by combining the rule language for describing general policies for classical domains with *constraints* that ensure that the general policies that solve the classical problems $\mathcal{Q}_D$ are safe and do not visit dead-end states of the FOND problem:

Definition 1. *The language for representing a* general policy *over a class $\mathcal{Q}$ of FOND problems is made up of a set R of rules $C \mapsto E$ like for general classical policies, and a set of constraints B, each one being an (implicit) conjunction of Boolean feature conditions like C.*

Both the rules R and the constraints B are defined over a set Φ of Boolean and numerical features that are well defined over the reachable states of the problems $P \in \mathcal{Q}$. The *general FOND policy* defined by a pair of rules R and constraints B is as follows:

Definition 2. *A set of rules R and constraints B define a general FOND policy $\pi = \pi_{R,B}$ over $\mathcal{Q}$ such that in a problem $P \in \mathcal{Q}$, the concrete policy π_P is such that $a \in \pi_P(s)$ iff*

- *there is a state $s' \in F(a, s)$ such that the transition (s, s') satisfies a rule $C \mapsto E$ in R, and*
- *there is* no *state $s' \in F(a, s)$ such that s' satisfies a constraint in B.*

Let us say that a set of constraints B is *sound* relative to a class of FOND problems $\mathcal{Q}$ if every reachable dead-end state s in a problem P in $\mathcal{Q}$ satisfies a constraint in B. Furthermore, a general classical policy π is B-*safe* if for every reachable state s and every (deterministic) action $b_i \in \pi(s)$, there is a (non-deterministic) action a such that $b_i \in a$ and no $s' \in F(a, s)$ satisfies a constraint in B. The basic idea of the method for learning general FOND policies that we will pursue can then be expressed as follows:

Theorem 2. *Let $\mathcal{Q}$ be a class of FOND problems, $\mathcal{Q}_D$ its determinization, and B a sound set of constraints relative to $\mathcal{Q}$. If the rules R encode a general classical policy that solves $\mathcal{Q}_D$ which is B-safe, then the general FOND policy $\pi_{R,B}$ that follows from Definition 2 solves $\mathcal{Q}$.*

5 Learning General FOND Policies

Following Theorem 2, we will learn general policies $\pi_{R,B}$ that solve classes of FOND problems $\mathcal{Q}$ as follows: we sample a subclass of small FOND problems $\mathcal{Q}'$ from $\mathcal{Q}$ and learn rules R and constraints B such that the general policy π_R solves the classical problems in $\mathcal{Q}'_D$ and is B-safe for a sound set of constraints B. With Definition 2, we then obtain a general FOND policy $\pi_{R,B}$ that solves the FOND problems in $\mathcal{Q}'$ (but not necessarily all FOND problems in the target class $\mathcal{Q}$). By looking for the simplest such policies in terms of the cost of the features involved, we will see that general policies that solve $\mathcal{Q}$ can be obtained.

5.1 Min-Cost SAT Formulation

Following [Francès *et al.*, 2021; Bonet *et al.*, 2019], the problem of learning a general policy for a class of classical problems $\mathcal{Q}'_D$ is cast as a combinatorial optimization problem, and more specifically as min-cost SAT problem over a propositional theory $T = T(\mathcal{S}, \mathcal{F})$ where $\mathcal{S}$ is the set of (possible) state transitions (s, s') over the instances P_i in $\mathcal{Q}$ with states S_i, and $\mathcal{F}$ is the pool of features constructed from predicates in the common domain of these instances. The policy rules R are then extracted from the transitions (s, s') that are labeled as "good" in the min-cost satisfying assignment of T by looking at how the selected features change across the transitions. The constraints B will be extracted from T by enforcing a separation between the states that are dead-ends in $\mathcal{Q}'$ from those that are not. The states appearing in $\mathcal{S}$ are prepartitioned into alive, dead-end, and goal states, as explained below in Section 5.2.

The cost of an assignment is given by adding the costs of the features selected from the pool $\mathcal{F}$. Every feature $f \in \mathcal{F}$ has a weight $w(f)$ defined by the number of grammar rules needed to derive the unary predicate $p(x)$ that defines f. The numerical feature n_p expresses the number of grounded $p(o)$ atoms in a state s (i.e., the number of objects that satisfy p in s), while the Boolean feature b_p is true if n_p is positive.

The **propositional variables** in $T(\mathcal{S}, \mathcal{F})$ are the following:

- $Good(s, s')$ is true if the transition (s, s') is good,
- $Select(f)$ is true if the feature f is selected,
- $V(s, d)$ is true if the distance of s to a goal is at most d, where $0 \le d \le |S_i|$ for $s \in S_i$.

The **formulas** in $T(\mathcal{S}, \mathcal{F})$ are in turn:

(1) For every alive state s:

$$\bigvee_{a \in \mathit{Safe}(s)} \bigvee_{s' \in F(a,s)} \mathit{Good}(s, s')$$

where $a \in \mathit{Safe}(s)$ if no $s' \in F(a, s)$ is a dead-end.

(2) For every goal state s: $V(s, 0)$

(3) For every alive state s: Exactly-$1_{d \in \mathbb{N}}$: $\{V(s, d)\}$

(4) For every transition (s, s'):

$$\mathit{Good}(s, s') \wedge V(s, d) \rightarrow \bigwedge_{\substack{a \in A(s): \\ s' \in F(a,s)}} \bigvee_{s'' \in F(a,s)} V(s'', d'') \rightarrow d'' < d$$

(5) For every alive state s and dead state s': $\neg \mathit{Good}(s, s')$

(6) For every goal state s and non-goal state s':

$$\bigvee_{f : [\![f(s)]\!] \neq [\![f(s')]\!]} \mathit{Select}(f)$$

(7) For every alive state s and dead state s':

$$\bigvee_{f : [\![f(s)]\!] \neq [\![f(s')]\!]} \mathit{Select}(f)$$

(8) For all transitions (s_1, s'_1) and (s_2, s'_2):

$$\mathit{Good}(s_1, s'_1) \wedge \neg \mathit{Good}(s_2, s'_2) \rightarrow D(s_1, s_2) \vee D2(s_1, s'_1, s_2, s'_2)$$

where

$$D(s_1, s_2) = \bigvee_{f : [\![f(s_1)]\!] \neq [\![f(s_2)]\!]} \mathit{Select}(f)$$

and

$$D2(s_1, s'_1, s_2, s'_2) = \bigvee_{f : \Delta_f(s_1, s'_1) \neq \Delta_f(s_2, s'_2)} \mathit{Select}(f)$$

The expressions $[\![f(s)]\!]$ and $\Delta_f(s, s')$ stand for the value of feature f in s, and the way in which the value of f changes in the transition from s to s' (up, down, and same value, for both Boolean and numerical features). The formulas express the following. For every alive state, there must be a good transition such that the corresponding FOND action is *safe*, i.e., none of the outcomes lead to a dead-end (1) and such that one good transition leads towards a goal (2, 3, 4). A transition leading to a dead-end may never be good (5). Furthermore, the selected features must be able to distinguish goal from non-goal states (6), alive states from dead-ends (7) and good from non-good transitions (8).

The satisfying assignments of $T(\mathcal{S}, \mathcal{F})$ yield the rules R and the constraints B such that B is sound relative to the sampled class $\mathcal{Q}'$ of FOND problems, and the classical policy π_R given by the rules R constitute a general policy for the classical problems $\mathcal{Q}'_D$ that is B-safe. From Theorem 2, the resulting $\pi_{R,B}$ FOND policy that follows from Definition 2 solves the collection of FOND problems $\mathcal{Q}'$.

Algorithm 1 Dead-End Detection

Input: FOND model $M(P) = \langle S, s_0, S_G, Act, A, F \rangle$
Output: FOND dead-end set $D \subseteq S$

- $D \leftarrow \emptyset$;
- **repeat**
  - **for all** $s \in S \setminus D$ **do**
    - **for all** $a \in A(s)$ **do**
      - **if** $F(a, s) \cap D \neq \emptyset$ **then**
        - Remove a from $A(s)$
  - **for all** $s \in S \setminus D$ **do**
    - **if** $\neg\exists$path $s \xrightarrow{a_1} \ldots \xrightarrow{a_k} s_g.\, a_i \in A(s_i), s_g \in S_G$ **then**
      - Add s to D
- **until** D does not change
- **return** D

Theorem 3. *The theory* $T(\mathcal{S}, \mathcal{F})$ *is* satisfiable *iff there is a general FOND policy* $\pi_{R,B}$ *over the features in the pool* $\mathcal{F}$ *that solves the set of sampled FOND problems* $\mathcal{Q}'$*, such that the selected features distinguish dead, alive, and goal states.*

Since we aim to learn a policy that generalizes beyond the sample instances, the sum of the weights $w(f)$ of selected features f is minimized to penalize overfitting. Given a satisfying assignment $T(\mathcal{S}, \mathcal{F})$, the rules R and the constraints B that define the general FOND policy $\pi_{R,B}$ are extracted as follows. First, the features Φ are obtained from the true $\mathit{Select}(f)$ atoms. Then, for each true atom $\mathit{Good}(s, s')$, a rule $C \mapsto E$ is obtained where C is the Boolean feature valuation true in s (literals p, $\neg p$, $n = 0$, or $n > 0$), and $n\!\uparrow \in E$ if $\Delta_n(s, s') = \uparrow$, $n\!\downarrow \in E$ if $\Delta_n(s, s') = \downarrow$, $p \in E$ if $\Delta_p(s, s') = \uparrow$, and $\neg p \in E$ if $\Delta_p(s, s') = \downarrow$. Duplicate rules are pruned. Finally, the state constraints B are extracted from the Boolean feature evaluations of the dead-end states.

5.2 Dead-End Detection

To identify the sets D of dead-end states in the sampled FOND problems P_i, similar to [Daniele *et al.*, 2000], we iteratively exclude every action a from the set of applicable actions $A(s)$ when a state $s' \in F(a, s)$ is in D, and place s in D when there is no path from s to the goal using the applicable sets $A(s)$ that result. The resulting algorithm, shown in Algorithm 1, is sound and complete:

Theorem 4. *Algorithm 1 is sound and complete, i.e., state* $s \in D$ *iff there is no solution of the FOND problem* $P[s]$.

6 Evaluation

We evaluate the approach on a number of FOND benchmarks, and analyze some of the learned general policies.[3]

6.1 Experimental Results

We modeled and solved the min-cost SAT problem represented by the theory $T(\mathcal{S}, \mathcal{F})$ as an *Answer Set Program* (ASP) [Lifschitz, 2016] in `clingo` [Gebser *et al.*, 2011]. We use the library `pddl` [Favorito *et al.*, 2023] for PDDL parsing

[3]The source code, benchmark domains, and results are available at [Hofmann and Geffner, 2024].

$\mathcal{Q}$	$\lvert P\rvert$	$\lvert S\rvert$	$\lvert T\rvert$	$\lvert O\rvert_T$	$\lvert O\rvert_P$	t_{solve}/s	t_{wall}/s	mem/MB	$\lvert\mathcal{F}\rvert$	$\lvert\Phi\rvert$	$\lvert\mathcal{C}\rvert$	k^*	c_Φ
acrobatics	18	18	3	3	9	<0.1	139	49	23	3	1	4	6
beam-walk	9	9	2	3	9	<0.1	13	41	22	2	0	4	5
blocks3ops	95	95	4	4	20	224	1968	22 237	194	3	0	5	11
blocks-clear	95	95	2	3	20	1	37	185	34	2	0	4	6
blocks-on	190	190	2	3	20	116	158	1966	704	3	0	6	11
doors	19	19	5	7	33	78	1476	2805	625	4	1	10	19
first-responders	99	$\mathbf{15^M}$	2	5	36	2020	13 929	212 496	332	5	2	7	20
islands	300	300	4	32	83	3903	13 871	72 895	1182	4	1	7	13
miner	69	$\mathbf{13^I}$	2	9	184	1 071 294	48 964	199 942	1073	8	4	6	28
spiky-tireworld	170	$\mathbf{36^I}$	3	6	23	5162	9114	73 985	479	8	5	8	36
tireworld	980	$\mathbf{7^C}$	1	3	100	<0.1	382	370	27	5	4	4	12
triangle-tireworld	10	$\mathbf{1^I}$	1	6	231	<0.1	1868	70 973	27	3	1	4	9

Table 1: Evaluation results, where $|P|$ is the total number of problems, $|T|$ is the number of problems used in training, and $|S|$ is the number of solved problems, that includes training and testing. $|O|_T$ is the maximum number of objects in all training instances, $|O|_P$ is the maximum number of objects in all instances, t_{solve} is the solver's CPU time needed for finding the best policy, t_{wall} is the total wall time, mem is the maximum memory consumption, $|\mathcal{F}|$ is the size of the feature pool, $|\Phi|$ is the number of selected features, $|\mathcal{C}|$ is the number of constraints, k^* is the maximum cost of the selected features, and c_Φ is the total cost of all selected features. When the incremental learning approach does not deliver FOND policies that generalize to all problems in the distribution, the reason for the failure is indicated: **I** indicates that the number of facts exceeded the `clingo` limits, **C** indicates that no solution was found with max complexity 15, and **M** indicates that the solver ran out of memory.

and `DLPlan` [Drexler *et al.*, 2022a] for feature generation in the same way as [Drexler *et al.*, 2022b; Francès *et al.*, 2021]. As optimizations, instead of using the ranking $V(s, d)$, we incrementally label all states where all selected transitions lead to the goal as *safe* and require that all *alive* states are also *safe*. Additionally, we do not try to distinguish all dead states from alive states and instead only compare alive states to *critical* states, which are those states that are dead-ends but have an incoming transition from an alive state. Finally, we preprocess the state space S by pruning all dead states that are not critical.

The FOND domains considered were taken from the FOND-SAT distribution [Geffner and Geffner, 2018], leaving out domains with unsupported features. All instances are either randomly generated or taken from the original benchmarks. In *acrobatics*, *beam-walk*, and *doors*, we augmented the existing problem set with smaller instances. The problems in the *blocks* variants are generated by scaling from small problems with only three blocks up to 20 blocks. In *blocks3ops*, the goal is to build a tower of blocks using a three-operator encoding (without a gripper). The domains *blocks-clear* and *blocks-on* use a four-operator encoding (including the gripper) and the goal is to clear a single block and stack a single pair of blocks. In *islands*, we created five variations of each problem from the original problem set. *Miner* and *triangle-tireworld* use the original problem set, while the instances for *spiky-tireworld* and *tireworld* are randomly generated. For all domains, the largest generated instances are of similar size or larger than the largest instances in the original benchmarks.

All experiments were run on Intel Xeon Platinum 8352M CPUs with 32 threads, a memory limit of 220 GB, and a maximal feature complexity $c_{\text{max}} = 15$. The results are shown in Table 1. The suite of problems P in each domain is ordered by size, with the smallest problems used for training and the largest problems for testing. More precisely, starting with a singleton training set consisting of the smallest instance of P, the solver learns a new policy and iteratively tests whether the policy solves the next problem. If this validation fails, the failed instance is added to the training set and the process repeats. Since the instances in these domains become quite large and the min-cost SAT solver does not scale up to large instances, if the policies learned from the smallest instances do not generalize, the approach fails, as shown by the rows in the table with coverage numbers $|S|$ in bold; namely, 5 of the 12 domains. In 7 of the 12 domains, on the other hand, the learning method delivers general FOND policies, some of which will be shown to be correct in the next section.

6.2 Correctness

For proving the correctness of learned general FOND policies, we adapt a method from [Francès *et al.*, 2021; Seipp *et al.*, 2016] based on *complete* and *descendent* policies:

Definition 3. *A FOND policy π is*

(1) dead-end-free *if no π-trajectory visits a dead-end state,*

(2) complete for an instance P *if for every alive state s, we have $\pi(s) \cap A(s) \neq \emptyset$,*

(3) descending over P *if there is some function γ that maps states of P to a totally ordered set $\mathcal{U}$ such that for every alive state s and action $a \in \pi(s) \cap A(s)$, we have $\gamma(s') < \gamma(s)$ for some $s' \in F(a, s)$.*

Typically, one can show that a FOND policy π is descending by providing a fixed tuple $\langle f_1, \ldots, f_n \rangle$ of state features. If for every π-compatible transition (s, s'), we have $\langle f_1(s'), \ldots, f_n(s') \rangle < \langle f_1(s), \ldots, f_n(s) \rangle$ with lexicographic order $<$, then π is descending. It can be shown that such a policy indeed solves P:

Theorem 5. *If π is a policy that is dead-end-free, complete and descending for an instance P, then π solves P.*

Acrobatics

An acrobat needs to reach the end of a beam consisting of n segments. The only ladder to climb up the beam is at its beginning. The acrobat may walk left or right on the beam and on the ground, climb up or down if there is a ladder, and jump on the beam. When walking on the beam, the acrobat may fall down. The acrobat may skip a segment by jumping over it, but she may fall down and break her leg while doing so. Once the leg is broken, she may no longer move.

The learned policy π_{acro} uses three features: (1) the distance $d \equiv \text{dist}(position, next\text{-}fwd, position_G)$ between the current position and the goal position, (2) a Boolean feature $U \equiv |up|$ which is true if the agent is currently on the beam, (3) a Boolean feature $B \equiv |broken\text{-}leg|$ which is true if the agent's leg is broken. The learned policy $\pi_{\text{acro}} = \pi_{R,B}$ consists of the following rules R:[4]

$$r_1 : \quad \{U, d > 0, \neg B\} \mapsto \{d\downarrow\}$$
$$r_2 : \quad \{\neg B, \neg U\} \mapsto \{U\} \mid \{d\uparrow\}$$

It has a single constraint $B = \{b_1\}$:

$$b_1 : \quad \{B, \neg U\}$$

If the acrobat is currently on the beam (U), she is not at the goal ($d > 0$), and the leg is not broken ($\neg B$), then she should decrease the distance to the goal. Otherwise, if she is not on the beam ($\neg U$) and the leg is not broken ($\neg B$), then she should either climb up the ladder or move away from the goal (and therefore closer to the ladder). For the first rule, she may decide to jump to decrease the distance and thereby break her leg. The state constraint forbids this by requiring that she may not end up in a state where she has a broken leg and is not on the beam.

Proposition 1. *The general policy $\pi_{acro} = \pi_{R,B}$ solves the class $\mathcal{Q}_{acro}$ of solvable FOND* acrobatics *problems.*

Doors

The player needs to move through a sequence of n rooms, which are connected by doors. Whenever the player goes to the next room, the incoming and outgoing doors of the room may open or close non-deterministically. There are separate actions for moving to the next room depending on whether the door is open or closed. For the last door, if the door is closed, the player needs to use a key, which is located in the first room. The player may not move back.

The learned policy π_{doors} uses four features: (1) a Boolean feature $G \equiv |player\text{-}at \sqcap final\text{-}loc|$, which is true if the player is at the final location, (2) a Boolean feature $S \equiv |\neg\exists door\text{-}in.player\text{-}at|$, which is true if the player is at the start location (which does not have any incoming door), (3) a Boolean feature $K \equiv |hold\text{-}key|$ which is true if the player is holding the key, (4) a Boolean feature $F \equiv |open \sqcap (\exists door\text{-}out.player\text{-}at) \sqcap \exists door\text{-}in.final\text{-}loc|$, which is true if the player is in the second-last room and the door to the final room is open.

[4] The notation $C \mapsto E_1 \mid E_2$ abbreviates the two rules $C \mapsto E_1$ and $C \mapsto E_2$ with the same condition C.

The policy $\pi_{\text{doors}} = \pi_{R,B}$ uses the following rules R:

$$r_1 : \quad \{\neg G, S, K, \neg F\} \mapsto \{\neg S\}$$
$$r_2 : \quad \{\neg G, S, K, F\} \mapsto \{G, \neg S, \neg F\}$$
$$r_3 : \quad \{\neg G, S, \neg K\} \mapsto \{K\} \mid \{G, \neg S, \neg F\}$$
$$r_4 : \quad \{\neg G, \neg S, K, \neg F\} \mapsto \{\} \mid \{F\} \mid \{G\}$$
$$r_5 : \quad \{\neg G, \neg S, \neg K, F\} \mapsto \{G, \neg F\}$$

It uses one constraint $B = \{b_1\}$:

$$b_1 : \quad \{\neg G, \neg F, \neg S, \neg K\}$$

The need for feature F may not be immediately obvious, as it is not necessary for a strong-cyclic policy starting in the initial state. However, it is needed to distinguish dead from alive states, as the state where the player is in the second-last room without a key and the last door is open is also alive: the player may just move through the open door without a key. Similarly, if F is false and the player is not holding the key, then the state is dead if the player is not at the start location.

We can show that this policy is a solution for $\mathcal{Q}_{\text{doors}}$:

Proposition 2. *The general policy $\pi_{doors} = \pi_{R,B}$ solves the class $\mathcal{Q}_{doors}$ of solvable FOND* doors *problems.*

Islands

In *Islands*, there are two islands connected by a bridge. The person starts on one island while the goal is on the other island. They may swim across but with the risk to drown, from which they cannot recover. Alternatively, they may cross a bridge, but only if there are no monkeys on the bridge. A monkey can be moved to a drop location.

The learned policy π_{islands} uses three features: (1) a Boolean feature $A \equiv |person\text{-}alive|$; (2) a numerical feature $d_{drop} \equiv \text{dist}(bridge\text{-}drop\text{-}location \sqcap bridge\text{-}road[0], road, person\text{-}at)$, which is the distance to a location that is both drop location and starting point of the bridge; (3) a numerical feature d_g, which is the distance to the goal: $d_g \equiv \text{dist}(person\text{-}at_G, road, person\text{-}at)$.

The policy $\pi_{\text{islands}} = \pi_{R,B}$ consists of two rules $R = \{r_1, r_2\}$:

$$r_1 : \quad \{A, d_{drop} = 0, d_g > 0\} \mapsto \{\} \mid \{d_g\downarrow, d_{drop}\uparrow\}$$
$$r_2 : \quad \{A, d_{drop} > 0, d_g > 0\} \mapsto \{d_{drop}\downarrow\} \mid \{d_g\downarrow\}$$

It uses a single constraint $B = \{b_1\}$:

$$b_1 : \quad \{\neg A, d_{drop} > 0, d_g > 0\}$$

The agent first moves to the bridge (r_2). After it has reached the bridge, it directly crosses it if possible ($\{d_g\downarrow, d_{drop}\uparrow\}$). Otherwise, it selects an action that does not have any effect on the features ($\{\}$). The only action that is compatible with $\{\}$ is moving a monkey. As this demonstrates, it is not necessary to encode the monkeys in the policy explicitly. Finally, the only constraint b_1 requires that the person never dies.

Proposition 3. *The general policy $\pi_{islands} = \pi_{R,B}$ solves the class $\mathcal{Q}_{islands}$ of solvable FOND* islands *problems.*

$\mathcal{Q}$	$\|P\|$	$\|S\|$	$\|T\|$	$\|O\|_T$	$\|O\|_P$	t_{solve}/s	t_{wall}/s	mem/MB	$\|\mathcal{F}\|$	$\|\Phi\|$	$\|\mathcal{C}\|$	k^*	c_Φ
acrobatics	18	18	3	3	9	<0.1	206	49	23	3	1	4	6
beam-walk	9	9	2	3	9	<0.1	16	40	22	2	0	4	5
blocks3ops	95	95	5	4	20	664	4475	77 722	235	3	0	5	11
blocks-clear	95	95	2	3	20	1	37	185	34	2	0	4	6
blocks-on	190	190	2	3	20	87	161	2021	704	3	0	6	11
doors	19	19	3	5	33	1	1230	1003	68	3	2	6	11
first-responders	99	$\mathbf{15^I}$	2	5	36	1354	7577	105 752	332	5	6	7	20
islands	300	300	4	32	83	1889	14 165	73 173	1182	4	4	7	13
miner	69	$\mathbf{13^I}$	2	9	184	2 468 172	130 980	131 741	1073	8	61	6	28
spiky-tireworld	170	$\mathbf{17^I}$	3	6	23	115 973	25 454	81 211	1284	7	36	7	29
tireworld	980	980	4	5	100	8112	21 571	72 160	1804	5	20	8	20
triangle-tireworld	10	10	2	15	231	3905	1489	107 053	264	4	5	7	1

Table 2: Evaluation results for policy learning with transition constraints, using the same notation as in Table 1.

7 Variation: Transition Constraints

The general FOND policies and learning schema presented above is based on *state constraints*, which describe states that must be avoided. Alternatively, we can also formulate general policies based on *transition constraints*. Syntactically, transition constraints are like policy rules and have the form $C \mapsto E$. However, they describe *bad transitions* and hence the policy π defined by a set of rules and transition constraints is such that for any $P \in \mathcal{Q}$, $a \in \pi_P(s)$ if the transition (s', s) for some $s' \in F(a, s)$ satisfies a rule, and no state $s'' \in F(a, s)$ satisfies a transition constraint. Formally:

Definition 4. *The language for representing a* general policy with transition constraints *over a class $\mathcal{Q}$ of FOND problems is made up of a set R of rules $C \mapsto E$ like for general classical policies, and a set of transition constraints T of the same form as rules.*

Definition 5. *A set of rules R and transition constraints T define a transition-constrained general FOND policy π over $\mathcal{Q}$ such that $a \in \pi_P(s)$, where π_P is the concrete policy determined by the general policy π in problem P in $\mathcal{Q}$ if*

- *there is a state $s' \in F(a, s)$ such that the transition (s, s') satisfies a rule $C \mapsto E$ in R, and*
- *there is* no *state $s' \in F(a, s)$ such that the transition (s, s') satisfies a transition constraint $C \mapsto E$ in T.*

We call a transition (s, s') in a problem P *critical* if s is alive and s' is a dead-end. Analogously to state constraints, we say that a set of transition constraints T is *sound* relative to a class of FOND problems $\mathcal{Q}$, if every critical transition (s, s') in a problem P in $\mathcal{Q}$ satisfies a constraint in T, and that a general policy π for a class of classical or FOND problems $\mathcal{Q}$ is *T-safe* if for no instance P in $\mathcal{Q}$, there is a π-trajectory containing a critical transition.

Theorem 6. *Let $\mathcal{Q}$ is a class of FOND problems, let $\mathcal{Q}_D$ be its determinization, and let T be a sound set of transition constraints relative to $\mathcal{Q}$. Then if the rules R encode a general classical policy that solves $\mathcal{Q}_D$ which is T-safe, the rules R and constraints T define a general FOND policy $\pi_{R,T}$ that solves $\mathcal{Q}$.*

The experimental results that follow from the use of transition constraints instead of state constraints for defining and learning general FOND policies are shown in Table 2. We can see that in contrast to the state-based variant, the transition-based variant solves all instances of *tireworld* and *triangle-tireworld*.

8 Conclusion

We have extended the formulation for learning general policies for classical planning domains to fully-observable non-deterministic domains. The new formulation for expressing and learning FOND policies exploits a correspondence between the general policies that solve a family $\mathcal{Q}$ of FOND problems and the general safe policies that solve a family $\mathcal{Q}_D$ of classical problems P_D obtained from the all-outcome relaxation (determinization) of the instances P in $\mathcal{Q}$, where the safe policies are those that avoid the dead-end states of P. A representation of the collection of dead-end states is learned along with the features and rules. The resulting safe policies for the family of classical problems P_D do not just solve the FOND problems in $\mathcal{Q}$ but potentially many other FOND problems as well, like those that result from random perturbations which do not create new dead states. This is because the formulation pushes the uncertainty in the action outcomes into uncertainty in the initial states that are all covered by the general policy that solves $\mathcal{Q}_D$. The experiments over existing FOND benchmarks show that the approach is sufficiently practical, resulting in general FOND policies that can be understood and shown to be correct.

Acknowledgements

This work has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research. It has also received funding from the European Research Council (ERC), Grant agreement No 885107, the Excellence Strategy of the Federal Government and the NRW Länder, Germany, and the Knut and Alice Wallenberg (KAW) Foundation under the WASP program.

References

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, and Daniele Nardi. *The description logic handbook: Theory, implementation and applications*. Cambridge U.P., 2003.

[Baader *et al.*, 2008] Frank Baader, Ian Horrocks, and Ulrike Sattler. *Handbook of Knowledge Representation*, chapter Description Logics, pages 135–180. Elsevier, 2008.

[Bajpai *et al.*, 2018] Aniket Nick Bajpai, Sankalp Garg, and Mausam. Transfer of deep reactive policies for MDP planning. In *NeurIPS*, pages 10965–10975, 2018.

[Belle and Levesque, 2016] Vaishak Belle and Hector J. Levesque. Foundations for generalized planning in unbounded stochastic domains. In *KR*, pages 380–389, 2016.

[Bonassi *et al.*, 2023] Luigi Bonassi, Giuseppe De Giacomo, Marco Favorito, Francesco Fuggitti, Alfonso Emilio Gerevini, and Enrico Scala. FOND planning for pure-past linear temporal logic goals. In *ECAI*, pages 279–286. 2023.

[Bonet and Geffner, 2018] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In *IJCAI*, pages 4667–4673, 2018.

[Bonet *et al.*, 2017] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, and Sasha Rubin. Generalized planning: Non-deterministic abstractions and trajectory constraints. In *IJCAI*, pages 873–879, 2017.

[Bonet *et al.*, 2019] Blai Bonet, Guillem Francès, and Hector Geffner. Learning features and abstract actions for computing generalized plans. In *AAAI*, pages 2703–2710, 2019.

[Boutilier *et al.*, 2001] Craig Boutilier, Ray Reiter, and Bob Price. Symbolic dynamic programming for first-order MDPs. In *IJCAI*, pages 690–700, 2001.

[Bylander, 1994] Tom Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[Camacho *et al.*, 2016] Alberto Camacho, Christian Muise, and Sheila McIlraith. From FOND to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *ICAPS*, pages 65–69, 2016.

[Camacho *et al.*, 2019] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, pages 58–67, 2019.

[Celorrio *et al.*, 2019] Sergio Jiménez Celorrio, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *Knowl. Eng. Rev.*, 34, 2019.

[Chevalier-Boisvert *et al.*, 2019] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A platform to study the sample efficiency of grounded language learning. In *ICLR*, 2019.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003.

[Daniele *et al.*, 2000] Marco Daniele, Paolo Traverso, and Moshe Y. Vardi. Strong cyclic planning revisited. In *Recent Advances in AI Planning*, pages 35–48, Berlin, Heidelberg, 2000. Springer.

[Drexler *et al.*, 2021] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Expressing and exploiting the common subgoal structure of classical planning domains using sketches. In *ICAPS*, pages 258–268, 2021.

[Drexler *et al.*, 2022a] Dominik Drexler, Guillem Francès, and Jendrik Seipp. DLPlan. https://doi.org/10.5281/zenodo.5826139, 2022.

[Drexler *et al.*, 2022b] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *ICAPS*, pages 62–70, 2022.

[Favorito *et al.*, 2023] Marco Favorito, Francesco Fuggitti, and Christian Muise. pddl. https://ai-planning.github.io/pddl/, 2023.

[Fern *et al.*, 2006] Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *JAIR*, 25:75–118, 2006.

[Francès *et al.*, 2021] Guillem Francès, Blai Bonet, and Hector Geffner. Learning general planning policies from small examples without supervision. In *AAAI*, pages 11801–11808, 2021.

[François-Lavet *et al.*, 2018] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Found. Trends. Mach. Learn.*, 2018.

[Fu *et al.*, 2011] Jicheng Fu, Vincent Ng, Farokh B. Bastani, and I-Ling Yen. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *IJCAI*, pages 1949–1954, 2011.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *AI Communications*, 24(2):107–124, June 2011.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Geffner and Geffner, 2018] Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *ICAPS*, pages 88–96, 2018.

[Ghallab *et al.*, 2016] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[Groshev *et al.*, 2018] Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *ICAPS*, 2018.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers, 2019.

[Hofmann and Geffner, 2024] Till Hofmann and Hector Geffner. Generalized FOND planning. Zenodo, May 2024. https://doi.org/10.5281/zenodo.11171181.

[Hu and De Giacomo, 2011] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, pages 918–923, 2011.

[Illanes and McIlraith, 2019] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: arbitrary numbers of objects. In *AAAI*, 2019.

[Khardon, 1999] Roni Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

[Kissmann and Edelkamp, 2009] Peter Kissmann and Stefan Edelkamp. Solving fully-observable non-deterministic planning problems via translation into a general game. *KI 2009: Advances in AI*, pages 1–8, 2009.

[Kolobov *et al.*, 2010] Andrey Kolobov, Mausam, and Daniel S. Weld. Sixthsense: Fast and reliable recognition of dead ends in MDPs. In *AAAI*, pages 1108–1114, 2010.

[Kuter *et al.*, 2008] Ugur Kuter, Dana Nau, Elnatan Reisner, and Robert P. Goldman. Using classical planners to solve nondeterministic planning problems. In *ICAPS*, pages 190–197, 2008.

[Lifschitz, 2016] Vladimir Lifschitz. Answer sets and the language of answer set programming. *AI Magazine*, 37(3):7–12, October 2016.

[Lipovetzky *et al.*, 2016] Nir Lipovetzky, Christian Muise, and Hector Geffner. Traps, invariants, and dead-ends. In *ICAPS*, pages 211–215, 2016.

[Littman *et al.*, 1998] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[Martín and Geffner, 2004] Mario Martín and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19, 2004.

[Mattmüller *et al.*, 2010] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher. Pattern database heuristics for fully observable nondeterministic planning. In *ICAPS*, 2010.

[Muise *et al.*, 2012] Christian Muise, Sheila A. McIlraith, and Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*, 2012.

[Muise *et al.*, 2024] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. PRP rebooted: Advancing the state of the art in FOND planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 20212–20221, March 2024.

[Patrizi *et al.*, 2013] Fabio Patrizi, Nir Lipovetzky, and Hector Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *IJCAI*, 2013.

[Pereira *et al.*, 2022] Ramon Fraga Pereira, André Grahl Pereira, Frederico Messa, and Giuseppe De Giacomo. Iterative depth-first search for FOND planning. In *ICAPS*, pages 90–99, 2022.

[Ramírez and Sardina, 2014] Miquel Ramírez and Sebastian Sardina. Directed fixed-point regression-based planning for non-deterministic domains. In *ICAPS*, 2014.

[Rintanen, 2004] Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS*, pages 345–354, 2004.

[Rivlin *et al.*, 2020] Or Rivlin, Tamir Hazan, and Erez Karpas. Generalized planning with deep reinforcement learning. *arXiv preprint arXiv:2005.02305*, 2020.

[Sanner and Boutilier, 2009] Scott Sanner and Craig Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, 2009.

[Seipp *et al.*, 2016] Jendrik Seipp, Florian Pommerening, Gabriele Röger, and Malte Helmert. Correlation complexity of classical planning domains. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3242–3250, New York, New York, USA, July 2016. AAAI Press.

[Srivastava *et al.*, 2008] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *Proc. AAAI*, pages 991–997, 2008.

[Srivastava *et al.*, 2011a] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):615–647, 2011.

[Srivastava *et al.*, 2011b] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *AAAI*, 2011.

[Ståhlberg *et al.*, 2021] Simon Ståhlberg, Guillem Francès, and Jendrik Seipp. Learning generalized unsolvability heuristics for classical planning. In *IJCAI*, pages 4175–4181, 2021.

[Ståhlberg *et al.*, 2022a] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS*, pages 629–637, 2022.

[Ståhlberg *et al.*, 2022b] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning generalized policies without supervision using GNNs. In *Proc. KR*, pages 474–483, 2022.

[Ståhlberg *et al.*, 2023] Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *KR*, pages 647–657, 2023.

[Steinmetz and Hoffmann, 2017] Marcel Steinmetz and Jörg Hoffmann. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *Artificial Intelligence*, 245:1–37, 2017.

[Sutton and Barto, 1998] Richard Sutton and Andrew Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[Teichteil-Königsbuch *et al.*, 2010] Florent Teichteil-Königsbuch, Ugur Kuter, and Guillaume Infantes. Incremental plan aggregation for generating policies in mdps. In *AAMAS*, pages 1231–1238, 2010.

[Toyer *et al.*, 2020] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68, 2020.

[van Otterlo, 2012] Martijn van Otterlo. Solving relational and first-order logical markov decision processes: A survey. In *Reinforcement Learning*, pages 253–292. Springer, 2012.

[Wang *et al.*, 2008] Chenggang Wang, Saket Joshi, and Roni Khardon. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31:431–472, 2008.

[Yoon *et al.*, 2007] Sungwook Yoon, Alan Fern, and Robert Givan. FF-replan: A baseline for probabilistic planning. In *ICAPS*, pages 352–359, 2007.

A Feature Pool

The feature pool is constructed iteratively based on a description logic grammar [Baader *et al.*, 2003] using `DLPlan` [Drexler *et al.*, 2022a] similar to [Francès *et al.*, 2021].

A.1 Description Logic Concepts and Roles

In description logic, *concepts* represent unary relations and *roles* represent binary relations. Higher-arity domain predicates can be represented by concepts and roles as follows: For each k-ary domain predicate p, we add a primitive concept $p[i]$ for $0 \leq i < k$ that denotes the k-th argument of p. Similarly, we add a primitive role $p[i,j]$ for $0 \leq i, j < k$ that denotes the pair (p_i, p_j) of the ith and jth argument of p.

Here, we define the semantics directly in terms of a planning state s, where the universe Δ^s consists of the set of objects occurring in s and the semantics of primitive concepts and roles is defined as follows. For every k-ary state predicate p and $0 \leq i, j < k$:

- $(p[i])^s = \{c_i \in \Delta^s \mid p(c_0, \ldots, c_i, \ldots, c_{k-1}) \in s\}$,
- $(p[i,j])^s = \{(c_i, c_j) \in \Delta^s \times \Delta^s \mid p(c_0, \ldots, c_i, \ldots, c_j, \ldots, c_{k-1}) \in s\}$.

We continue with the compositional roles and concepts. Let C, D be concepts and R, S roles. We iteratively construct the following compositional concepts:

- the universal concept $\top$ where $\top^s = \Delta^s$,
- the bottom concept $\bot$ where $\bot^s = \emptyset$,
- intersection $C \sqcap D$ where $(C \sqcap D)^s = C^s \cap D^s$,
- union $C \sqcup D$ where $(C \sqcup D)^s = C^s \cup D^s$,
- negation $(\neg C)$ where $(\neg C)^s = \Delta^s \setminus C^s$,
- difference $(C \setminus D)$ where $(C \setminus D)^s = (C^s \setminus D^s)$,
- existential restriction $\exists R.C$ where $(\exists R.C)^s = \{a \mid \exists b : (a,b) \in R^s \wedge b \in C^s\}$,
- universal restriction $\forall R.C$ where $(\forall R.C)^s = \{a \mid \forall b : (a,b) \in R^s \rightarrow b \in C^s\}$,
- constant concept c, one for each domain constant c, where $c^s = \{c\}$.

We iteratively construct the following compositional roles:

- the universal role $\top$ where $\top^s = \Delta^s \times \Delta^s$,
- the role intersection $R \sqcap S$ where $(R \sqcap S)^s = R^s \cap S^s$,
- the role union $R \sqcup S$ where $(R \sqcup S)^s = R^s \cup S^s$,
- the role negation $\neg R$ where $(\neg R)^s = \top^s \setminus R^s$,
- the role inverse R^{-1} where $(R^{-1})^s = \{(b,a) \mid (a,b) \in R^s\}$,
- the role composition $R \circ S$ where $(R \circ S)^s = \{(a,c) \mid (a,b) \in R^s \wedge (b,c) \in S^s\}$,
- the transitive closure R^+ where $(R^+)^s = \bigcup_{n \geq 1} (R^s)^n$,
- the transitive reflexive closure $R^\star$ where $(R^\star)^s = \bigcup_{n \geq 0} (R^s)^n$,
- the role restriction $R|_C$ where $(R|_C)^s = R^s \cap (\Delta^s \times C^s)$,
- the identify $id(C)$ where $(id(C))^s = \{(a,a) \mid a \in C^s\}$.

The iterated composition $(R^s)^n$ is constructed inductively with $(R^s)^0 = \{(a,a) \mid a \in \Delta^s\}$ and $(R^s)^{n+1} = (R^s)^n \circ R^s$.

The *complexity* of a concept or role is the number of rules that are applied during its construction, or, equivalently, the size of its syntax tree. We only consider a finite subset of roles and concepts up to complexity bound $c_{\max}$.

A.2 Features

Let C, D be concepts and R, S, T roles up to complexity $c_{\max}$. We construct the following Boolean features f and define their values f^s as follows:

- empty feature $Empty(C)$ where $(Empty(C))^s = \top$ iff $C^s = \emptyset$,
- concept inclusion $C \sqsubseteq D$ where $(C \sqsubseteq D)^s = \top$ iff $C^s \subseteq D^s$,
- role inclusion $R \sqsubseteq S$ where $(R \sqsubseteq S)^s = \top$ iff $R^s \subseteq S^s$,
- nullary $Nullary(p)$ where $(Nullary(p))^s = \top$ iff p is nullary state predicate and $p \in s$.

Similarly, we construct the following numerical features:

- count $Count(C)$ where $(Count(C))^s = |C^s|$,
- concept distance $\mathrm{dist}(C,R,D)$ where $(\mathrm{dist}(C,R,D))^s$ is the smallest $n \in \mathbb{N}_0$ such that there are objects $o_0, \ldots, o_n$ with $o_0 \in C^s$, $o_n \in D^s$, and $(x_i, x_{i+1}) \in R^s$ for all $0 \leq i < n$. If C^s is empty or no such n exists, then $(\mathrm{dist}(C,R,D))^s = \infty$,
- the sum concept distance $\mathrm{sdist}(C,R,D)$ where $(\mathrm{sdist}(C,R,D))^s = \sum_{x \in C^s} \mathrm{dist}^s(\{x\}, R, D)$,
- role distance $\mathrm{rdist}(R,S,T)$ where $(\mathrm{rdist}(R,S,T))^s$ is the smallest $n \in \mathbb{N}_0$ such that there are objects $a, o_0, \ldots, o_n$ with $(a, o_0) \in R^s$, $(a, o_n \in T^s$, and $(x_i, x_{i+1}) \in R^s$ for all $0 \leq i < n$. If R^s is empty or no such n exists, then $(\mathrm{rdist}(C,R,D))^s = \infty$,
- the sum role distance $\mathrm{srdist}(R,S,T)$ where $(\mathrm{srdist}(R,S,T))^s = \sum_{r \in R^s} \mathrm{rdist}^s(\{r\}, S, T)$.

B Implementation

We implemented the propositional theory $T(\mathcal{S}, \mathcal{F})$ as an *Answer Set Program* (ASP) [Lifschitz, 2016] with `clingo` [Gebser *et al.*, 2011]. We use the library `pddl`[5] for PDDL parsing and `DLPlan` [Drexler *et al.*, 2022a] for feature generation in the same way as [Drexler *et al.*, 2022b; Francès *et al.*, 2021].

The control loop is shown in Algorithm 2, which uses an iterative approach to solve a class of problems $\mathcal{Q}$. Starting with the smallest instance (in terms of number of objects) of $\mathcal{Q}$, it iteratively tests whether the current policy solves the current problem P. If the policy fails on P, then P is added to the training set and a new policy for the complete training set is determined. This approach avoids the need to select good training instances manually, as the solver determines which instances to use for training.

[5] https://github.com/AI-Planning/pddl

Algorithm 2 Incremental solver for learning a policy.

Input: Class $\mathcal{Q} = P_1, \dots, P_k$, max complexity c_{max}
Output: Generalized FOND policy π for $\mathcal{Q}$

```
c_min ← 1; T ← ∅, π ← (∅, ∅, ∅); S ← ∅; U ← ∅
for all P ∈ Q do
    if CheckPolicy(π, P) then
        Add P to S
        continue
    Add P to T
    cost_max ← ∞
    for all c ∈ {c_min, ..., c_max} do
        F ← GenerateFeatures(c)
        π_new ← Solve(T, F, c, cost_max)
        if π_new then
            Add P to S
            π ← π_new; c_min ← c; cost_max ← cost(π_new) − 1
return π
```

In each iteration, the solver increments the maximal complexity of all features in $\mathcal{F}$ from the complexity needed for the last policy up to a maximal complexity c_{max}. After finding a policy π, it continues with features of higher complexity but with a total cost bounded by the cost of the last policy. This way, the solver often finds an expensive first policy with low-complexity features and then iteratively improves this policy with features of higher cost while exploiting the upper bound on the total cost. Without this optimization, the solver would often run out of memory for high-complexity features because the number of possible feature combinations grows too large.

For CHECKPOLICY(π, P), we simulate the policy π on P, i.e., we choose each action according to π and then randomly choose one outcome. Note that a successful run does not guarantee that the policy solves P. For this reason, we repeat each check ten times and assume that the policy is a solution if it succeeds every time.

Each call to SOLVE$(T, \mathcal{F}, c, \text{cost}_{\text{max}})$ instantiates the `clingo` program shown in Listing 1 with the training set T, features $\mathcal{F}$ up to complexity c, and an upper limit of the total cost of cost_{max}, implementing the theory $T(\mathcal{S}, \mathcal{F})$. As an optimization, if it is known that there is no policy with features up to complexity $c-1$, then `min_feature_complexity(c)` enforces the use of at least one feature of complexity c. This helps the solver to quickly prune solution candidates that do not lead to a solution. As further optimizations, instead of using the ranking $V(s, d)$ as introduced in Section 5, we incrementally label all states where all selected transitions lead to the goal as *safe* and require that all *alive* states are also *safe*. Additionally, we do not aim to distinguish all dead states from alive states. Instead, we only compare alive states to *critical* states, which are those states that are dead-ends but have an incoming transition from an alive state. Finally, we pre-process the state space S by pruning all dead states that are not critical.

C Additional Correctness Results

C.1 Blocks

In *blocks3ops*, the player can move a block from the table to a stack or vice versa and he may move a block between two stacks. The learned policy π_{blocks} uses three features: (1) a numerical feature $c_{on} \equiv |\exists on_G^+ . \, clear|$, the number of blocks which should be above another block that is currently clear, (2) a numerical feature $m \equiv |on \setminus on_G|$, the number of misplaced blocks, i.e., a block that is stacked on a block that it should not be stacked on, (3) a numerical feature $on \equiv |on|$, the number of currently stacked blocks.

Note that if $m = 0$, then c_{on} is the number of incorrectly placed blocks: If a block a is correctly placed, i.e., it is stacked on the correct block and all blocks below a are also stacked correctly, then there cannot be any clear block that should be below a. On the other hand, if a is not correctly placed, then there must be some clear block b that should be below it, as otherwise there is block c on b that is misplaced, and so $m > 0$.

The rules of the learned policy $\pi_{\text{blocks}} = \pi_{R,B}$ are the following:

$$r_1: \quad \{c_{on} > 0, m = 0\} \mapsto \{on\uparrow, c_{on}\downarrow\} \mid \{on\downarrow\}$$
$$r_2: \quad \{m > 0, on > 0\} \mapsto \{m\downarrow, on\downarrow\} \mid \{m\downarrow, on\downarrow, c_{on}\uparrow\} \mid \{on\downarrow\} \mid \{on\downarrow, c_{on}\uparrow\}$$

The learned policy does not contain any state constraints as there are no dead-ends in blocks.

Proposition 4. *The general policy $\pi_{blocks} = \pi_{R,D}$ solves the class $\mathcal{Q}_{blocks3ops}$ of solvable FOND* blocks3ops *problems.*

D Proofs

Theorem 1. *Let $\mathcal{Q}$ be a collection of solvable FOND problems P that is closed, and let $\mathcal{Q}_D$ be determinization of $\mathcal{Q}$.*

(1) If π is a general policy that solves the FOND problems $\mathcal{Q}$, a general safe policy π^D can be constructed from π that solves the class of classical problems $\mathcal{Q}_D$.

(2) If π^D is a general safe policy that solves the classical problems $\mathcal{Q}_D$, a general policy π that solves the FOND problems $\mathcal{Q}$ can be constructed from π^D.

Proof. In the following, let π_P be the concrete policy for some $P \in \mathcal{Q}$ obtained from the general policy π. Similarly, let π_P^D be the concrete policy for some $P_D \in \mathcal{Q}_D$ obtained from the general policy π^D.

(1) We construct π^D from π as follows. Let $P \in \mathcal{Q}$ be any instance of $\mathcal{Q}$ and P_D its determinization. Let s be any reachable state in P and $\pi_P(s) = a$ with $a = \{b_1, \dots, b_k\}$. For each i, let $\tau_i = s b_i s_1 \cdots s_{g,i}$ the shortest π_P-trajectory starting in s with first action b_i and ending in some goal state $s_{g,i}$. Then, set $\pi_P^D(s) = b_j$ where τ_j has minimal length among all τ_i.

We need to show that π^D is safe and solves $\mathcal{Q}_D$. First, suppose π^D is not safe and hence for some $P_D \in \mathcal{Q}_D$, there is some s such that $\pi_P^D(s) = b_i$ and for every non-deterministic action a with $b_i \in a$, there is a $s' \in F(a, s)$ such that s' is a dead-end in the FOND problem P. But

Listing 1 The clingo code for selecting features and good transitions on a given set of features and a set of instances. For each instance, the input contains facts for states `state` (partitioned into `alive`, `goal`, and implicit dead states), transitions `trans`, features `feature`, and feature evaluations `eval`. The solver selects features `selected` and good transitions `good_trans` such that there is an outgoing good transition for each alive state and such that it can distinguish good and non-good transitions as well as alive, dead, and goal states with the selected features.

```
{ selected(F) } :- feature(F).
1 { good_trans(I, S1, S2) : trans(I, S1, A, S2), safe_action(I, S1, A) } :- alive(I, S1), not goal(I, S1).
{ good_trans(I, S1, S2) : trans(I, S1, A, S2), alive(I, S2) } :- alive(I, S1), not goal(I, S1).
:- alive(I, S), not good_trans(I, S, _), not goal(I, S).
:- good_trans(I, _, S), not alive(I, S).
bool_eval(I, S, F, 1) :- eval(I, S, F, V), V > 0.
bool_eval(I, S, F, 0) :- eval(I, S, F, V), V = 0.
bool_dist(I1, S1, I2, S2) :- state(I1, S1), state(I2, S2), bool_eval(I1, S1, F, V1), bool_eval(I2, S2, F, V2),
                             selected(F), V1 != V2.
trans_delta(I, S1, S2, F, -1) :- trans(I, S1, _, S2), eval(I, S1, F, V1), eval(I, S2, F, V2), V2 < V1.
trans_delta(I, S1, S2, F, 0)  :- trans(I, S1, _, S2), eval(I, S1, F, V),  eval(I, S2, F, V).
trans_delta(I, S1, S2, F, 1)  :- trans(I, S1, _, S2), eval(I, S1, F, V1), eval(I, S2, F, V2), V2 > V1.
trans_diff(F, I1, S11, S12, I2, S21, S22) :- trans_delta(I1, S11, S12, F, D1), trans_delta(I2, S21, S22, F, D2), D1 != D2.
distinguished(I1, S11, S12, I2, S21, S22) :- trans_diff(F, I1, S11, S12, I2, S21, S22), selected(F).
:- alive(I1, S11), alive(I2, S21), not bool_dist(I1, S11, I2, S21), good_trans(I1, S11, S12),
   trans(I2, S21, _, S22), not good_trans(I2, S21, S22), not distinguished(I1, S11, S12, I2, S21, S22).
:- state(I1, S1), state(I2, S2), not bool_dist(I1, S1, I2, S2), goal(I1, S1), not goal(I2, S2).
safe_state(I, S) :- goal(I, S).
safe_state(I, S1) :- alive(I, S1), safe_state(I, S2) : good_trans(I, S1, S2).
:- alive(I, S), not safe_state(I, S).
safe_action(I, S1, A) :- state(I, S1), trans(I, S1, A, _), alive(I, S2) : trans(I, S1, A, S2).
crit_state(I, S2) :- alive(I, S1), trans(I, S1, _, S2), not alive(I, S2).
:- alive(I1, S1), crit_state(I2, S2), not bool_dist(I1, S1, I2, S2).
#minimize { C,F : selected(F), feature_complexity(F, C) }.
#program limit_feature_cost(c).
:- #sum { C,F : selected(F), feature_complexity(F, C) } > c.
#program min_feature_complexity(c).
:- C < c : selected(F), feature_complexity(F, C).
```

then π_P may also reach s' and so π_P does not solve P and so π does not solve $\mathcal{Q}$, contradicting the assumption.

Now, suppose there is an instance P_D with initial state s_0 not solved by π_P^D. Let τ be a maximal π_P^D-trajectory starting in s_0. Clearly, as π^D is safe, τ cannot end in a dead-end state and so must be infinite. But then, as π_P^D selects the action with minimal distance to a goal, every π_P-trajectory must be infinite, and so π_P does not solve P, contradicting the assumption.

Hence, π^D is safe and solves $\mathcal{Q}_D$.

(2) We construct π from π^D as follows. Let $P \in \mathcal{Q}$ be any instance of $\mathcal{Q}$ and P_D its determinization. Let s be any reachable non-goal state in P and $\pi_P^D(s) = b$. As π_D^P is safe, there must be some non-deterministic action $a \in A(s)$ with $b \in a$ and such that no $s' \in F(a, s)$ is a dead-end. For this action a, set $\pi^P(s) = a$.

We show by contradiction that π solves the FOND problems $\mathcal{Q}$. Suppose there is an instance $P \in \mathcal{Q}$ with initial state s_0 such that π^P does not solve P and so there is a π^P-trajectory $\tau = s_0 \cdots s_n$ ending in some dead-end state s_n. Let s_i be the last alive state of τ. As $\mathcal{Q}$ is closed, $P[s_i] \in \mathcal{Q}$ and so $\pi_P^D(s_i)$ is defined, say $\pi_P^D(s_i) = b$. As π^D is safe, there must be a non-deterministic action a with $b \in a$ such that no $s' \in F(a, s_i)$ is a dead-end. As π_P is constructed from π_P^D, s_{i+1} cannot be a dead-end, contradicting the assumption that s_i is the last alive state of τ. Hence, no π^P-trajectory may end in a dead-end state and so π solves $\mathcal{Q}$. □

Theorem 2. *Let $\mathcal{Q}$ be a class of FOND problems, $\mathcal{Q}_D$ its determinization, and B a sound set of constraints relative to $\mathcal{Q}$. If the rules R encode a general classical policy that solves $\mathcal{Q}_D$ which is B-safe, then the general FOND policy $\pi_{R,B}$ that follows from Definition 2 solves $\mathcal{Q}$.*

Proof. By contraposition. Suppose π^D is a general classical policy encoded by rules R that is B-safe and let B be sound relative to $\mathcal{Q}$. Assume $\pi = \pi_{R,B}$ does not solve $\mathcal{Q}$ and so there is a $P \in \mathcal{Q}$ such that the corresponding concrete policy π_P does not solve P, i.e., there is a maximal π_P-trajectory τ that is fair but does not reach the goal.

Suppose τ is finite and so it ends in a non-goal state s such that $\pi_P(s) = \emptyset$ and so for every action $a \in A(s)$, if there is $s' \in F(a, s)$ satisfying some rule in R, then there must be $s'' \in F(a, s)$ that satisfies some constraint in B. As π_D is B-safe and follows the same rules R, this also means that $\pi_P^D(s) = \emptyset$ and so π_D does not solve P_D.

Now, suppose τ is infinite and so contains a cycle $c = s_0 \cdots s_n s_0$. Wlog, assume c is the largest cycle in τ. First, note that all states in c must be alive: Otherwise, suppose s_j is the last alive state in τ. As π^D is B-safe, B is sound relative to $\mathcal{Q}$, and none of the states $s' \in F(\pi_P(s_j), s_j)$ satisfy a constraint in B, no such s' can be a dead-end. Hence, as $\mathcal{Q}$ and $\mathcal{Q}_D$ is closed, it follows for each s_i of c, as s_i is alive, that $P[s_i] \in \mathcal{Q}_D$. Furthermore, as τ is fair, for every s_i of c and every $a \in \pi_P(s_i)$, all successors $F(a, s_i)$ are in c, as otherwise τ would eventually leave the cycle and thus be finite. Now, note that every π_P^D-trajectory is also a π_P-trajectory and so every π_P^D-trajectory visiting some state of c is infinite. Hence, for each s_i of c, π^D does not solve $P_D[s_i]$

and so π^D does not solve $\mathcal{Q}_D$.

Therefore, in either case, π^D does not solve $\mathcal{Q}_D$. $\square$

Theorem 3. *The theory $T(\mathcal{S}, \mathcal{F})$ is* satisfiable *iff there is a general FOND policy $\pi_{R,B}$ over the features in the pool $\mathcal{F}$ that solves the set of sampled FOND problems $\mathcal{Q}'$, such that the selected features distinguish dead, alive, and goal states.*

Proof. **⇒:** Let σ be a satisfying assignment for $T = T(\mathcal{S}, \mathcal{F})$. We first construct the feature set Φ such that $f \in \Phi$ iff $\sigma \models Select(f)$. The feature set Φ distinguishes dead, alive, and goal states: By formula 7, for each pair of alive state s and dead-end s', there must be a feature f such that $\sigma \models Select(f)$ and $[\![f(s)]\!] \neq [\![f(s')]\!]$. Similarly, for each pair of goal state s and non-goal state s', such a distinguishing feature is selected by formula 6.

For the policy, we use the following construction: Let Φ be a set of features, $\mathcal{D}$ a set of states, and $\mathcal{T}$ a set of transitions in $\mathcal{S}$, then the policy $\pi_{\mathcal{T},\mathcal{D}}$ is the policy given by the rules $\Phi(s) \mapsto E_1 | \cdots | E_m$ and constraints B where

- s is a source state in some transition (s, s') in $\mathcal{T}$;
- $\Phi(s)$ is the set of Boolean conditions given by the evaluation of Φ on s, i.e., $\Phi(s) = \{p \mid p(s) = \top\} \cup \{\neg p \mid p(s) = \bot\} \cup \{n > 0 \mid n(s) > 0\} \cup \{n = 0 \mid n(s) = 0\}$;
- for state s_i with $s_i \models \Phi(s)$ and transition (s_i, s'_i) in $\mathcal{T}$, E_i captures the feature changes for (s_i, s'_i): $E_i = \{p \mid \Delta_p(s_i, s'_i) = \uparrow\} \cup \{\neg p \mid \Delta_p(s_i, s'_i) = \downarrow\} \cup \{n\uparrow \mid \Delta_n(s_i, s'_i) = \uparrow\} \cup \{n\downarrow \mid \Delta_n(s_i, s'_i) = \downarrow\}$;
- the constraints B are the feature evaluations of the states in $\mathcal{D}$, i.e., $B = \{\Phi(s) \mid s \in \mathcal{D}\}$.

The policy $\pi = \pi_{R,B}$ is the policy $\pi_{\mathcal{T},\mathcal{D}}$ where $\mathcal{T} = \{(s, s') \in \mathcal{S} \mid \sigma \models Good(s, s')\}$ and $\mathcal{D}$ is the set of dead-end states in $\mathcal{S}$.

We show that π solves $\mathcal{Q}'$. Let $P \in \mathcal{Q}'$ and π_P the corresponding concrete policy. First, note that for every dead-end state s, there is a constraint $B_i \in B$ with $B_i = \Phi(s)$ and so there is no π_P-trajectory ending in a dead-end state. We now show that for every alive state of P there is a maximal π_P-trajectory reaching a goal. Let s be an alive state of P. By formula 1, there is at least one safe action a with $\sigma \models Good(s, s')$ and $s' \in F(a, s)$. By formula 7, s' cannot satisfy any constraint of B and so $\pi_P(s) \neq \emptyset$. Furthermore, by formula 8, for every $a \in \pi_P(s)$, there is at least one s' such that $\sigma \models Good(s, s')$. By formulas 2, 3, and 4, each *good* transition corresponds to an action where one of the outcomes reduces the distance to the goal. Hence, there must be a finite maximal π_P-trajectory starting in s ending in a goal state.

⇐: Let $\pi = \pi_{R,B}$ a general FOND policy over features Φ that solves $\mathcal{Q}'$ and such that Φ distinguishes dead, alive, and goal states. For each alive state s, let $d_\pi(s)$ be the length of the shortest maximal π-trajectory starting in s and $V_\pi(s) = 1 + \max_{a \in \pi(s)} \min_{s' \in F(a,s)} d_\pi(s')$. Note that $d_\pi(s)$ and so $V_\pi(s)$ is well-defined because π solves $\mathcal{Q}'$ and so each maximal π-trajectory ends in a goal state and hence its length is finite. We construct an assignment σ for the variables in $T = T(\mathcal{S}, \mathcal{F})$ that satisfies T:

- $\sigma \models Select(f)$ iff $f \in \Phi$;
- $\sigma \models Good(s, s')$ iff the transition (s, s') is compatible with π;
- $\sigma \models V(s, d)$ iff $d = V_\pi(s)$

We show that σ satisfies the formulas in T:

(1) For every alive state s, as π solves $\mathcal{Q}'$, there is at least one transition (s, s') compatible with π and so $\sigma \models \bigvee_{a \in Safe(s)} \bigvee_{s' \in F(a,s)} Good(s, s')$.

(2) By definition, the distance of a goal state to a goal is 0, satisfying $\sigma \models V(s, 0)$ for every goal state s.

(3) As $V_\pi(s)$ is well-defined, $\sigma \models$ Exactly-1 $V(s, d)$ for each alive state s.

(4) As π solves $\mathcal{Q}'$, there must be some π-trajectory starting with a transition (s, s') moving towards the goal, i.e., $V_\pi(s') < V_\pi(s)$, and hence satisfying $Good(s, s') \wedge V(s, d) \rightarrow \bigwedge_{a \in A(s): s' \in F(a,s)} \bigvee_{s'' \in F(a,s)} V(s'', d'') \rightarrow d'' < d$.

(5) As π solves $\mathcal{Q}'$, there cannot be any transition (s, s') compatible with π that from an alive state s to a dead-end s' and so $\sigma \models \neg Good(s, s')$.

(6) By assumption, Φ distinguishes goal from non-goal states and so $\sigma \models \bigvee_{f : [\![f(s)]\!] \neq [\![f(s')]\!]} Select(f)$.

(7) By assumption, Φ distinguishes alive from dead-end states and so $\sigma \models \bigvee_{f : [\![f(s)]\!] \neq [\![f(s')]\!]} Select(f)$.

(8) Let (s_1, s'_1) and (s_2, s'_2) be two transitions. Clearly, if (s_1, s'_1) is compatible with a rule $C \mapsto E$ but (s_2, s'_2) is incompatible, then (a) $\Phi(s_1) \neq \Phi(s_2)$, or (b) $\Delta_f(s_1, s'_1) \neq \Delta_f(s_2, s'_2)$ for some $f \in \Phi$. Otherwise, (s_2, s'_2) would be compatible with $C \mapsto E$. Hence, $\sigma \models Good(s_1, s'_1) \wedge \neg Good(s_2, s'_2) \rightarrow D(s_1, s_2) \vee D2(s_1, s'_1, s_2, s'_2)$. $\square$

Theorem 4. *Algorithm 1 is sound and complete, i.e., state $s \in D$ iff there is no solution of the FOND problem $P[s]$.*

Proof.
⇒: By contraposition. Let s_1 be a solvable state. Assume π is a solution for $P[s_1]$ and so there is a maximal π-trajectory $\tau = (s_1, s_2, \ldots, s_k, s_g)$ from s to a goal s_g with transitions $(s_1, a_1, s_2), (s_2, a_2, a_2), \ldots, (s_k, a_k, s_g)$. We show by induction on k that no state of τ (and hence also s_1) is marked as dead-end and none of the a_i is removed from $A(s_i)$.
Base case. Clearly, there is 0-length path from s_g to a goal state, as s_g is a goal state.
Induction step. Assume $s_i, s_{i+1}, \ldots, s_g$ are not marked as dead and for any $j > i$, $a_j \in A(s_j)$. As $a_j \in A(s_j)$ for each $j > i$, there is path satisfying the condition in line 8, and so s_i is not added to D. Furthermore, $F(a_i, s_i) \cap D = \emptyset$ (otherwise τ would not be a π-trajectory) and so a_i is not removed from $A(s_i)$.
⇐: By contraposition. Let s be a state not marked as dead, D the states marked as dead, and M be the set of unsafe actions, i.e., $(s_i, a_i) \in M$ if a_i is removed from $A(s_i)$ by the algorithm in line 5. We construct a policy π that solves $P[s]$ as follows: For each $s_1 \in S \setminus D$, if $s_1 \in S_G$, set $\pi(s_1) = \emptyset$. Otherwise, as s_1 is not in D, there is a path

$p = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots \xrightarrow{a_k} s_g$ from s_1 to a goal state s_g such that $\forall i. (s_i, a_i) \notin M$. Wlog assume that p is the shortest of those paths and thus cycle-free, i.e., no state is repeated in the path. Set $\pi(s_1) = a_1$.

We show that every fair maximal π-trajectory ends in a goal state. Assume there is a π-trajectory $\tau = (s, s_1, \dots, s_D)$ that ends in a non-goal state s_D. By definition of π, $s_D \in D$, as otherwise τ would not end in s_D. As $s \notin D$ and $s_D \in D$, there is a k such that $s_k \notin D$ but $s_{k+1} \in D$. Hence $F(\pi(s_k), s_k) \cap D \neq \emptyset$. But then, $(s_k, \pi(s_k)) \in M$, in contradiction to the construction of π. Now, assume there is a fair infinite π-trajectory τ and so there is a cycle c in τ. Wlog, assume c is the longest cycle in τ. Clearly, there is a state s in c with $\pi(s) = a$ for some a such that there is a path $s \xrightarrow{a} s' \to \dots \to s_g$ ending in a goal state s_g and such such that s' is not in c. As s' is not in c, it can occur in τ only finitely many times. On the other hand, τ visits each state of c infinitely often and so τ cannot be fair, a contradiction. Therefore, every fair maximal π-trajectory ends in a state $s' \in S \setminus D$. ☐

Theorem 5. *If π is a policy that is dead-end-free, complete and descending for an instance P, then π solves P.*

Proof. Let a *descending trajectory* be a (finite or infinite) trajectory $\tau = (s_1, s_2, \dots)$ such that $\gamma(s_{i+1}) < \gamma(s_i)$ for every i. Clearly, as π is descending, for every alive state s, there is a maximal π-trajectory starting in s that is descending. Now, suppose there is an alive state s_1 such that some descending maximal π-trajectory τ starting in s_1 does not end in a goal state. We have two cases:

(1) The trajectory $\tau = (s_1, \dots, s_k)$ is finite and s_k is not a goal state. As π is dead-end-free, it follows that s_k is alive and hence, as π is complete, there must be some $a \in \pi(s_k) \cap A(s_k)$. But then τ is not maximal.

(2) The trajectory $\tau = (s_1, s_2, \dots)$ is infinite. As the state space of P is finite, the set $\{\gamma(s_i) \mid s_i \in \tau\}$ has a minimal element. Hence, there is an i such that $\gamma(s_{i+1}) \not< \gamma(s_i)$. But then τ is not descending.

Hence, for every alive state s, there is a descending maximal trajectory starting in s and each such trajectory ends in a goal state. Therefore, π solves P. ☐

Proposition 1. *The general policy $\pi_{acro} = \pi_{R,B}$ solves the class $\mathcal{Q}_{acro}$ of solvable FOND* acrobatics *problems.*

Proof. Let P be any $\mathcal{Q}_{\text{acro}}$ instance and π_P the concrete policy for P as defined by π_{acro}. First, note that the only critical states are those were the acrobat's leg is broken and she is not on the beam, because in order to break the leg, she needs to fall down with a `jump`. As the state constraint is $\{B, \neg U\}$, there is no π_P-compatible transition ending in such a state and so π_P is dead-end-free.

We now show that π_P is complete for P. In every alive state $s \in \mathrm{S_A}(P)$ of P, we have $B(s) = \bot$. Also, $d(s) > 0$ or $U(s) = \bot$. If $U(s) = \bot$, then the acrobat is either at the beginning, in which case she can climb up the ladder (U), or she may walk back and thereby increase the distance to the end ($d\uparrow$). Either way, there is an action compatible with r_2. Otherwise, if $U(s) = \top$ and the acrobat is not at the goal, then she can continue walking on the beam and decrease the distance to the end ($d\downarrow$) without violating the constraint. Hence, if $U(s)$, then there is a transition compatible with r_1 and so π_P is complete.

Finally, we show that π_P is descending over tuple $\langle 1 - U, -(1-U)d, d \rangle$. First, if the acrobat is on the beam, then $(1-U)$ and $-(1-U)d$ always evaluate to 0 and r_1 is the only applicable rule. The only compatible transition is walking toward the end of the beam, decreasing the distance d. Second, if the acrobat is not on the beam, she can either climb up and decrease the value of $1 - U$ from 1 to 0, or she may move toward the beginning and decrease the value of $-(1-U)d$ while leaving the value of $1 - U$ unchanged. Hence, by Theorem 5, π_P solves P. As every concrete policy π_P solves P, it follows that π_{acro} solves $\mathcal{Q}_{\text{acro}}$. ☐

Proposition 2. *The general policy $\pi_{doors} = \pi_{R,B}$ solves the class $\mathcal{Q}_{doors}$ of solvable FOND* doors *problems.*

Proof. Let P be any $\mathcal{Q}_{\text{doors}}$, π_P the concrete policy for P as defined by π_{doors}, and s an alive state. We first show that π_P is dead-end-free. It is easy to see that the dead states are exactly those where the player is not holding a key, is not at the start location, and is also not in the second-last room with an open final door, i.e., those states that satisfy $\{\neg F, \neg S, \neg K\}$, which is the (only) state constraint of π_{doors} and hence those states will never be visited.

Next, we show that every maximal π_P-compatible trajectory starting in s is finite and ends in a goal state. As the player may not move back and may also not put down the key, any trajectory and hence also every π_P-compatible trajectory may not visit the same state more than once. Hence, any such trajectory is finite. Now, for states with $G(s) = \bot$, notice that for every Boolean combination of S, K, and F except $\{\neg F, \neg S, \neg K\}$ and hence for every possible evaluation of an alive state, the policy contains a rule $C_i \mapsto E_i$ such that C_i is satisfied and there is an E_i-compatible transition. Hence, any maximal trajectory may not end in a state with $G(s) = \bot$. As $G(s) = \top$ iff s is a goal state, every maximal π_P-compatible trajectory starting in an alive state ends in a goal state. Hence, for every instance P, the corresponding concrete policy π_P solves P, and so π_{doors} is a solution for $\mathcal{Q}_{\text{doors}}$. ☐

Proposition 3. *The general policy $\pi_{islands} = \pi_{R,B}$ solves the class $\mathcal{Q}_{islands}$ of solvable FOND* islands *problems.*

Proof. Let P be a solver *islands* instance and π_P the corresponding concrete policy according to π_{islands}. We first show that π_P is dead-end-free. The only dead-ends are those where the person is not alive, caused by swimming across the water. As swimming never directly leads to the goal (because the player needs to continue moving on the other island) and swimming from the bridge is also not possible, any attempt to swim violates the state constraint $\{\neg A, d_{drop} > 0, d_g > 0\}$. Hence, π_P never reaches a dead-end.

We now show that π_P is complete: Clearly, moving closer towards the bridge and hence decreasing d_{drop} is possible as long as $d_{drop} > 0$. Once the person has reached the bridge, it is either free of monkeys, in which case the agent can cross the bridge and thereby decrease d_g while increasing d_{drop}. Otherwise, if there are monkeys on the bridge, they can always move a monkey, because they are at a drop location, until the bridge is eventually free. Note that this assumes that the drop location is actually the same location as the start of the bridge, which is the case for all instances. If this were not the case, then the policy would be incomplete. After crossing the bridge, they can always move towards the goal until eventually reaching it.

Now, we show that π_P descending over tuple $\langle d_g, n_m, d_{drop}\rangle$, where n_m is the number of monkeys on the bridge. (Note that for tuple $\langle d_g, d_{drop}\rangle$ and in fact for any tuple over Φ, the policy is not descending, as r_1 has an effect that does not change any feature values). Clearly, while the person is on the first island away from the bridge, they will decrease distance d_{drop} with every action. Once they have reached the bridge, either d_g or n_m will be decreased, until the goal is reached eventually.

Hence, by Theorem 5, π_P solves P, and so π_{islands} solves $\mathcal{Q}_{\text{islands}}$. □

Theorem 6. *Let $\mathcal{Q}$ is a class of FOND problems, let $\mathcal{Q}_D$ be its determinization, and let T be a sound set of transition constraints relative to $\mathcal{Q}$. Then if the rules R encode a general classical policy that solves $\mathcal{Q}_D$ which is T-safe, the rules R and constraints T define a general FOND policy $\pi_{R,T}$ that solves $\mathcal{Q}$.*

Proof. By contraposition. Let π^D be a general classical policy encoded by rules R. Assume $\pi = \pi_{R,B}$ does not solve $\mathcal{Q}$ and so there is a $P \in \mathcal{Q}$ such that the corresponding concrete policy π_P does not solve P.

As T is sound relative to $\mathcal{Q}$ and no transition compatible with π_P satisfies a constraint in T, π_P does not reach any dead-end state. Now, suppose there is a π-reachable non-goal state s such that $\pi_P(s) = \emptyset$ and so for every action $a \in A(s)$, if there is $s' \in F(a, s)$ satisfying some rule in R, then there must be $s'' \in F(a, s)$ such that (s, s'') satisfies some constraint in T. But as π_D is T-safe and follows the same rules R, this also means that $\pi_P^D(s) = \emptyset$ and so π_D does not solve P_D.

Therefore, if π_P does not solve P, there must be a π_P-reachable state s such that every maximal π_P-trajectory starting in s is infinite. By definition, every trajectory of the classical policy π_P^D is also a trajectory of π_P. Hence, every maximal π_P^D-trajectory starting in s is infinite and so π_P^D does not solve P.

Hence, in either case, π^D does not solve $\mathcal{Q}_D$. □

Proposition 4. *The general policy $\pi_{\textit{blocks}} = \pi_{R,D}$ solves the class $\mathcal{Q}_{\textit{blocks3ops}}$ of solvable FOND* blocks3ops *problems.*

Proof. Let P be any instance of $\mathcal{Q}_{\text{blocks3ops}}$ and π_P the corresponding concrete policy. First, as there are no dead-ends in *blocks3ops*, any *blocsk3ops* policy is dead-end-free.

Next, we show that π_P is complete. First, assume $m > 0$, and so there are is at least one tower. It is easy to see that rule r_2 contains all the combinations how unstacking a block from the tower may affect the features: on is always decreased, while m may be either decreased or remain unchanged and c_{on} may be either increased or remain unchanged. Second, assume $m = 0$. We may already have a tower segment, where the lowest block should be stacked on another block, and hence the segment needs to be unstacked. This will be done by decreasing on while leaving the other features unchanged. Otherwise, there is already a partially correct tower (consisting of 1 block or more) and the action compatible with $\{on\uparrow, c_{on}\downarrow\}$ puts the correct block on the partial tower until the tower is completed.

Finally, the policy π_P is descending over tuple $t = \langle mon, c_{on}, on\rangle$. Rule r_2 always decreases on and decreases m or leaves it unchanged and hence always decreases t. If the condition of rule r_1 is satisfied, then $m = 0$ and so $mon = 0$. If rule r_1 is applied, either c_{on} is decreased (while increasing on), or c_{on} remains unchanged and on is decreased. In both cases, t decreases. By Theorem 5, π_P is a solution for P and so π_{blocks} is a solution for $\mathcal{Q}_{\text{blocks3ops}}$. □