Directed Fixed-Point Regression-Based Planning for Non-Deterministic Domains

Miquel Ramirez and Sebastian Sardina*
School of Computer Science and IT
RMIT University
Melbourne, Australia
{miquel.ramirez,sebastian.sardina}@rmit.edu.au

Abstract

We present a novel approach to fully-observable nondeterministic planning (FOND) that attempts to bridge the gap between symbolic fix-point computation and recent approaches based on forward heuristic search. Concretely, we formalize the relationship between symbolic and dynamic programming nondeterministic planners, and then exploit such connection to propose a novel family of planning algorithms that reason over symbolic policies in a directed manner. By doing so, our proposal reasons over sets of states and executions in a succinct way (as done by symbolic planners) while biasing the reasoning with respect to the initial and goal states of the specific planning problem at hand (as done by heuristic planners). We show empirical results that prove this approach promising in settings where there is an intrinsic tension between plan efficiency and plan “robustness,” a feature to be expected in nondeterministic domains.

Introduction

As classical planning has enjoyed unprecedented progress over the last decade or so, more generalised forms of planning have lately attracted much attention in the community. Indeed one highly active area of work today is that of fully-observable non-deterministic planning (FOND), in which the outcomes of an action are uncertain, but observable after execution (Daniele, Traverso, and Vardi 2000). The work presented in this paper aims at providing the missing link between two mainstream approaches to FOND planning.

When it comes to planning in non-deterministic settings, two state-of-the-art approaches stand out. One approach involves leveraging on the efficiency of latest classical planning techniques, and hence building a FOND planner on top of a classical one. This is the case of recent successful planners like PRP (Muise, McIlraith, and Beck 2012), NDP (Kuter and Nau 2008), and FIP (Fu et al. 2011). Roughly speaking, the idea is to first build a weak plan—a linear plan that achieves the goal under specific action outcomes—using a classical planner, and then iteratively fill its “gaps” by synthesizing more plans, again using classical planners, that handle contingencies not yet accounted for. In that way, a set of weak plans is incrementally put together until all potential outcomes of actions are accounted for and a complete plan solution, so-called a strong-cyclic plan, is obtained. At a conceptual level, these can be somehow linked to conditional planners developed in the early nineties, such as WARPLAN-C (Warren 1976) and PLINTH (Goldman and Boddy 1994), in that a systematic case reasoning on contingencies is performed by repeatedly invoking an underlying linear planner.1 While reliance on (fast) classical planners allows these FOND systems to generate weak plans quickly, there is no guarantee such plans are “robust” w.r.t. contingencies: the most efficient weak plans may not be the most robust ones. This is the case, for example, in problems like Triangle-Tireworld (Little and Thiébault 2007) where, as recognized by (Muise, McIlraith, and Beck 2012) themselves, the “attractive nature of driving straight to the goal” may go against building robust plans.

Another powerful and popular approach to FOND planning is that pursued by so-called “symbolic” planners. The basic idea is that, as the non-determinism contributes to the the exponential growth in the search space (Rintanen 2004), a compact symbolic description of sets of states as well as of non-deterministic transition functions should yield great benefit. Indeed, planners based on ad-hoc fix-point CTL model checking algorithms, like MBP (Cimatti et al. 2003), or reductions into abstract two-player games, like GAMER (Kissmann and Edelkamp 2009), are able to reason over such succinct representations in order to synthesise plans accounting for all potential contingencies. By doing so, they somehow prioritise the “robustness” aspect of plans. Besides the complexity involved in some operations over such representations (Ferrara, Pan, and Vardi 2005), a major drawback of these systems however is their inability to discriminate which transitions are relevant to the initial state. As a result, they could end up wasting a substantial amount of computational resources on areas of the state space that are not relevant to the problem at hand (Fu et al. 2011).

In this paper we present GRENDEL, a FOND planner built around a novel, yet simple, non-deterministic regression operator inspired by the work in (Rintanen 2008). We shall argue that the proposed approach constitutes a middle ground between the two mainstream strategies for FOND planning.

---

1The difference with old conditional planners though is striking, in terms of scalability and generality (unlike old conditional planners, FOND planners are not restricted to tree-like plans).

---

*We acknowledge the support of the Australian Research Council under a Discovery Project (grant DP120100332).
described above. Informally, GRENDDEL can be seen as a directed fix-point based regression planner, where search over a huge set of simple formulas replaces expensive symbolic manipulation over one single succinct formula, providing thus an important step towards closing the gap between symbolic and heuristic planners. In providing our technique, we shall relate the approach to various techniques and ideas in the literature and evaluate GRENDDEL over the benchmarks from past IPC’s and those discussed in (Little and Thiébaux 2007), comparing with PRP. As hypothesised, the technique proposed works best when there is a clear tension between plan efficiency and robustness.

Preliminaries
Fully Observable Non-Deterministic Planning

We mostly follow the characterisation of non-deterministic planning given in (Rintanen 2008), as it provides a more formal framework than others to work on. However, such account is indeed equivalent to the usual “oneof” clauses in PDDL based characterisations (Bonet and Givan 2005). For simplicity, though, we do not consider here conditional effects, which are not actually required in the benchmarks considered to evaluate the approach.

A FOND planning problem is a tuple $P = \langle A, s_0, O, G \rangle$ consisting of a set of Boolean state variables $A$ (atoms), an initial state $s_0$, a goal condition $G$ as a conjunction of literals $l \in \mathcal{L}(A)$, where $\mathcal{L}(A) = \{a, \neg a \mid a \in A\}$, and operator set $O$. A state $s$ is a consistent set (or conjunction) of literals such that $|s| = |A|$—every atom is either true or false. We use $S$ to denote the set of all states of task $P$. A condition $\phi$ is a consistent conjunction of literals with $|\phi| < |A|$. An operator is a pair $o = \langle \text{Pre}_o, \text{Eff}_o \rangle$, where $\text{Pre}_o$ is a condition describing the preconditions of operator $o$, and $\text{Eff}_o = e_1 \land \cdots \land e_n$ the (non-deterministic) effects of $o$, where each $e_i$ is a (deterministic) effect condition and $n \geq 1$. The intended meaning is that one of the $e_i$ events ensue non-deterministically, by Nature’s choice.

An operator $o = \langle \text{Pre}_o, \text{Eff}_o \rangle$ is executable on a state $s$ if $s \models \text{Pre}_o$. The successor states resulting from executing operator $o$ in state $s$, denoted $\text{next}(o, s)$, is defined as $\text{next}(o, s) = [\text{Eff}_o]_s$, where $[\cdot]_s$ is defined as follows: (\tilde{I} denotes the complement of literal $I$)

$$[e_1 \cdots e_n]_s = \bigcup_{i=1}^{n} \{s \setminus \{I \mid e_i \models I\} \cup \{I \mid e_i \models \tilde{I}\}\}.$$

Note that $[\text{Eff}_o]_s$ denotes, in general, a set of states (one per non-deterministic effect); if $o$ is deterministic (i.e., $\text{Eff}_o = e$), then $[\text{Eff}_o]_s$ contains one single successor state $s'$.

An execution is a, possibly infinite, sequence of state-action pairs $\lambda = s_0 o_0 s_1 o_1 \cdots$, where $s_{i+1} \in \text{next}(s_i, o_i)$, for $i \geq 0$. An execution is acyclic iff $s_i \neq s_j$, for all $j < i$. An infinite execution $\lambda$ is fair when for each state-action pair $s o$ that appears an infinite number of times in $\lambda$, the triplet $s o s'$ also appears an infinite number of times, for each $s' \in [\text{Eff}_o]_s$. In other words, fair executions assume that the effects of an operator are exactly those with non-zero probability—every effect eventually ensue.

The semantics of a planning task $P$ are given by the underlying non-deterministic state model $M(\bar{P}) = \langle S, s_0, O \cup \{g, d\}, A', F, s_G, c \rangle$, where:

1. $S = \{s \mid s$ is a state in $\bar{P}\}$.
2. $s_0$ is $P$’s initial state and $s_G = \{s \mid s \in S, s \models G\}$.
3. $D(s)$ denotes all actions doable in state $s$ and is defined as
   (i) $D(s) = \{o \mid o \models O, s \models p\}$, if $\{o \mid o \models O, s \models \neg \text{Pre}_o \neq \emptyset\}$; and
   (ii) $D(s) = \{d\}$, otherwise. Dummy actions $g$ and $d$ are used to portray the absorbing nature of goal and dead-end states, resp.
4. $F : S \times O \cup \{g, d\} \mapsto 2^S$ is $M(\bar{P})$’s transition function and defined as follows: $F(s, g) = F(s, d) = \{s\}$ and $F(s, o) = \text{next}(s, o)$, for all $o \in O$ and $s \in S$.
5. $c(g) = 0, c(d) = \infty$, and $c(o) = 1$ for all $o \in O$ is the $M(\bar{P})$’s action cost model.

A policy (or conditional plan) is a set of pairs $\langle \phi, o \rangle$ mapping conjunctions of $A$-literals $\phi$ onto operators $o \models O$ such that $\phi \models \text{Pre}_o$. The set of operators prescribed by policy $\pi$ on state $s$ is set $\pi(s) = \{o \mid s \models \phi, o \models \pi, s \models \phi\}$. A policy $\pi$ defines a set of possible executions $\Lambda_\pi$, made up of executions $\lambda_\pi = s_0 o_0 s_1 \cdots o_i s_{i+1} \cdots$, where $s_0$ is $M(\bar{P})$’s initial state, $o_i \models \pi(s_i), s_{i+1} \models \text{Pre}_o$, and $s_{i+1} \models F(s_i, o_i)$, for all $i \geq 0$. The set of states relevant to a policy $\pi$ is defined as $S_\pi = \bigcup_{\lambda \in \Lambda_\pi} \{s_i \mid s_i \models \lambda\}$ (we abuse notation and write $s \models \lambda$ to say that execution $\lambda$ mentions state $s$). A policy $\pi$ is closed if $\bigcup_{o \models \pi(s)} \text{next}(o, s) \subseteq S_\pi$ for all states $s_i \in S_\pi$.

The cost of execution $\lambda$ is defined as $C(\lambda) = \sum_{o \models \lambda} c(o)$. The worst-case cost function $V_{\text{max}}^\pi$ and best-case cost function $V_{\text{min}}^\pi$ of a policy $\pi$ are defined (Geffner and Bonet 2013) as:

$$V_{\text{max}}^\pi(s) = \begin{cases} \min_{o \models \pi(s)} c(o) + \max_{s' \in F(s, o)} V_{\text{max}}^\pi(s') & \text{if } s \notin s_G \\ 0 & \text{if } s \in s_G \end{cases}$$

$$V_{\text{min}}^\pi(s) = \min_{o \models \pi(s)} c(o) + \min_{s' \in F(s, o)} V_{\text{min}}^\pi(s').$$

Observe that the worst-case cost $V_{\text{max}}^\pi(s)$ may turn out to be infinite for many policies $\pi$, and that $V_{\text{min}}^\pi(s)$ captures the “best” or “optimistic” accumulated cost attained by executing policy $\pi$.

Finally, a policy $\pi$ is a strong cyclic solution for $M(P)$ iff $\pi$ is closed and all possible executions $\lambda \in \Lambda_\pi$ are either finite and ending in a goal state or are infinite and unfair (Daniele, Traverso, and Vardi 2000). Strong cyclic policies turn out to be those for which $V_{\text{min}}^\pi(s)$ is finite for all $s \in S_\pi$ and capture the idea that the goal will be eventually achieved if the domains behave fairly: all actions’ outcomes will eventually ensue. Strong policies are a special case of strong cyclic policies, for which all executions are finite and acyclic: they solve the planning problem in a bounded number of steps.

Computation Tree Logic and FOND planning
Computation Tree Logic (Clarke and Emerson 1982) (CTL) has been proved to be a convenient framework for describing
the notion of strong cyclic policies in a precise and concise manner (Daniele, Traverso, and Vardi 2000). In this paper, we shall use the following restricted subset of CTL formulas:

\[ \phi ::= a \mid \neg a \mid \text{do}(o) \mid (\phi \land \phi) \mid AX \phi \mid EX \phi \mid AG \phi \mid EF \phi, \]

where \( a \) ranges over the set of state variables \( A \) and \( \text{do}(o) \) is an auxiliary propositional variable denoting that operator \( o \) is executed. Formula \( AX \phi \) states that all next states satisfy \( \phi \), whereas \( EX \phi \) states that there exist one successor state satisfying \( \phi \). Formula \( AG \phi \) denotes that in all executions, formula \( \phi \) holds always, that is, \( \phi \) is true along all executions. Similarly, \( EG \phi \) states that there exists one execution in which \( \phi \) always holds (i.e., holds in every state of the execution). The meaning of such CTL formulas is given over the states and paths of a transition system, with a branching-time interpretation of time.

Importantly there is a direct mapping relating nondeterministic state model \( M(\mathcal{P}) \) and policies \( \pi \), as described above, to the Kripke structures commonly used to define CTL’s semantics. It follows then that:

- **policy \( \pi \) is strong cyclic for a planning problem \( \mathcal{P} \) if and only if**
  \[ K_\pi^M(\mathcal{P}), s_0 \models AG(EG\mathcal{F}) \]

where \( K_\pi^M(\mathcal{P}) \) is the CTL branching-time Kripke structure induced by executing policy \( \pi \) on planning problem \( \mathcal{P} \). In words, starting from the initial state \( s_0 \), whatever actions we choose to execute and whatever their outcomes are, we always \( (AG) \) has a way of reaching the goal \( (EG\mathcal{F}) \). Interestingly, because strong cyclic plans are closed, this property applies to every relevant state of the policy, that is, \( K_\pi^M(\mathcal{P}), S \models AG(EG\mathcal{F}) \) for every \( s \in S_\pi \).

We close by noting that symbolic FOND planners such as MBP (Cimatti et al. 2003) cast the problem of computing a strong cyclic policy as that of checking if the set of strong cyclic policies admitted by \( M(\mathcal{P}) \) is not empty. Such planners have the ability to reason about sets of states and transitions, as well as sets of (partial) policies, in a compact manner, by relying on symbolic representations like OBDDs. This is appealing for FOND planning since it is necessary to reason about multiple potential executions of a plan due to nondeterministic effect of actions. However, such techniques are tailored towards finding closed policies, regardless of the initial state—they are **undirected**.

**Regression Search under Non-determinism**

In this section, we will lay down the abstract framework behind our FOND planning strategy, to be detailed in the next two sections. The idea, informally, is to synthesise a strong cyclic policy by searching through a so-called “causal graph” built by applying a regression operator on conditions \( \phi \). In doing so, moreover, we argue that such framework can also be used to understand the operation of other FOND planners.

The **regression** of a condition \( \phi \) with respect to operator \( o=\langle \text{Pre}_o, e_1 \mid \cdots \mid e_n \rangle \) is defined as follows (Rintanen 2008):

\[ \mathcal{R}(\phi, o) = \text{Pre}_o \land (\mathcal{R}(\phi, e_1) \lor \cdots \lor \mathcal{R}(\phi, e_n)), \]

where \( \mathcal{R}(\phi, e) \) for deterministic effect \( e \) is as follows:

- \( \mathcal{R}(\phi, e) = \text{True} \) if \( a \in A \) and \( e \models a; \)
- \( \mathcal{R}(\phi, e) = \text{False} \) if \( a \in A \) and \( e \models \neg a; \)
- \( \mathcal{R}(\phi, e) = a \) if \( a \in A \), \( e \models \neg a \) and \( e \models \neg a; \)
- \( \mathcal{R}(\phi_1 \land \phi_2, e) = \mathcal{R}(\phi_1, e) \land \mathcal{R}(\phi_2, e); \)
- \( \mathcal{R}(\phi_1 \lor \phi_2, e) = \mathcal{R}(\phi_1, e) \lor \mathcal{R}(\phi_2, e); \)
- \( \mathcal{R}(\neg \phi_1, e) = \neg \mathcal{R}(\phi_1, e). \)

Intuitively, the regression operator implicitly identifies the subset of the state space \( S \) containing those states which are **causally relevant** to an arbitrary formula \( \phi \). Using such relation, we can build a structure encoding all such relations.

**Definition 1.** Let \( \mathcal{P} = \langle A, s_0, O, \mathcal{G} \rangle \) be a FOND problem. The causal graph of \( \mathcal{P} \) is a triple \( CG(\mathcal{P}) = \langle V, L, E \rangle \), where:

- \( V = \{ \phi \mid \phi \) is a conjunction of literals \( l \in L(A) \) \} is the set of vertices;
- \( L = O \cup \{ g, d \} \) is the set of labels;
- The set of edges is defined as \( E = E_1 \cup E_2 \), where:
  - \( E_1 = \{ (\phi, \psi) \mid \phi, \psi \in V, o \in O, \phi \models \mathcal{R}(\psi, o) \}; \)
  - \( E_2 = \{ (\mathcal{G}, \mathcal{G}) \} \cup \{ (\phi, \psi) \mid \phi \in V, \mathcal{G}, o. (\phi, \psi) \in E \}. \)

Intuitively, an edge \( (\phi, \psi) \) means that executing operator \( o \) in a state where \( \phi \) is true may result in a successor state where \( \psi \) is true. Set of edges \( E_1 \) accounts for the dummy actions \( g \) and \( d \) to loop over the goal and dead-ends. Clearly, graph \( CG(\mathcal{P}) \) includes nodes with no relation with the goal—there is no path from them to vertex \( \mathcal{G} \).

**Definition 2.** The causal relevant graph of \( \mathcal{P} \) for goal \( \mathcal{G} \), denoted \( CG(\mathcal{P}, \mathcal{G}) \), is defined as the sub-graph of \( CG(\mathcal{P}) \) obtained by restricting to those vertices from where there is a (directed) path to the (goal) vertex \( \mathcal{G} \).

An example of a causal graph and its relevant fragment is depicted in Figure 1. In a relevant causal graph, every vertex is related to the goal. More concretely, each of the edges \( (\phi, \psi) \) in \( CG(\mathcal{P}, \mathcal{G}) \) corresponds exactly with the CTL statement \( \phi \lor \text{do}(o) \models EF\mathcal{G} \): if \( \phi \) is true and \( o \) is executed, there exist a (potential) execution where the goal \( \mathcal{G} \) is indeed achieved. It follows then that a path from any vertex \( \phi \) to the goal vertex \( \mathcal{G} \) encodes a set of weak solutions from
any state where \( \phi \) holds, as formulas implicitly denote sets of states.

We argue then that, besides our planner (to be described in the next sections), other FOND techniques are also, in some way or another, exploring/building this \( G \)-relevant symbolic graph. For example, the FOND state-of-the-art planner PRP (Muise, McIlraith, and Beck 2012) continuously builds paths in \( CG(\mathcal{P}, \mathcal{G}) \) from a given vertex \( \phi \) (initially one characterizing the initial state \( s_0 \)) to the goal vertex \( \mathcal{G} \) by first issuing calls to a (forward) classical planner over the all-outcomes determination of \( \mathcal{P} \) (Yoon, Fern, and Givan 2007) and then generalizing the weak plan solution to account for more states per step by using a regression operator analogous to \( R(\cdot, \cdot) \).

Though in a very different manner, Cimatti et al. (2003)'s MBP planner can also be seen as operating on the causal relevant graph of a planning problem. Indeed, the models of the formula generated through the pre-image procedure \( \text{WeakPreImage} \) correspond exactly with the paths in graph \( CG(\mathcal{P}, \mathcal{G}) \). Thus, if there is no vertex \( \phi \) in \( CG(\mathcal{P}, \mathcal{G}) \) such that \( s_0 \models \phi \) (i.e., the initial state is not relevant for the goal), then one can conclude that there is no weak policy solving planning task \( \mathcal{P} \). A subtle, but important, difference is that MBP puts all states denoted by the regression step (via \( \text{WeakPreImage} \)) together, whereas \( CG(\mathcal{P}, \mathcal{G}) \) separates the regression w.r.t. both operators and their non-deterministic effects. Technically, let \( \{\phi\} = \{s \mid s \in S, s \models \phi\} \) be the set of states where \( \phi \) holds and consider the predecessors of a node \( \phi \) in \( CG(\mathcal{P}, \mathcal{G}) \), denoted \( \text{Pred}(\phi, o) = \{\phi' \mid \phi', \phi \in o\} \).

**Theorem 1.** For every vertex \( \phi \) in \( CG(\mathcal{P}, \mathcal{G}) \) and \( o \in O \),

\[
\text{WeakPreImage}(\{\phi\}) = \bigcup_{o \in O} (\text{Pred}(\phi, o) \times \{o\}).
\]

**Proof (sketch).** The proof relies on the following equalities stating that each disjunct term of \( R(\phi, o) \) denotes a subset of \( \text{Pred}(\phi, o) \), namely, those states for which the execution of \( o \) and the occurrence of the \( i \)-th effect of \( o \) result in a state satisfying \( \phi \):

\[
\begin{align*}
\text{Pred}(\phi, o) & = [\{s \mid s \models \text{Pre}_o, s' \in \text{next}(s, o), s' \models \phi\}] ; \\
& = \bigcup_{i \in \{1, \ldots, n\}} [s \mid s \models \text{Pre}_o \land R(\phi, e_i)]; \\
& = \{s \mid \langle s, o \rangle \in \text{WeakPreImage}(\{\phi\})\} .
\end{align*}
\]

This “separation” of the weak pre-image shall facilitate directing the fix-point computation for extraction a strong-cyclic solution. Whereas MBP iteratively manipulates, implicitly, all state-action pairs that are one, two, three and so on steps away from the goal, the graph \( CG(\mathcal{P}, \mathcal{G}) \) will be explored un-evenly—not in a uniform manner—by expanding those state-action pairs that are more “promising” w.r.t. the initial state first.

We close by noting also a very interesting relationship between the optimistic \( \min \)-\( \min \) relaxation \( V^\pi_{\min}(s) \) defined above and the value of the policies induced by \( CG(\mathcal{P}, \mathcal{G}) \).

1: \textbf{procedure BACKWARDS POLICY SEARCH}
2: \hspace{1em} \pi_{\phi} \leftarrow \{\langle \mathcal{G}, g \rangle \}; \pi_{\phi} \leftarrow \pi_{\phi}; \pi_{\phi} \leftarrow \emptyset;
3: \textbf{repeat}
4: \hspace{1em} \pi'_l \leftarrow \pi_l; \pi'^*_{\phi} \leftarrow \pi_\phi \quad \triangleright \text{Regression phase}
5: \hspace{1em} \langle \phi, o \rangle \leftarrow \text{GetBest}(\pi_{\phi});
6: \hspace{1em} \text{if } s_0 \models \phi \text{ then return } \pi_{\phi};
7: \hspace{1em} \pi^*_{\phi} \leftarrow \pi^*_{\phi} \backslash \{\langle \phi, o \rangle\};
8: \hspace{1em} \text{for } o' \in O \text{ do}
9: \hspace{2em} \pi_l \leftarrow \pi_l \cup \text{Regress}(\phi, o');
10: \textbf{while } \Psi \text{ do} \quad \triangleright \text{Checking phase}
11: \hspace{1em} \langle \phi, o \rangle \leftarrow \text{GetBest}(\pi_l);
12: \hspace{1em} \{Y, Z\} \leftarrow \text{Check}(\phi, o, \pi_{\phi}, \pi_{\phi});
13: \hspace{1em} \pi_{\phi} \leftarrow \pi_{\phi} \cup Y; \pi_{\phi} \leftarrow \pi_{\phi} \cup \pi_{\phi};
14: \hspace{1em} \pi_l \leftarrow \pi_l \cup Z;
15: \text{until } (\pi_l = \pi_l \text{ and } \pi_{\phi} = \pi_{\phi})

Figure 2: Basic scheme for backwards policy search. Outer loop is finished when neither target or base policies change.

**Theorem 2.** Let \( \mathcal{P} = \{A, s_0, O, \mathcal{G}\} \) be a planning task, and \( \langle \phi, \psi \rangle \) an edge of \( CG(\mathcal{P}, \mathcal{G}) \) = \( \langle \mathcal{V}, \mathcal{L}, \mathcal{E}\rangle \). Then, for every \( \phi \in \mathcal{V} \) and state \( s \in S \) such that \( s \models \phi \):

\[
V^\pi_{\min}(s) = \min_{o \in O} Q_{\min}(\phi, o),
\]

where \( Q_{\min}(\phi, o) \) is defined as:

\[
Q_{\min}(\phi, o) = c(\phi) + \min_{\psi \in \{\psi(\phi, \psi, o) \in e\}} \min_{o' \in O} Q_{\min}(\psi, o').
\]

Intuitively \( Q_{\min}(\phi, o) \), for a vertex \( \phi \) in the relevant graph, is the optimistic cost of all possible policies induced by \( CG(\mathcal{P}, \mathcal{G}) \) prescribing operator \( o \) in any state where \( \phi \) holds.

This result is important in that it formalises the insight in (Geffner and Bonet 2013, pp. 77) hinting at a deep connection between symbolic synthesis (Daniele, Traverso, and Yardi 2000) and dynamic programming (Puterman 1994).

**Policy Search**

The relationship between dynamic programming and reasoning over the symbolic causal graph, as crystallised in Theorem 2, suggests many novel approaches to the problem of searching for strong cyclic policies effectively. We propose one, of many possible, general algorithm in Figure 2, which basically generates graph \( CG(\mathcal{P}, \mathcal{G}) \) in a systematic, incremental, and directed way towards the initial state. The suggested scheme is best understood as a backwards formulation of \( AO^* \) algorithm (Nilsson 1982).

Algorithm **BackwardsPolicySearch** operates on two policies, the target policy \( \pi_{\phi} \) and the base policy \( \pi_{\phi} \), both initialized in line 1 and defined as sets of so-called **policy nodes** pairs \( \langle \phi, o \rangle \) prescribing the execution of operator \( o \) in states satisfying \( \phi \). The target policy is meant to be “closed” in that all its policy nodes yield successor states that are already accounted in the policy. In fact, \( \pi_{\phi} \) is basically encoding a fragment of the causal relevant graph \( CG(\mathcal{P}, \mathcal{G}) \).

\footnote{In terms of \( AO^* \)'s nomenclature, \( \pi_{\phi} \) corresponds to the best partial solution graph, whereas \( \pi_{\phi} \) corresponds to the so-called explication graph.}
We note that these \( \pi' \) nodes are the active nodes and the outcoming "tips" of the partial policy \( \pi_{AG} \) that are up for (further) regression. The idea, thus, is to grow \( \pi_{AG} \) until the initial state \( s_0 \) is accounted for.

So, exploration of graph \( CG(\mathcal{P}, \mathcal{G}) \) is performed between lines 5 and 9. In line 5, the "best" active policy node \( (\phi, o) \) in the (current) target policy \( \pi_{AG} \) is selected heuristically. If such node corresponds to \( \mathcal{P}' \)'s initial state (line 6), then a solution has been found, namely \( \pi_{AG} \). Otherwise, the algorithm further extends the causal graph by regressing the chosen node w.r.t. every domain operator (lines 8 and 9). All generated new nodes are placed into the base policy \( \pi_p \) for further checking and processing.

The selection of which open policy node in the target policy to further regress is central to our proposal and embodies the "directness" of our fixed-point computation: we regress a "robust" policy node—one that has already been proven to be part of a closed policy to the goal—that is most promising w.r.t. the initial state of the planning task. Concretely, the \( \text{GetBest}() \) function selects a policy node in \( \pi \) minimising the following evaluation function:

\[
f((\phi, o)) = Wh(\phi) + Q(\phi, o),
\]

where \( h(\phi) \) is a heuristic estimator to measure the cost-to-go from the initial state \( s_0 \) to a formula \( \phi \), \( Q(\phi, o) \) is the value function for the best known policy with execution \( \Lambda = s_0 \cdot \cdots \cdot s \vdash \phi \), and \( W \) is a weight factor that helps regulating how "greedy" the search is. When a new policy node \( (\phi, o) \) is created, \( Q(\phi, o) \) is initialised to the length of the path (in the fragment of \( CG(\mathcal{P}, \mathcal{G}) \) generated so far) from node \( \phi \) (and outgoing edge \( o \)) to node \( \mathcal{G} \), which can be greater or equal to the actual \( Q_{\min}(\phi, o) \) value. We note that these \( Q(\phi, o) \) can be revised, performing policy backups for instance, when a cheaper path to \( \phi \) in \( CG(\mathcal{P}, \mathcal{G}) \) is uncovered. When it comes to the heuristic \( h(.) \) we shall rely, in practice, on functions related to the \( \min\min \) relaxation (Geffner and Bonet 2013).\(^{3}\)

Let us now discuss the actual expansion via regression (lines 8 and 9). Basically, procedure \( \text{Regress} \) applies \( \mathcal{R}(\phi, o) \) and generates a new policy node \( (\phi', o') \) for each of the DNF clauses in \( \mathcal{R}(\phi, o) \). Provided \( o' \neq \bot \) (i.e., the node is not inconsistent) and \( h(\phi') < \infty \), the policy node is added to the "pending" basic policy \( \pi_p \). Note that if \( h(\phi') = \infty \), then the formula denotes a state that is not reachable from \( \mathcal{P}' \)'s initial state \( s_0 \) and hence is not relevant for the planning task. Non-directed symbolic planners, like MBP, would still consider those nodes.

After the target policy has been expanded once, the algorithm enters into the "checking" phase (lines 10 to 14). Intuitively, in this phase the algorithm checks—via procedure \( \text{Check} \)—whether a policy node \( (\phi, o) \) in the base policy \( \pi_p \) already entails "desired properties" and should be moved to the target policy \( \pi_{AG} \). In our case for strong cyclic policies, \( \text{Check} \) seeks to verify the entailment of property \( AG(\mathcal{EFG}) \) for the chosen node.\(^{4}\) When this happens, a set of policy nodes \( \mathcal{Y} \), including the one selected for checking and any other policy nodes that \( \text{Check} \) finds necessary for the proof, is incorporated into the frontier/tip of the target policy (line 13). As a side-effect of its execution, procedure \( \text{Check} \) may also generate new policy nodes \( \mathcal{Z} \) to be added to the pending base policy \( \pi_p \) (line 14). Condition \( \Psi \) determines when the checking phase is over and a new policy node in the target policy needs to be regressed. The details of such condition as well as those of procedure \( \text{Check} \) to perform the above inferences is to be covered in the next sections.

Finally, if neither target nor base policy changes in a full iteration, then it means that the whole causal relevant graph \( CG(\mathcal{P}, \mathcal{G}) \) has been generated and no node accounts for the initial planning state, thus proving the problem unsolvable. We note that BackwardsPolicySearch in this case will have computed the maximal set of partial strong cyclic policies supported by \( \mathcal{P} \), and such policies would be readily available for further analysis.

This concludes the general symbolic directed regression-based strategy for searching policies. Let us next go over the details of \( \text{Check} \), the heuristic \( h \) and \( \Psi \) for synthesising strong cyclic ones.

### Recognizing Strong Cyclic Solutions

In this section, we explain the procedure \( \text{Check} \) in BackwardsPolicySearch (Figure 2), so as to search in a directed manner for strong cyclic policies. We want \( \text{Check} \) to (correctly) recognize policies like the one shown on the left of Figure 1, and discard those that get stuck in dead-ends (e.g., those like the one depicted in the middle of Figure 1) or those that get trapped inside an infinite loop where it is not possible to escape even under fairness assumptions (e.g., like the ones depicted on the right of Figure 1).

We choose to implement \( \text{Check} \) following HDP (Bonet and Geffner 2003)\(^{5}\) in order to check that at least one of the many greedy policies

\[
\pi(\phi) = \arg\min_{o \in \{o|W(\psi, o) \in \pi_p, \phi \vdash v\}} Q(\psi, o)
\]

is both closed and leading to states where (current) target policy \( \pi_{AG} \) can be executed. Since the directed graph \( G_\pi \) induced by the executions \( \Lambda_\pi \) of \( \pi \) over \( S \) can be cyclic, the problem above is cast into that of verifying that the leaf vertexes of the (acyclic) directed graph \( G_\pi^{CC} \), with vertexes corresponding to the strongly connected components (SCCs) of \( G_\pi \), contains vertexes of \( G_\pi \) where \( \pi_{AG} \) is executable. Our reformulation of the \( \text{CheckSolved} \) procedure in HDP is detailed in Figure 3.

The backbone of both HDP and \( \text{Check} \) is Tarjan’s linear-time algorithm for detecting the s.c.c’s in a directed graph.

---

\(^{3}\)Since they have proven effective on nondeterministic reformulations of IPC benchmarks (Muis, McIlraith, and Beck 2012), we will especially focus on approximations of the optimal plans for the all-outcomes relaxation (Yoon, Fern, and Givan 2007).

\(^{4}\)We note it is possible to define \( \text{Check} \) so as to search for strong policies or even policies adhering to weaker solution notions, such as the ones recently by Domshlak (2013) for \( k \)-robust plans.

\(^{5}\)Still, we note that several other goal-based MDP planning algorithms could be used instead of HDP.
Figure 3: Algorithm for checking pending policy nodes for strong cyclic inference.

```plaintext
1: function CHECK ((φ, o), π_AG, π_P)
2:     Y ← ∅; Z ← ∅; W ← ∅
3:     visited ← ∅; S ← ∅
4:     index ← 0
5:     v₀ = [φ, (φ, o)]
6:     n.index ← ∞
7:     DFS(v₀, Y, Z, W, π_AG)
8:     π_P ← π_P \ W
9:     return [Y, Z]

10: function GENSUCCESSORS (χ, o, Succ, Z)
11:     for eᵢ in effects of o do
12:         Succ(i) ← NIL; Succ(i).index ← ∞
13:         if ψ = ψ', n = (ψ', (φ, o)) ∈ visited then
14:             Succ(i) ← n; continue
15:     Succ(i) ← [ψ, SelectGreedyPolicy (ψ, π_P)]
16:     if Succ(i) = Nil. then
17:         for o' ∈ O do
18:             if ψ ∧ Preₜ = ⊥ then continue
19:             if h(ψ ∧ Preₜ) = ∞ then continue
20:             Z ← Z ∪ {(ψ ∧ Preₜ', o')}
21:         return true
22:     return false

24: function DFS (v = [χ, (φ, o)], Y, Z, W, π_AG)
25:     if (φ, o) ∈ π_AG then return false
26:     if φ' = χ, [φ', (φ, o)] ∈ visited then return false
27:     v.Succ ← ∅
28:     visited.push(v)
29:     S.push(v)
30:     v.index ← v.lowlink ← index
31:     index ← index + 1
32:     flag ← GENSUCCESSORS (χ, o, v.Succ, Z)
33:     if flag then return flag
34:     for v' ∈ Succ do
35:         if v'.index = ∞ then
36:             flag ← flag ∨ DFS(v', Y, Z, W, π_AG)
37:             n.lowlink = min{v.lowlink, v'.lowlink}
38:         if v' ∈ S then
39:             v.lowlink = min{v.lowlink, v'.index}
40:         if flag then return flag
41:     else if v'.index = v.lowlink then
42:         retrieve SCC from S
43:     Backup(SCC)
44:     W ← W ∪ {[φ, o] | [φ', (φ, o)] ∈ SCC}
45:     if SCC connected to π_AG then
46:         Y ← Y ∪ {[φ', (φ, o)] | [φ', (φ, o)] ∈ SCC}
47:     else
48:         Z ← Z ∪ {[φ', (φ, o)] | [φ', (φ, o)] ∈ SCC}
49:     return true
50: return flag
```

directed graph (Tarjan 1972). The algorithm traverses the graph \( G_\pi \) in a depth-first manner, keeping track of (i) the visit number for each vertex \( v \) (the index variable); (ii) the SCC they belong to (the variable lowlink holds the identifier assigned to each discovered SCC, with each vertex \( v \) initially considered to be a SCC); (iii) the set of vertexes found along a path in \( G_\pi \) (the stack \( S \)); and (iv) the set of vertexes already visited, for which both index and lowlink have been already set (the visited hash table).

While HDP operates on an explicit graph that contains the possible executions of a greedy policy, we operate on an implicitly represented graph \( G_\pi \), generating its vertexes \( v \) as necessary. Each vertex \( v \) contains a policy node \( \langle \phi, o \rangle \in \pi_P \) and a formula \( \chi \), with \( \chi = \phi \), denoting the context where the greedy policy \( \pi \) is being executed. The DFS traversal of graph \( G_\pi \) stops whenever it is found that \( \langle \phi, o \rangle \in \pi_AG \) or that the context \( \chi \) has already been visited (lines 25 and 26). The GenSuccessors function (lines 10 – 23) generates the set of successor vertexes Succ of a given vertex \( v \), by progressing the context \( \chi \) through each effect \( e_i \) of operator \( o \) (line 13). This results in a formula \( \psi \) that denotes the set of states which will be reached when \( e_i \) is the actual outcome of \( o \). For each of these, we evaluate \( π(ψ) \) (calling the procedure SelectGreedyPolicy in line 16) to obtain a policy node \( \langle ψ', o' \rangle \), thus generating further vertexes of \( G_\pi \).

It is indeed possible that SelectGreedyPolicy does not return any policy node, since BackwardsPolicySearch—unlike standard model checking-based computations like MBP—generates first those paths in \( CG(P, G) \) which are deemed (by \( h \)) to lead to \( S_0 \) with the least cost. If there are one or more operators \( o' \) such that \( ψ ∧ Preₜ = \top \), then we generate new speculative policy nodes \( \langle ψ ∧ Preₜ', o' \rangle \) (line 21) that are to be added to base policy \( π_P \), and force Check to backtrack (lines 22 and 33). Doing so serves two purposes. First, it prevents Check from degenerating into a deep and badly informed rollout. Second, it schedules a call of Check over \( \langle \phi, o \rangle \) with (hopefully) more of \( CG(P, G) \) in \( π_P \). Since there is no guarantee that speculative nodes \( ψ ∧ Preₜ' \) are part of \( CG(P, G) \), we will avoid invoking Check on them, until (if at all) they are shown to belong to \( CG(P, G) \) when generated by Regress. Nonetheless, if they are reached by further calls to Check before that, more speculative nodes accounting for these executions will be generated. When that occurs, Check will be performing an iterative deepening forward search (Korf 1985), eventually leading to terminal non-goal states or to a node \( \langle \phi, o \rangle \in π_P \) known to be in \( CG(P, G) \). The Q-values for these speculative nodes, necessary to inform the greedy policy selection in SelectGreedyPolicy, are set to \( Q(ψ ∧ Preₜ', o') = c(o) + h(G) - h(ψ ∧ Preₜ') \), a rough yet admissible approximation on \( V_\min (ψ ∧ Preₜ') \).

Whenever \( v \) is considered to be the “root” vertex of a SCC, checking that index and lowlink match (line 41), and not contained in a bigger SCC, we determine if any vertex \( v' \) in SCC has a successor \( v'' = [χ, (φ, o)] \) where \( \langle φ, o \rangle \in \pi_AG \). This implies that SCC is connected to \( \pi_AG \) as well (lines 45). Otherwise, SCC may potentially be an infinite loop which cannot be escaped, and therefore, not part
of $\pi_{AG}$. The possible divergences between the context $\chi$ and the formulae in policy nodes $(\phi, o)$ are handled by removing the weaker conditions $\phi$ from $\Pi_P$ (line 8), previously collected in the $W$ set (line 44).

Another major divergence from HDP lies in how we perform the backups of the value function associated to the policy nodes, $Q(\phi, o)$. In order to ensure that $Q(\phi, o)$ is a monotone function, we cannot rely on the visit number index as HDP does, since the topological ordering defined by such values, in our setting, is relative rather than absolute. We need to ensure that the order in which backups of $Q(\phi, o)$ are done takes into account the relative distances between the initial state $s_0$ and the states denoted by the vertexes of the policy graph. We achieve this by implementing $SCC$ with a max-heap data structure, where they are ordered according to the heuristic value $h(\chi)$, with the visit number index breaking any ties. If $h$ is consistent, then the backups

$$Q(\chi, o) = c(o) + \max_{(\chi', o') \in SC(v)} Q(\chi', o')$$

done by Backup procedure (line 43) for each vertex $v = [\chi, (\phi, o)]$ in $SCC$ will produce monotone $Q(\chi, o)$ values.\footnote{Note that $h$ is consistent if $h(\phi) \leq c(o) + h(\psi)$, where $\psi$ is such that $next(o, \phi) \models \psi$.}

**GRENDEL: A Strong Cyclic FOND planner**

The ideas discussed in the two previous sections have been used to build a new FOND planner, called GRENDEL. We have implemented GRENDEL in Python+C++ borrowing the extension by (Muir, McIlraith, and Beck 2012) of the parser for deterministic PDDL in the FASTDOWNSWARD framework. We point out that we have consciously not tried to optimize the system in any way, but rather, demonstrate how concepts and algorithms from classical planning can be borrowed off-the-shelf, rather than whole planning systems.

We next describe the heuristic $h$ and the stopping criterion $\Psi$ that were left undefined when discussing BackwardsPolicySearch. So, the heuristic function used by GRENDEL is defined as follows:

$$h(\phi) = \max\{h_{add}(\phi; s_0), h_\gamma(\phi)\},$$

where $h_{add}(\phi; s_0)$ stands for the evaluation of the set of literals in $\phi$ on the additive heuristic (Geffner and Bonet 2013) computed from the initial state $s_0$ over the all-outcomes determination $P_D$ of the input FOND task $P$. In turn, $h_\gamma(\phi)$ is the heuristic that results from verifying that $\phi$ is consistent with a CNF formula $\gamma$, consisting of binary clauses $\neg l \lor \neg l'$, for each pair of literals $l, l'$ in the mutex groups (Helmert 2009) computed by FASTDOWNSWARD. Formally:

$$h_\gamma(\phi) = \begin{cases} 0 & \phi \land \gamma \not\models \bot \\ \infty & \text{otherwise} \end{cases}$$

The main reason to have $h$ defined is this manner is to make evaluations of policy node $(\phi, o)$ as fast to compute as possible, yet reasonably informed.

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Benchmark & T & PRP & GRENDEL & Ratio \\
\hline
TrianglETireworld & 40 & 40 & 40 & .67 \\
ScTrianglETireworld & 40 & 5 & 14 & .38 \\
Faunts & 55 & 55 & 55 & 2.17 \\
Forest & 90 & 76 & 0 & n/a \\
First Responders & 100 & 100 & 34 & 88 \\
BlocksWorld & 30 & 30 & 6 & 2.54 \\
\hline
\end{tabular}
\caption{Coverage and relative run-times of PRP and GRENDEL over several FOND benchmarks. Column $T$ stands for the number of instances in the benchmark, where columns PRP and GRENDEL are the number of instances solved by each planner. The last column, shows the ratio between GRENDEL and PRP run-times averaged over the instances that both planners solve within the imposed time and space limits (1800 seconds and 2 GBytes of memory).}
\end{table}

The overarching idea for defining the stopping condition $\Psi$ in BackwardsPolicySearch is that we should stop the checking of nodes in $\Pi_P$ for policy “closeness,” when it is judged that it may be more promising to further expand the outer policy nodes in the current target policy $\pi_{AG}$ instead. Hence we define $\Psi$ as follows:

$$\Psi = \min_{(\phi, o) \in \pi_P} f((\phi, o)) < \min_{(\phi', o) \in \pi_{AG}} f((\phi', o)).$$

That is, algorithm BackwardsPolicySearch (Figure 2) jumps into the regression phase so as to further Regress the best policy node in $\pi_{AG}$ (the frontier of the current target policy) if such node is judged, by the evaluation function $f$, to be better (i.e., closer to the initial state) than the best, not yet “closed,” policy node in policy $\pi_P$.

**Evaluation and Analysis**

We have tested GRENDEL and compared it with the FOND planner PRP (Muir, McIlraith, and Beck 2012) over the 2006 IPC benchmark BlocksWorld, the 2008 IPC benchmarks Faunts, First Responders and Forest, the TriangleTireworld benchmark presented in (Little and Thébaux 2007), and new benchmark ScTrianglETireworld derived from the former and the 2006 IPC benchmark Tireworld. ScTrianglETireworld results from changing the action schemas discussed in (Little and Thébaux 2007) for those in IPC 2006 Tireworld so solutions ought to be strong cyclic policies, rather than just strong as in the original TriangleTireworld. In all cases we used the FOND reformulation of these probabilistic planning tasks due to Muise, McIlraith, and Beck (2012).

The results of our evaluation are shown in Table 1. As one can see, GRENDEL outperforms PRP in Tireworld and its derivatives. First, on the version of TriangleTireworld which only allows strong solutions, GRENDEL takes 67% of PRP’s run-time (i.e., around 30% faster), and both planners solve all instances, some of them with several thousand operators, under 1000 seconds. Most importantly, PRP is outperformed by GRENDEL on the strong cyclic version ScTrianglETireworld, both in terms of coverage and run-time. In both domains, the optimal plans for $P_D$ are
not embedded in the execution of any strong or strong cyclic policy; even if PRP copes with this issue when the solution is acyclic, this ability does not seem to carry over to problems with cyclic policies. This highlights the limitations of relying on incrementally extending weak plans obtained from classical planners on domains where there is a tension between plan efficiency (execution length) and robustness (guarantees on executions achieving the goal). GRENDel directed regression approach, instead, only extends robust plans and performs less search than PRP.

On the other hand, GRENDel performance on the last four benchmarks is clearly not as good as PRP, both in terms of coverage and speed. Arguably, these four domains may not be considered “nondeterministically interesting” (Little and Thiebaux 2007), as opposed to the first two. In particular, they have the property that almost every weak plan is an execution of some strong cyclic solution. This suggests we need to consider a more diverse set of “nondeterministically interesting” domains, with parameters that allow to precisely control to what degree instances are “interesting”.

Nonetheless, we have identified three causes for GRENDel’s poor performance, which could inform future work. First, the regression-based reasoning results in huge numbers of inconsistent formulae (i.e., states) being produced, by both Regress and Check, when generating speculative nodes for consideration. Since \( h \), is generally very poorly informed in those domains, those impossible states are not identified as such. Forward planners will, of course, never encounter those states as they are assumed to being with a consistent initial state. Incorporating “consistent” state constraints restricting legal initial states (e.g., it is not legal to have a circular block tower or be holding two blocks simultaneously), as typically done in reasoning about action (Reiter 2001), would address this problem immediately.

Second, in many cases, the all-outcomes determinization has no valid (weak) plans from many states, and this is swiftly reported by the very efficient classical planner embedded in PRP. In an extreme case, for example, 25 instances of FIRST RESPONDERS domain has no weak solution from the initial state itself. Allowing GRENDel to perform a quick “weak” solution existence test right from the start would close the gap in the benchmark tested. However, this may not be an adequate long-term solution, as classical planners and standard deterministic domains are not well suited for cases admitting no (weak) solution at all—a classical planner may just never return.

Third, and most interesting, we have observed GRENDel to pursue, simultaneously, a huge number of partial strong cyclic policies—all with the same \( Q \)-value—in BLOCKS\_WORLD, FIRST\_RESPONDERS, and even (SC)TRIANGLE\_TIRE\_WORLD instances. That is, GRENDel oscillates from policy to policy rather than committing to one of them. The fact is that, in a problems with reversible operators like BLOCKS\_WORLD, the heuristic guidance provided by the values of \( h_{dead} \) is clearly not enough to force the required commitment. Developing the notion of helpful operators (Hoffmann and Nebel 2001) in regression search or incorporating techniques to restrict search oscillation as in (Dionne, Thayer, and Ruml 2011)’s Deadline-Aware-Search, are two possible avenues to handle this issue. Finally, the FOREST scenario—a grid navigation problem where in order to proceed towards the destination it is necessary to solve a small classical planning problem—happens to combine all the three characteristics above and completely defeats GRENDel.

We close by noting that we have also tested MBP on the TIRE\_WORLD instances. As expected, GRENDel performs better (\( \approx 10 \times \) faster in solvable cases, and \( \approx 2.5 \times \) faster in unsolvable ones). While this should not come as a surprise—after all GRENDel is a type of directed MBP—we think such comparison is not too meaningful at this point and a comparison should be made with more state-of-the-art verification systems, such as game solver NuGAT.\(^9\)

Conclusions

We have provided a middle ground between directed search and symbolic regression-based reasoning for planning under nondeterministic actions. Our proposal is able to reason about set of states and transitions in a succinct way (as symbolic approaches do) while biasing the plan/policy generation (as heuristic planners do). In doing so, the framework prioritizes the expansion of robust (i.e., closed) partial policies which we believe is a sensible strategy when it comes to non-deterministic settings where plan efficiency goes at odds with robustness. As made explicit along the sections, the technique developed draws from many existing ideas. Besides providing the theoretical framework (e.g., the causal relevant graph and the relationship with dynamic programming) and the overall planning strategy (Figures 2 and 3), we demonstrated how all the ingredients can be put together in the GRENDel planner. While our planner is far from optimized, it already shows important improvements in FOND planning problems where there is a tension between robustness of plans and their “efficiency”—the quickest plan may be the most fragile. Our approach resembles that of the probabilistic planners RETR\_ASE (Kolobov, Weld, and Mausam 2012) and RFF (Teichteil-Königsbuch, Kuter, and Infantes 2010) that address this tension too, but departs from them in that, like PRP, policy backups are not performed directly on states, but rather on sets of states.

Since what we have proposed is a family of potential “hybrid” algorithms, there are many possibilities for improvement, study, and optimization. In particular, how to incorporate “commitment” in a regression-based approach as well as some convenient mechanism for avoiding purely inconsistent states are two aspects that we expect to yield significant improvements. We have already provided some hints for exploring these issues. An important task, we believe, is to carry an analysis characterizing the boundaries between classical and FOND planning, in a similar way as done by

---

\(^{8}\) Personal communication with Christian Muise indicates that PRP’s performance may be due to PRP’s inability to generalize dead-ends in the problem that are not dead-ends in \( p_0 \), and shortcomings in the strong-cyclic detection algorithm used.

\(^{9}\) http://es.fbk.eu/tools/nugat/
Little and Thiébaux (2007) for planning and replanning approaches. We also observe that standard benchmarks for deterministic planning may not generalize to “nondeterministically interesting” problems. To that end, we plan to perform deeper empirical analysis on more natural nondeterministic domains, such as those found in (Ramirez, Yadav, and Sardina 2013) or (Patrizi, Lipovetzky, and Geffner 2013).

References