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; Ilannes 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; Stählberg 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. 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 \( Q \) of FOND problems can be obtained from a general policy for a class \( Q_D \) of classical problems obtained by the outcome relaxation from those in \( Q \) [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.

---

1Classical 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 [Harden, 1999; Martin 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 [Francesc 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; Stahlberg et al., 2022b; Stahlberg 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 [Stahlberg 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 \( P = (D, I) \), 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 \( I = (O, s_0, G) \) 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 \( P = (D, I) \) 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 \( P = (D, I) \) defines a state model \( M = (S, s_0, S_G, Act, A, f) \) 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 actions 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 \( \pi \) for a problem \( P \) is a partial function mapping states \( s \) of \( P \) into sets \( \pi(s) \) of actions from \( P \), possibly empty. A \( \pi \)-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 in both \( \pi(s_n) \) and \( A(s_n) \), or (4) \( \pi(s_n) \) is undefined. The policy \( \pi \) solves \( P \) if the maximal \( \pi \)-trajectories all reach a goal state of \( P \).

3.2 Generalized Classical Planning

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

A general policy \( \pi \) 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 \( \pi \) for classes of problems \( Q \) expressed by sets of rules \( C \rightarrow E \) in terms of a collection \( \Phi \) 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 \( \Phi \), and feature value changes are of the form \( p, \neg p, p? \) for Boolean \( p \), and \( n, n? \) and \( n? \) for numerical \( n \).

The general policy \( \pi \) for a class of problems \( Q \) defined by a set \( \hat{R} \) of rules \( C \rightarrow E \) determines for each problem \( P \) in
Q the policy \( \pi_P \) that maps a reachable state \( s \) in \( P \) into the set of actions \( \pi_P(s) \), where \( a \in \pi_P(s) \) if \( a \) 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 \( C \Rightarrow E \) 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 \( n \) (resp. \( \neg n \)) is in \( E \), \( n(s) > n(s') \) (resp. \( n(s) < n(s') \)), if \( p \) (resp. \( \neg p \)) is not mentioned at all in \( E \). The transition \( (s, s') \) satisfying rule \( R \) is also said to be compatible with rule \( R \) and \( (s, s') \) is compatible with some rule \( R \) of policy \( \pi \), if it is called compatible with the policy \( \pi \).

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 \( Q \) 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 \( M = (\mathcal{S}, s_0, Q, A, F) \) 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 result in the 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 \( \pi \) for a FOND problem \( P \) is a partial function that maps states into sets of actions in \( P \). The \( \pi \)-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 \( \pi \)-trajectories that include the actions \( a_0, a_1, \ldots, a_n \), where \( a_k \in \pi(s_k) \). 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 \( \pi \) is a strong cyclic solution or simply a solution of \( P \) if the maximal \( \pi \)-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 \( s_{i+1} = f(a_i, s_i) \) 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 \( \pi \) is often aimed at solving all solvable instances \( Q \) in a given domain, it is natural to ask for the class \( Q \) to be closed, in the sense that if \( P \) is in \( Q \), then \( P[s] \) is in \( Q \) 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 \( Q \) of FOND problems is clear and direct: a general policy \( \pi \) for \( Q \) must determine a policy \( \pi_P \) for each problem \( P \) in \( Q \), and \( \pi \) solves \( Q \) if each problem \( P \) in \( Q \) is solved by \( \pi_P \); i.e., if \( \pi_P \) 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 \( Q \) be a collection of solvable FOND problems \( P \) that is closed in the following sense: if \( P \) is in \( Q \) and \( s \) is an alive state reachable in \( P \), then \( P[s] \) is also in \( Q \). Let \( Q_D \) stand for the determination of \( Q \); namely, the collection of classical problems \( P_D \) obtained from the deterministic (all-outcome) relaxation of the FOND problems \( P \) in \( Q \). Let us also say that a general policy \( \pi_D \) for the determination \( Q_D \) of \( Q \) 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, \( \pi_D \) is safe in \( Q_D \) if it is safe in every \( P_D \in Q_D \). We can show the following relation between the general policies that solve the class of FOND problems \( Q \) and the general policies that solve the class of classical problems \( Q_D \): 3

---

3Proofs can be found in [Hofmann and Geffner, 2024b].
Theorem 1. Let \( Q \) be a collection of solvable FOND problems \( P \) that is closed, and let \( Q_D \) be determination of \( Q \).

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

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

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 \( Q \) of FOND problems is by finding a general policy for the classical problems in the determination \( Q_D \). The theorem qualifies this intuition by requiring that the policy that solves \( Q_D \) must be safe and not visit a dead-end state in \( P \), because a state may be a dead-end in \( P \) but not in its determination \( P_D \). 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 [Mousse 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 \( \pi \) for a class of FOND problems \( Q \) can be obtained from the general policies \( \pi^{D} \) for \( Q_D \) 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 \( Q_D \) 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 \( Q \) of FOND problems is made up of a set \( R \) of rules \( C \Rightarrow E \) 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 \( \Phi \) of Boolean and numerical features that are well defined over the reachable states of the problems \( P \in Q \). 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 \( Q \) such that in a problem \( P \in Q \), the concrete policy \( \pi_{P} \) 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 \( C \Rightarrow E \) 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 \( Q \) if every reachable dead-end state \( s \) in a problem \( P \) in \( Q \) satisfies a constraint in \( B \). Furthermore, a general classical policy \( \pi \) 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 \( Q \) be a class of FOND problems, \( Q_D \) its determination, and \( B \) a sound set of constraints relative to \( Q \). If the rules \( R \) encode a general classical policy that solves \( Q_D \) which is \( B \)-safe, then the general FOND policy \( \pi_{R,B} \) that follows from Definition 2 solves \( Q \).

5 Learning General FOND Policies

Following Theorem 2, we will learn general policies \( \pi_{R,B} \) that solve classes of FOND problems \( Q \) as follows: we sample a subclass of small FOND problems \( Q' \) from \( Q \) and learn rules \( R \) and constraints \( B \) such that the general policy \( \pi_R \) solves the classical problems in \( Q_D \) 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 \( Q' \) (but not necessarily all FOND problems in the target class \( Q \)). By looking for the simplest such policies in terms of the cost of the features involved, we will see that general policies that solve \( Q \) 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 \( Q_D \) is cast as a combinatorial optimization problem, and more specifically as min-cost SAT problem over a propositional theory \( T = T(S,F) \) where \( S \) is the set of (possible) state transitions \( (s, s') \) over the instances \( P_i \) in \( Q \) with states \( S_i \), and \( F \) 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 \( Q' \) from those that are not. The states appearing in \( S \) 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 \( F \). Every feature \( f \in F \) 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 \( T(S,F) \) are the following:

- \( \text{Good}(s, s') \) is true if the transition \( (s, s') \) is good,
- \( \text{Select}(f) \) 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 \( 0 \leq d \leq |S_i| \) for \( s \in S_i \).
The formulas in $T(S, F)$ are in turn:

1. For every alive state $s$:
   $$\bigvee_{a \in Safe(s)} \bigvee_{s' \in F(a, s)} Good(s, s')$$
   where $a \in Safe(s)$ 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 $(s, s')$:
   $$Good(s, s') \land V(s, d) \rightarrow \bigwedge_{a \in A(s)} \bigvee_{s'' \in F(a, s)} V(s'', d') \rightarrow d'' < d$$

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

6. For every goal state $s$ and non-goal state $s'$:
   $$\bigvee_{f : \textbf{[}f(s)\textbf{]}} \textbf{Select}(f)$$

7. For every alive state $s$ and dead state $s'$:
   $$\bigvee_{f : \textbf{[}f(s)\textbf{]}} \textbf{Select}(f)$$

8. For all transitions $(s_1, s'_1)$ and $(s_2, s'_2)$:
   $$Good(s_1, s'_1) \land \neg Good(s_2, s'_2) \rightarrow D(s_1, s_2) \lor D2(s_1, s'_1, s_2, s'_2)$$

where

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

and

$$D2(s_1, s'_1, s_2, s'_2) = \bigvee_{f : \textbf{[}f(s_1)\textbf{]}} \textbf{Select}(f)$$

The expressions $\textbf{[}f(s)\textbf{]}$ 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 $T(S, F)$ yield the rules $R$ and the constraints $B$ such that $B$ is sound relative to the sampled class $Q'$ of FOND problems, and the classical policy $\pi_R$ given by the rules $R$ constitute a general policy for the classical problems $Q'_P$ 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 $Q'$.

### 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 $D \subseteq S$

1. $D \leftarrow \emptyset$
2. repeat
3. for all $s \in S \setminus D$ do
4. for all $a \in A(s)$ do
5. if $F(a, s) \cap D \neq \emptyset$ then
6. Remove $a$ from $A(s)$
7. for all $s \in S \setminus D$ do
8. if $\neg \text{path} s \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_g, a_i \in A(s_i), s_g \in S_G$ then
9. Add $s$ to $D$
10. until $D$ does not change
11. return $D$

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

Theorem 3. The theory $T(S, F)$ is satisfiable if there is a general FOND policy $\pi_{R, B}$ over the features in the pool $F$ that solves the set of sampled FOND problems $Q'$, 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 $T(S, F)$, the rules $R$ and the constraints $B$ that define the general FOND policy $\pi_{R, B}$ are extracted as follows. First, the features $\Phi$ are obtained from the true $Select(f)$ atoms. Then, for each true atom $Good(s, s')$, a rule $C \rightarrow E$ is obtained where $C$ is the Boolean feature valuation true in $s$ (literals $p, \neg p, n = 0, n > 0$), and $n \uparrow \in E$ if $\Delta_p(s, s') = \uparrow$, $n \downarrow \in E$ if $\Delta_n(s, s') = \downarrow$, $p \in E$ if $\Delta_p(s, s') = \downarrow$, 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 $T(S, F)$ 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, 2024a].
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 $V(s, a)$, 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.

| $Q$          | $|P|$ | $|S|$ | $|T|$ | $|O_P|$ | $|O_P|$ | $t_{\text{solve}}/s$ | $t_{\text{wall}}/s$ | mem/MB | $|F|$ | $|\Phi|$ | $|C|$ | $k^*$ | $c_{\Phi}$ |
|--------------|------|------|------|--------|--------|---------------------|---------------------|--------|-------|--------|------|-------|--------|
| 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               | 22237   | 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 | 15   | 2    | 5      | 36     | 2020              | 13929              | 212496  | 332   | 5      | 2    | 7     | 20     |
| islands      | 300  | 300  | 4    | 32     | 83     | 3903              | 13871              | 72895   | 1182  | 4      | 1    | 7     | 13     |
| miner        | 69   | 13   | 2    | 9      | 184    | 1071294           | 48964              | 199942  | 1073  | 8      | 4    | 6     | 28     |
| spiky-tireworld | 170 | 36  | 3    | 6      | 23     | 5162              | 9114               | 73985   | 479   | 8      | 5    | 8     | 36     |
| tireworld    | 980  | 7    | 1    | 3      | 100    | <0.1              | 382               | 370     | 27    | 5      | 4    | 4     | 12     |
| triangle-tireworld | 10 | 1    | 1    | 6      | 231    | <0.1              | 1868              | 70973   | 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. $|O_P|$ is the maximum number of objects in all training instances, $|O_P|$ is the maximum number of objects in all instances, $t_{\text{solve}}$ is the solver’s CPU time needed for finding the best policy, $t_{\text{wall}}$ is the total wall time, mem is the maximum memory consumption, $|F|$ is the size of the feature pool, $|\Phi|$ is the number of selected features, $|C|$ is the number of constraints, $k^*$ is the maximum cost of the selected features, and $c_{\Phi}$ 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.

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 descending policies: Definition 3. A FOND policy $\pi$ is

(1) dead-end-free if no $\pi$-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 $\gamma$ 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 $\pi$ is descending by providing a fixed tuple $(f_1, \ldots, f_n)$ of state features. If for every $\pi$-compatible transition $(s, s')$, we have $(f_1(s'), \ldots, f_n(s')) < (f_1(s), \ldots, f_n(s))$ with lexicographic order $<$, then $\pi$ is descending. It can be shown that such a policy indeed solves $P$:

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

6738
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 \( \pi_\text{acro} \) uses three features: (1) the distance \( d \equiv \text{dist}(\text{position}, \text{next-fwd}, \text{position}_G) \) between the current position and the goal position, (2) a Boolean feature \( U \equiv [\text{up}] \) which is true if the agent is currently on the beam, (3) a Boolean feature \( B \equiv [\text{broken-leg}] \) which is true if the agent’s leg is broken. The learned policy \( \pi_\text{acro} = \pi_{R,B} \) consists of the following rules \( R_4^* \):

\[
\begin{align*}
  r_1 & : \{ U, d > 0, \neg B \} \rightarrow \{ d \downarrow \} \\
  r_2 & : \{ \neg B, \neg U \} \rightarrow \{ U \} | \{ d \uparrow \}
\end{align*}
\]

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

\[
  b_1 : \{ B, \neg U \}
\]

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 \( \pi_\text{acro} = \pi_{R,B} \) solves the class \( Q_\text{acro} \) 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 \( \pi_\text{doors} \) uses four features: (1) a Boolean feature \( G \equiv [\text{player-at} \land \text{final-loc}] \), which is true if the player is at the final location, (2) a Boolean feature \( S \equiv [\neg \exists \text{door-in-player-at}] \), which is true if the player is at the start location (which does not have any incoming door), (3) a Boolean feature \( K \equiv [\text{hold-key}] \) which is true if the player is holding the key, (4) a Boolean feature \( F \equiv [\text{open} \land (\exists \text{door-out-player-at}) \lor \exists \text{door-in-final-loc}] \), which is true if the player is in the second-last room and the door to the final room is open.

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

\[
\begin{align*}
  r_1 & : \{ \neg G, S, K, \neg F \} \rightarrow \{ \neg S \} \\
  r_2 & : \{ \neg G, S, K, F \} \rightarrow \{ G, \neg S, \neg F \} \\
  r_3 & : \{ \neg G, S, K \} \rightarrow \{ K \} | \{ G, \neg S, \neg F \} \\
  r_4 & : \{ \neg G, S, K, F \} \rightarrow \{ \} \lor \{ F \} \lor \{ G \} \\
  r_5 & : \{ \neg G, S, K \} \rightarrow \{ \} \lor \{ G \}
\end{align*}
\]

It uses one constraint \( B = \{ b_1 \} \):

\[
  b_1 : \{ \neg G, \neg F, \neg S, \neg K \}
\]

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 \( Q_\text{doors} \):

**Proposition 2.** The general policy \( \pi_\text{doors} = \pi_{R,B} \) solves the class \( Q_\text{doors} \) 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 \( \pi_\text{islands} \) uses three features: (1) a Boolean feature \( A \equiv [\text{person-alive}] \); (2) a numerical feature \( d_{\text{drop}} \equiv \text{dist}([\text{bridge-drop-location}) \land [\text{bridge-road}[0], \text{road}, \text{person-at}]) \), 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: \( d_g \equiv \text{dist}([\text{person-at}_G, \text{road}, \text{person-at}]) \).

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

\[
\begin{align*}
  r_1 & : \{ A, d_{\text{drop}} > 0, d_g > 0 \} \rightarrow \{ \} \lor \{ d_g \downarrow, d_{\text{drop}} \uparrow \} \\
  r_2 & : \{ A, d_{\text{drop}} > 0, d_g > 0 \} \rightarrow \{ d_{\text{drop}} \uparrow \} \lor \{ d_g \downarrow \}
\end{align*}
\]

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

\[
  b_1 : \{ \neg A, d_{\text{drop}} > 0, d_g > 0 \}
\]

The agent first moves to the bridge \( (r_2) \). After it has reached the bridge, it directly crosses it if possible \( \{ d_{\text{drop}} \uparrow \} \). 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 \( \pi_\text{islands} = \pi_{R,B} \) solves the class \( Q_\text{islands} \) of solvable FOND islands problems.
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 $C \rightarrow E$. However, they describe bad transitions and hence the policy $\pi$ defined by a set of rules and transition constraints is such that for any $P \in Q$, $a \in \pi_P(s)$ 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 $Q$ of FOND problems is made up of a set $R$ of rules $C \rightarrow E$ 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 $\pi$ over $Q$ such that $a \in \pi_P(s)$, where $\pi_P$ is the concrete policy determined by the general policy $\pi$ in problem $P$ in $Q$ if

- there is a state $s' \in F(a, s)$ such that the transition $(s, s')$ satisfies a rule $C \rightarrow E$ in $R$, and
- there is no state $s' \in F(a, s)$ such that the transition $(s, s')$ satisfies a transition constraint $C \rightarrow E$ 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 $Q$, if every critical transition $(s, s')$ in a problem $P$ in $Q$ satisfies a constraint in $T$, and that a general policy $\pi$ for a class of classical or FOND problems $Q$ is $T$-safe if for no instance $P$ in $Q$, there is a $\pi$-trajectory containing a critical transition.

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

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 nondeterministic domains. The new formulation for expressing and learning FOND policies exploits a correspondence between the general policies that solve a family $Q$ of FOND problems and the general safe policies that solve a family $Q_D$ of classical problems $P_D$ obtained from the all-outcome relaxation (determinization) of the instances $P$ in $Q$, 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 $Q$ 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 $Q_D$. 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


