PRP Rebooted: Advancing the State of the Art in FOND Planning

Christian Muise[1,3], **Sheila A. McIlraith**[2,3], **J. Christopher Beck**[2]
[1] Queen's University, Kingston, Canada
[2] University of Toronto, Toronto, Canada
[3] Vector Institute for Artificial Intelligence, Toronto, Canada
christian.muise@queensu.ca, sheila@cs.toronto.edu, jcb@mie.utoronto.ca

Abstract

Fully Observable Non-Deterministic (FOND) planning is a variant of classical symbolic planning in which actions are nondeterministic, with an action's outcome known only upon execution. It is a popular planning paradigm with applications ranging from robot planning to dialogue-agent design and reactive synthesis. Over the last 20 years, a number of approaches to FOND planning have emerged. In this work, we establish a new state of the art, following in the footsteps of some of the most powerful FOND planners to date. Our planner, PR2, decisively outperforms the four leading FOND planners, at times by a large margin, in 17 of 18 domains that represent a comprehensive benchmark suite. Ablation studies demonstrate the impact of various techniques we introduce, with the largest improvement coming from our novel FOND-aware heuristic.

Introduction

Fully Observable Non-Deterministic (FOND) planning is a variant of classical symbolic planning in which actions are non-deterministic, the finite set of possible action outcomes is known a priori, and the realized outcome in a particular instant is observed following execution of the action (e.g., (Daniele, Traverso, and Vardi 1999; Cimatti et al. 2003)). As with classical planning, FOND planning assumes full observability, but the uncertainty in the outcome of actions during plan generation necessitates a new form of (contingent) solution – both in terms of representation and plan synthesis.

Since its introduction 20 years ago, FOND has emerged as a popular and highly versatile computational paradigm with applications ranging from generalized planning (Illanes and McIlraith 2019; Bonet et al. 2020) and robot planning (Andrés, de Barros, and Delgado 2020) to dialogue-agent design (Muise et al. 2019), and reactive synthesis from logical specification (Camacho et al. 2017). Given the diversity of applications for FOND planning, advances in FOND planning have the potential for significant impact.

Following the development of the original MBP FOND planner, based on model checking (Cimatti et al. 2003), we have witnessed a stream of innovations. In a departure from MBP, the NDP (Kuter et al. 2008) and FIP (Fu et al. 2011) planners reformulated the FOND problem into a classical planning problem, repeatedly solving this problem, to iteratively solve the original FOND problem. In 2012, the PRP planner emerged as the state of the art in FOND planning, often showing orders of magnitude improvements in plan generation time and/or solution size (Muise, McIlraith, and Beck 2012). Like NDP and FIP, PRP employed classical planning over a reformulation of the FOND problem, but much of its performance gains were the result of identifying and encoding only those aspects of the state that were relevant to plan validity. This was accomplished by representing families of states as conjunctive formulae and employing regression rewriting, a form of pre-image computation, to establish relevance (Waldinger 1977; Reiter 2001; Fritz and McIlraith 2007). Subsequent techniques explored new concepts for computing solutions to FOND problems: from the backwards search of GRENDEL (Ramírez and Sardiña 2014), to the SAT encoding of FONDSAT (Geffner and Geffner 2018), to the policy-based search of MyND and Paladinus (Mattmüller et al. 2010; Pereira et al. 2022). These new planners were often published with additional benchmark problems that showcased the new planner's potential and, in a number of cases, their superior performance compared to the incumbent, PRP.

In this work, we introduce *PRP Rebooted* (PR2), a new FOND planner that leverages the insights of a generation of FOND planners to realize a significant advance in the state of the art. Like NDP, FIP, and PRP before it, PR2 adopts the approach of repeatedly replanning in a classical planning reformulation. Further, it uses a powerful solution representation inspired by GRENDEL and FONDSAT, while maintaining the solving strength of PRP to produce this solution. The novel techniques introduced in the PR2 planner include better handling of deadends, a preprocessing step to simplify problems, and a new FOND-aware heuristic for the classical planning sub-process.

PR2 outperforms the four leading FOND planners (MyND, FONDSAT, PRP, Paladinus), at times by a large margin, in 17 of 18 domains that represent a comprehensive benchmark set (five of 800 problems are solved by PRP, but not PR2). Through an ablation study, we evaluate the impact of individual technical components on overall planner performance. Ultimately, PR2 constitutes a significant advancement in the field of FOND planning and a new state of the art for the planning formalism.

Preliminaries

Our notation and definitions largely follow the existing literature that uses multi-valued variables for representation.

Definition 1 (FOND Task (Muise, McIlraith, and Beck 2012))**.** A fully-observable non-deterministic (FOND) planning task is a tuple $\langle \mathcal{V}, s_0, s_*, \mathcal{A}\rangle$, where $\mathcal{V}$ is a set of finite domain variables, s_0 is the initial state (complete setting of $\mathcal{V}$), s_* is the goal condition (partial assignment to $\mathcal{V}$), and $\mathcal{A}$ is the set of potentially non-deterministic actions. Each action $a \in \mathcal{A}$ is represented by a tuple $\langle Pre_a, \mathit{Eff}_a\rangle$, where Pre_a is a partial assignment to $\mathcal{V}$ that stipulates when the action is executable and Eff_a is a set of outcomes, one of which will occur at execution. Each outcome $o \in \mathit{Eff}_a$ is a partial assignment to $\mathcal{V}$ and signifies the updates to the state.

For updating partial or complete states (represented as a partial or complete assignment to $\mathcal{V}$ respectively), we define the $\oplus$ operator (where p_1 and p_2 are (potentially partial) assignments to $\mathcal{V}$) as follows:

$$(p_1 \oplus p_2)(v) = \begin{cases} p_2(v) & \text{if } p_2(v) \text{ is defined} \\ p_1(v) & \text{otherwise} \end{cases}$$

We say that partial states p_1 and p_2 are *consistent* (written $p_1 \approx p_2$) when $\forall v \in \mathcal{V}$, either $p_1(v) = \bot$ or $p_2(v) = \bot$ or $p_1(v) = p_2(v)$, where $\bot$ indicates the variable is undefined. Action $a \in \mathcal{A}$ can be executed in state s only when $s \approx Pre_a$. If outcome $o \in \mathit{Eff}_a$ occurs as a result of applying a in state s, then the updated state is defined as $s \oplus o$.

In FOND, an initial state s_0 and set of actions $\mathcal{A}$ induce a set of reachable states. Despite the nondeterminism in FOND actions, a policy, π, is a mapping from states to individual actions (rather than an action distribution, for example). We restrict our attention to defining this mapping for the subset of states that are reachable from s_0, and as such we refer to such policies as a *partial policy*. If every state reachable by a policy itself has a mapping to an action then we say that the policy is *closed*. Here, a *plan* is a policy that is guaranteed to achieve a goal, potentially under some assumptions or restrictions. In the context of FOND planning, a *weak* plan is one that will reach the goal under some realization of the non-deterministic action effects; it need not be, and is typically not, closed. A *strong* plan is a closed (partial) policy and is guaranteed to reach the goal in a finite number of steps. Finally, a *strong cyclic* plan is a closed (partial) policy where the policy embodies a weak plan for every state that is reachable by the partial policy. A strong cyclic plan provides a solution to the FOND planning under an assumption of fairness (Cimatti et al. 2003). In this paper, we are concerned with strong cyclic plans.

As is common with several FOND approaches, we use an *all-outcomes determinization*. This is a reformulation of the problem so that every action $a \in \mathcal{A}$ is replaced with $|\mathit{Eff}_a|$ actions, each corresponding to one of the outcomes in Eff_a. Solving the classical planning problem created by the all-outcomes determinization gives us a sequence of actions that achieves the goal under some possible realization of environmental uncertainty and, thus, is a weak plan. Much of our work relies on computing relevant conditions for an action to achieve some partial state. We exploit regression, a pre-image computation (Waldinger 1977; Reiter 2001).

Definition 2 (Logical Regression)**.** Given a partial state p, action $a \in \mathcal{A}$, and outcome $o \in \mathit{Eff}_a$, the logical regression of p through the action a and outcome o is defined as:

$$\mathcal{R}(p, a, o)(v) = \begin{cases} Pre_a(v) & \text{if } Pre_a(v) \neq \bot \\ \bot & \text{else if } p(v) = o(v) \\ p(v) & \text{otherwise} \end{cases}$$

We only define (and apply) regression if $p \approx o$. We use $Can\mathcal{R}(p)$ to denote all of the actions and outcomes that have regression defined for the partial state p:

$$Can\mathcal{R}(p) = \{\langle a, o\rangle \mid a \in \mathcal{A}, o \in \mathit{Eff}_a, \text{and } p \approx o\}.$$

Finally, modern FOND planners need to avoid deadends – reachable states from which the goal is not reachable. We adopt PRP's concept of a *forbidden state-action pair* (FSAP). An FSAP is a tuple $\langle ps, a\rangle$, denoting that action a is forbidden in any state consistent with partial state ps because it could lead to a deadend. In the algorithms that follow, $\mathit{fsap} = \langle ps, a\rangle$ and we use $\mathit{fsap}.ps$ and $\mathit{fsap}.a$ to reference the corresponding partial state and associated action.

Approach

Similar to NDP, FIP, and PRP, PR2 iteratively solves a determinized version of the FOND planning problem to compute weak plans, incorporating them into an overall solution. We view PR2 as an evolution of PRP, as we use a similar high-level search framework, leveraging regression rewriting for relevance determination and forbidden state-action pairs for deadend avoidance. The PR2 planner is also built on top of the Fast Downward planning system (Helmert 2006) but otherwise shares minimal implementation overlap with PRP.

The overall approach is summarized in Algorithm 1 and described throughout this section (cf. Appendix A in (Muise, McIlraith, and Beck 2023) for an expanded version). PR2 incrementally builds a solution, potentially making missteps along the way. If the found solution is strong cyclic, then we are done. Otherwise, information is retained on what went wrong, and the process restarts.

Solution Representation and Construction

Algorithm 1 endeavours to compute a strong cyclic plan by incrementally building a solution from weak plans in the all-outcomes determinization. A key component of PR2 is the internal representation of the incumbent solution in terms of two structures: (1) a controller - a directed graph whose nodes capture (among other things) a compact representation of state-action pairs (i.e., a state the solution can reach and an action that is applicable in that state), and outgoing edges connecting to successor nodes; and (2) the reachable state space explored by the solution to this point, where nodes are *complete* state-action pairs. We refer to the first as CONTROLLER and the second as REACHABLE (i.e., an incumbent solution $sol = \langle \text{CONTROLLER}, \text{REACHABLE}\rangle$), with the former inspired by GRENDEL and FONDSAT and the latter similar to MyND, FIP, and Paladinus. Both play

Algorithm 1: PR2 High-Level Planner

```
   Input: FOND planning task, Π = ⟨V, s0, s*, A⟩
   Output: Policy
1  incumbent = make_empty_solution();   FSAPS = ∅;
2  while !incumbent.is_strong_cyclic() do
3      sol = make_empty_solution({s0});
4      while sol.REACHABLE contains unhandled nodes do
5          n = sol.REACHABLE.pop_unhandled_node();
6          switch analyze_node (n) do
7              case 0: skip_if_strong_cyclic(n)
8              case 1: skip_if_poisoned(n)
9              case 2: match_complete_state(n)
10             case 3: apply_predefined_path(n)
11             case 4: match_complete_state(n)
12             case 5: find_and_update_weak_plan(n)
13             case default (case 6) do
14                 record_deadend(n);
15                 if n.state == s0 then
16                     return make_policy(incumbent.CONTROLLER);
17     if sol.success_rate() ≥ incumbent.success_rate() then
18         incumbent = sol;
19 return make_policy(incumbent.CONTROLLER);
```

an essential role to PR2: the dual representation allows PR2 to maintain the inner-loop search progress with the reachable state space while maintaining a compact controller representation of the partial solution. Intuitively, REACHABLE represents the search progress of PR2, including the open nodes that must be extended for a final solution, while CONTROLLER is a compact representation of a partial solution, including the conditions under which it is guaranteed to achieve the goal. For both graphs, we use $outcome(n_1, n_2)$ to label the edge between n_1 and n_2 with the outcome – from the set of effects of the action of n_1 in this context. More formally, the nodes in CONTROLLER are as follows.

Definition 3 (SolStep)**.** A SolStep, ss, is a node in the CONTROLLER represented by a tuple, $ss = \langle ps, a, in, out, sc\rangle$ where ps is a partial state, a is an action, in is a list of SolSteps that are connected to ss through an incoming edge (i.e., predecessors), out is the list of successors (directly corresponding to the outcomes of a), and sc is a Boolean flag that indicates if the SolStep is "strong cyclic" (discussed below). Every SolStep in (the possibly empty) $ss.in$ is defined, but some in $ss.out$ may be undefined.

In Figure 1, we show the CONTROLLER and REACHABLE representations. We may not explore every complete state reachable by the solution – unlike planners that rely solely on representations like REACHABLE – as we only continue exploring if CONTROLLER is not guaranteed to succeed. While a single node in REACHABLE corresponds to exactly one SolStep (i.e., node in CONTROLLER), it is often the case that a single SolStep corresponds to several nodes in REACHABLE. We use $\text{REACHABLE}(ss)$ to refer to the set of nodes in REACHABLE that are associated with SolStep ss, and $\text{CONTROLLER}(n)$ to refer to the single SolStep in CONTROLLER that is associated with node n in REACHABLE.

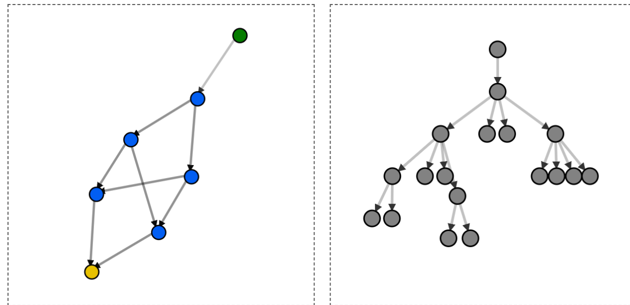


Figure 1: CONTROLLER (left) and REACHABLE (right) representations. In the CONTROLLER, green represents the initial SolStep to execute, gold the goal, and blue corresponds to SolSteps that have been established as strong cyclic (and thus a solution is guaranteed from that point on).

Algorithm 1 determines how a node in the search space should be handled via a single case `analyze_node` (line 6) and a final catch-all case for nodes that are deadends (line 13). Next, we summarize how each case is identified and maintains the two components of our incumbent solution.

Case 0: Strong Cyclic Nodes As a base case, if the search node we pop from the open list is handled by a node in CONTROLLER that is strong cyclic, then nothing need be done.

Case 1: Poison Nodes A node is "poisoned" if it, or one of its ancestor in CONTROLLER, is flagged as a forbidden state-action pair (FSAP). In this situation, we do not process the node further and the search continues. The poisoning of nodes happens in Case 6 with details in Section 15.

Case 2: Complete State Match In this case, we have an exact match of the state that corresponds to the popped node and a node in REACHABLE. At this point, we can assume that the search for a solution from this state has already run its course, and we connect things appropriately in both CONTROLLER and REACHABLE. This may involve creating a new edge in CONTROLLER but not introducing new nodes.

Case 3: Predefined Path In this case, there is already a predefined path for the search node: the SolStep associated with the parent search node in CONTROLLER has an edge defined for the outcome that has led to this particular search node. We, therefore, simply add all of the successor nodes to REACHABLE according to the action specified in CONTROLLER. This situation arises when the REACHABLE being constructed stumbles upon a part of the CONTROLLER solution that already contains enough information to handle those newly reached complete states. It is effectively overlaying new complete states in the expanded REACHABLE on top of existing SolSteps in CONTROLLER. If the CONTROLLER nodes were marked strong cyclic, then this node would have been handled with Case 0.

Case 4: Hookup Solution Steps Case 4 arises when the complete state corresponding to the current search node can be handled by a SolStep in CONTROLLER. Formally, for

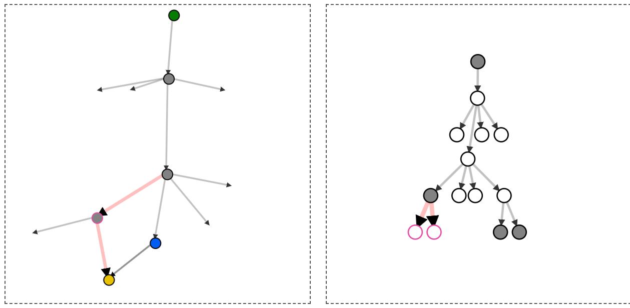


Figure 2: New path corresponding to a weak plan in the CONTROLLER (left) and REACHABLE (right). In the CONTROLLER, highlighted edges show the new path, and dangling edges correspond to parts of the solution yet to be explored. In the REACHABLE, the newly added nodes are highlighted, and white nodes correspond to search nodes that are still on the stack in Algorithm 1.

node n and SolStep ss, this means that $\forall v \in \mathcal{V}$, if $ss.ps \neq \perp$ then $n.state(v) = ss.ps(v)$. When this holds, a new edge is created from the previous search node's SolStep to this new matching one along the corresponding outcome. Case 4 can be seen as a generalization of Case 2. When these new connections are made, the CONTROLLER is updated through the fixed-point regression procedure defined in Section 19 that ensures that the partial states associated with each SolStep in CONTROLLER capture precisely what must hold in order for the solution to be strong cyclic from that point on.

Case 5: Find and Update Weak Plan If the previous cases fail to capture the current search node (and associated state), we turn to finding another weak plan. We take the state associated with the current node in the search and first attempt to "plan locally" before computing a new plan for the goal. Planning locally is a strategy adapted from PRP, where the goal is temporarily set to the partial state of the SolStep in CONTROLLER that the parent SolStep expected to be in – every time a weak plan is produced, one outcome is chosen in the all-outcomes determinization, and this dictates the temporary goal for planning locally. Figure 2 illustrates a new (short) path that was found for a Case-5 node. Note that `find_and_update_weak_plan` will add new nodes and edges to both the CONTROLLER and REACHABLE graph. It also adds all of the successors of node n to the REACHABLE so that they may be subsequently considered. Our weak planning procedure also includes (1) stopping the search when a state is reached such that some SolStep in the CONTROLLER matches (and thus we have a solution); (2) recording and generalizing all deadends found (cf. Section 15); and (3) using an FSAP-aware heuristic that takes non-determinism into account (cf. Section 15).

Case 6: Deadend In the final case, there is no weak plan to take us to the goal from the current node's state. We flag the current state as a deadend, generalize and apply regression if possible, and then poison the parent of this node and all of the parent's descendants in REACHABLE (cf. Section 15).

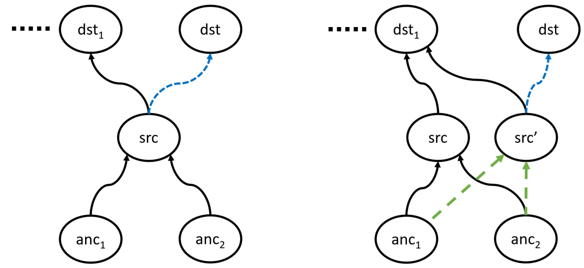


Figure 3: New connection in Fixed Point Regression

The final task of Algorithm 1 (line 19) is to convert the solution's CONTROLLER into a policy. For a complete state s, assume ss is the SolStep closest to the goal among all consistent SolSteps (i.e., $s \approx ss.p$). The action executed by *make_policy*(CONTROLLER) for state s would thus be $ss.a$.

Strengthening & Fixed-Point Regression

Ideally, our CONTROLLER embodies the conditions required for a solution fragment to achieve the goal. Specifically, each node n in CONTROLLER has a partial state associated with it that captures *what must be true in order for the CONTROLLER to be a strong cyclic solution for a complete state that is consistent with* $n.p$. When a new connection is made in the CONTROLLER, we must apply regression on that connection. This, in turn, may alter the partial state associated with the new connection's source node, which then necessitates the repeated regression recursively back through the CONTROLLER until no further updates are made.

Figure 3 shows the situation where we have a new blue connection being made in CONTROLLER between SolSteps src and dst (assuming dst is not already in $src.out$). The recursive call is made with src and dst, and is shown in Algorithm 2. The blue edge between src' and dst is the newly formed link, and the green edges between anc_1 (anc_2) and src' show the recursive calls made on Algorithm 2's line 12.

The algorithm repeatedly strengthens the conditions on the nodes found in CONTROLLER, and leaves the strengthened version as a new copy. We create new copies rather than overwrite the original since different paths in REACHABLE may overlay the same nodes in CONTROLLER and there is no guarantee that the states on the other paths will be consistent with the newly computed partial states.

Due to space, we forgo elaborating on the finer details of the process, including optimizations for when we can avoid cloning src, bookkeeping that removes unused aspects of the CONTROLLER, and proper maintenance of the strong cyclic property on nodes in CONTROLLER.

Full Strong Cyclic Marking

Because of the complex ways modifications are made to the CONTROLLER, we re-establish the strong cyclic property of the CONTROLLER nodes every time there is a modification that may change their status. The property captures a guarantee that the goal will be reached if the state of the world matches the partial state associated with a node in the CONTROLLER marked "strong cyclic". The procedure

Algorithm 2: Fixed-Point Regression, `fpr`

Input: Two nodes src and dst in CONTROLLER that should be connected, along with the associated REACHABLE nodes n_{src} and n_{dst}.

// If either src or dst are poisoned, then return since this part of the CONTROLLER *will be discarded.*
if `is_poisoned(src)` *or* `is_poisoned(dst)` **then**
  return

// Compute the combined regression
$ps = src.p \oplus \mathcal{R}(dst.p, src.a, outcome(src, dst))$;
if $ps == src.p$ **then**
  return *// Nothing to update*

// Clone src and add it to CONTROLLER
$src' = \langle ps, src.a, src.in, src.out \cup \{dst\}, src.sc\rangle$;
CONTROLLER.`add_node(`src'`);`

// Update the graph mappings
CONTROLLER (n_{src}) = src';
REACHABLE (src) = REACHABLE (src) $\setminus \{n_{src}\}$;
REACHABLE (src') = $\{n_{src}\}$;

// Recurse backwards
for $anc \in n_{src}.in$ **do**
  `fpr(`CONTROLLER(anc), src', anc, n_{src}`);`

Algorithm 3: Strong Cyclic Marking

Input: Incumbent CONTROLLER

// Identify all of the nodes not marked strong cyclic
1 $unmarked = \{n \mid n \in$ CONTROLLER and $\neg n.sc\}$;
// Flag not strong cyclic those with an unhandled successor
2 $notSC = \emptyset$;
3 $Q = \emptyset$;
4 **for** $n \in unmarked$ **do**
5   **if** $|n.out| \neq |Eff_{n.a}|$ **then**
6     $notSC.add(n)$;
7     $Q.add(n)$;

// Recurse backwards, marking more non-strong cyclic
8 **while** Q *is not empty* **do**
9   $n = Q.pop()$;
10   **for** $n' \in n.in$ **do**
11   **if** $n' \notin notSC$ **then**
12       $notSC.add(n')$;
13       $Q.add(n')$;

// Finally, mark all remaining ones as strong cyclic
14 **for** $n \in (unmarked \setminus notSC)$ **do**
15   $n.sc = true$;

for re-establishing the strong cyclic property of all nodes is given in Algorithm 3, and it assumes SolSteps in CONTROLLER default to having the $.sc$ property be false.

The algorithm relies on a few key assumptions. One is that the conditions associated with a SolStep are sufficient for executing from that point on, regardless of future action outcomes. This is guaranteed by the fixed-point regression procedure discussed in Section 19. The other property is that the $.sc$ property of a SolStep is monotonic, in the sense that once it becomes true, it will stay true during the construction of CONTROLLER.

Intuitively, this is a safe assumption to make because of the manner in which the CONTROLLER is constructed – a SolStep is added only if there is a path to the goal, and no modification removes this property. Seen another way, if there is some sequence of actions and outcomes that follows from a SolStep in CONTROLLER and leads to an unhandled outcome, then that final SolStep n would necessarily have $n.sc = false$. Inductively, this property would apply all the way back through its predecessors. The nodes that remain in $(unmarked \setminus notSC)$ on line 14 must then have a path to the goal down every possible extension and can safely be marked as strong cyclic.

Deadend Handling & Poisoning

If there is any point during the search when we discover a deadend as part of the solution, we take an aggressive strategy to immediately invalidate any aspect of the search that is impacted. The strategy has two components: (1) deadend generalization and regression and (2) node poisoning.

We first employ deadend generalization in the same way as SixthSense and PRP (Kolobov, Mausam, and Weld 2010; Muise, McIlraith, and Beck 2012): variables are progressively relaxed as long as a delete-relaxed (Bonet and Geffner 2001) deadend remains, and the final partial state represents a generalization of the complete-state deadend that was detected. Similar to PRP, the generalized deadend de is then regressed through every action and outcome that is consistent with de, in order to produce a set of forbidden state-action pairs (FSAPs) used in subsequent search iterations:

$$FSAPs(de) = \{\langle \mathcal{R}(de, a, o), a\rangle \mid \langle a, o\rangle \in Can\mathcal{R}(de)\}.$$

Second, given the node n that was determined to be a deadend, we poison the portion of the REACHABLE search space corresponding to the parent of n and all the parent's descendants. As described in Case 6 above, a poisoned node is subsequently ignored in this iteration. This reasoning follows from how the top-level search proceeds: if we cannot guarantee the goal is reachable from a node, then future passes should skip this part of the search entirely. PR2 stops looking for a solution down a path that will be skipped in a future iteration, thus saving search effort.

FSAP-aware FF Heuristic

FOND planners based on weak planning procedures carry very little of the FOND setting to the classical planner subcalls (Pereira et al. 2022). PR2 addresses this issue in several ways. Similar to PRP, PR2 (1) stops searching when the incumbent solution recognizes the state, and (2) does not consider forbidden actions when expanding a search node. PR2 further employs a custom classical heuristic that uses information from the FSAPs found so far to re-weight the h^{FF} heuristic computation (Hoffmann 2001).

The h^{FF} heuristic operates by finding a plan in the *delete relaxation* of the classical planning problem – all delete effects are ignored (in our case, a variable can take on multiple values). The actions at the start of this delete-relaxed plan are called *helpful actions*. Two key changes were made to the FF heuristic to take the FSAPs into account. First, when computing the helpful actions for a state, we only allow those that are not forbidden. Second, we change the heuristic value by adding a penalty to any potentially forbidden action that is used in the computation of the heuristic. This "potentially forbidden" property is calculated by observing which FSAPs have their conditions satisfied by the propositions reachable in the delete relaxation. Such actions may not truly be forbidden by the time they are needed/used. Thus, removing them entirely may cause the state to be mistakenly presumed to be a deadend. Thus, we only treat their presence with a penalty to the heuristic value (i.e., "pay a price" for needing an action that may potentially be forbidden). See Appendix B in (Muise, McIlraith, and Beck 2023) for further details.

Redundant Object Sampling

Prior to attempting to solve a problem, PR2 will explore simplifications to the instance by attempting to remove redundant objects.[1] This sound-but-incomplete transformation comes from the following observation: *if we delete some of the objects in a problem, and the resulting instance has a strong cyclic plan, then it must be a plan to the original problem.* Intuitively, removing objects has an impact equivalent to disallowing or deleting several actions, thus not invalidating any plans that remain – all remaining (ground) actions exist identically in the unmodified domain. The goal remains unchanged as well. Therefore, a solution to the simplified problem is also one for the original problem. This is a simple pre-processing step that could, in theory, be used with any FOND planner. Further details on the object sampling and the hyperparameters chosen (i.e., how many objects to remove and how long to search for a solution) can be found in Appendix C of (Muise, McIlraith, and Beck 2023).

Force 1-safe Weak Plans

When a weak plan is found (cf. Case 5 of Algorithm 1), rather than immediately adding the newly found nodes to both CONTROLLER and REACHABLE, we confirm that no immediate extension leads to a deadend. This is a simple process that involves (1) computing every state reachable from the actions in the plan (which will eventually become open nodes in REACHABLE); (2) checking if any such states are deadends; (3) recording the relevant generalized deadends and FSAPs, and then re-running the weak planning procedure if a deadend is detected. We are guaranteed to find a different weak plan if step (3) is taken, since the weak planning procedure will avoid the deadend discovered in step (2) due to new FSAPs. This optimization can potentially lead to far fewer updates to the CONTROLLER and REACHABLE.

[1]We forgo introducing objects in the paper, as it is only this one contribution that makes use of them. Intuitively, actions are parameterized by objects, allowing us to represent the domain specification compactly. Cf. (Haslum et al. 2019) for more details.

Theoretical Properties

The non-deterministic planning techniques exploited in PR2 yield a number of interesting theoretical properties. In what follows, we provide the proof sketches for two of the most important results: the soundness and completeness of the PR2 planner.

Theorem 1 (Soundness)**.** *For a given FOND task T, if a strong cyclic plan* CONTROLLER *is produced by PR2, then it is a strong cyclic plan for T.*

Proof sketch. The soundness of the approach rests on the CONTROLLER representation, and what it means for $incumbent.is\_strong\_cyclic()$ to return true in Algorithm 1. This check amounts to inspecting the sc property of the SolStep that corresponds to the initial state, and Algorithms 2 + 3 combine to ensure that this initial SolStep is marked strong cyclic only if the full CONTROLLER is.

Note that in Algorithm 3, nodes that are marked on the final line will be those that (1) have all their successors defined (cf. the first for-loop) and (2) cannot reach some unmarked node (cf. the second for-loop). Because every node in CONTROLLER marked sc cannot reach an unmarked node, we can conclude that a node newly marked sc can always reach the goal under the assumption of fairness.

The final step is to consider the continued executability of actions used when following CONTROLLER. By using the combined regression in Algorithm 2, we are guaranteed that for any SolStep ss, executing $ss.a$ in a state s where $s \models ss.ps$, the outcome taken will lead to a state s' that corresponds to the successor SolStep. This invariant on the CONTROLLER that ties together neighbouring SolSteps is the same one that leads to the soundness of the FONDSAT and GRENDEL planners. □

Theorem 2 (Completeness)**.** *For a given FOND task T, if a strong cyclic plan exists, then PR2 will eventually find one.*

Proof sketch. Completeness intuitively follows from the eventual discovery of all required FSAPs. If REACHABLE leads to a place in the search space where no strong cyclic solution exists, then a deadend will be discovered and new FSAPs created. The outer-loop of Algorithm 1 will repeat as long as new FSAPs are discovered, and so we need only consider the final round (assuming the FSAPs found are correct and only finitely many may exist – a natural consequence of the finite nature of the domain).

The correctness of the identified FSAPs (cf. Section 15) is established inductively by the correctness of the deadends they lead to. If a deadend is correct, then any FSAP that covers an action leading to it is also correct. If all actions executable in a state are forbidden by correctly identified FSAPs, then that state is itself a deadend, and the process proceeds backwards.

In the final round, the FSAP avoidance will mean the search for weak plans will only be in a space where strong cyclic solutions exist (otherwise we would have a contradiction in this being the final round). Since weak plans will always be found to build the policy, every node reached will have some solution found and included. Because of the exhaustive nature of processing nodes in REACHABLE, eventually, a CONTROLLER will be computed such that it is detected as being strong cyclic. □

Evaluation

Our objective is to understand the performance of PR2 compared to other FOND planners in terms of coverage, solution size, and solve time. We implemented PR2 on top of the Fast Downward planning system (Helmert 2006) and used very little of the released code for PRP: specifically, much of the translation and scripts were re-used, as is the case with other FOND planners (e.g., MyND and FONDSAT use the same parsing mechanism). The code, benchmarks, and detailed analysis can be found at `mulab.ai/pr2`.

We compare against the state of the art in FOND planning: MyND (Mattmüller et al. 2010), FONDSAT (Geffner and Geffner 2018), PRP (Muise, McIlraith, and Beck 2012), and Paladinus (Pereira et al. 2022). We configured each planner to its best settings based on aggregate performance across all domains, including using a modern SAT solver for FONDSAT (improving its coverage by a fair margin). Planners were given 4Gb of memory and 60min to solve an instance, and evaluations were run on a PowerEdge C6420 machine running Ubuntu with an Intel 5218 2.3GHz processor.

To evaluate our planners, we collected all of the benchmarks employed for evaluation of the FOND planners listed above, representing a total of 18 domains. Across the 18 domains, there is a wide disparity in the number of instances, from 8 in the smallest (acrobatics) to 190 in the largest (faults-new). Consequently, we normalize the coverage on a per-domain basis to be a maximum of 1.

Planner Comparison

In Table 1 we show the normalized coverage across all domains and planners. PR2 performs at least as well, and often better, than every other planner in virtually every domain. The one exception is for blockworlds-new, where the problem size for a single instance grows to an extent that PR2 runs out of memory while PRP just barely does not (it is the largest instance PRP is capable of solving). Additionally, there are four instances in forest-new that PRP solves and PR2 does not (though, several other instances that PR2 solves and PRP does not). Not only does PR2 perform well on the older benchmarks where PRP was known to be state of the art, but it also handles every one of the new benchmark domains – islands, miner, tire-spiky, and tire-truck.

We should note that in the domain ‘doors’ for PRP (cf. (*) in Table 1), a bug in PRP led it to incorrectly declare the three simplest problems have no solution. Presumably, if this bug were fixed, the performance would increase by 0.2 to match PR2 and FONDSAT with a perfect score. We have removed the instances (7 from firstresponders-new and 3 from the original tireworld) for which no strong cyclic solutions exist; while PR2 correctly returns that no strong cyclic solutions exist for these instances, some other planners fail in unpredictable ways. We did not detect any further erroneous behaviour among the planners and domains.

We report on the time and solution size comparison in Figures 4a and 4b. PR2 consistently outperforms the other planners on both measures. However, PRP produces smaller solutions as the problem size grows (the PRP planner uses a solution representation not shared by any of the other planners), and is also faster in a handful of problems. Finally, in Figure 4c, we show the normalized coverage over time to give a sense of how quickly the solutions are found. After roughly 0.5 seconds, PR2 surpasses PRP in terms of coverage and remains dominant.

domain (size)	pr2	prp	fsat	pala	mynd
acrobatics (8)	**1.00**	**1.00**	0.50	**1.00**	**1.00**
beam-walk (11)	**1.00**	**1.00**	0.27	0.82	**1.00**
bw-new (50)	0.82	**0.84**	0.16	0.36	0.42
chain (10)	**1.00**	**1.00**	0.10	**1.00**	**1.00**
earth-obs (40)	**1.00**	**1.00**	0.17	0.65	0.78
elevators (15)	**1.00**	**1.00**	0.47	0.53	0.93
faults-new (190)	**1.00**	**1.00**	0.04	0.12	0.84
first-new (88)	**0.99**	**0.99**	0.06	0.06	0.09
forest-new (100)	**0.94**	0.88	0.10	0.18	0.16
tidyup-mdp (10)	**1.00**	0.00	0.10	0.40	**1.00**
tire (12)	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
tri-tire (40)	**1.00**	**1.00**	0.10	0.20	0.17
zeno (15)	**1.00**	**1.00**	0.33	0.53	0.60
doors (15)	**1.00**	(*) 0.80	**1.00**	0.93	0.73
islands (60)	**1.00**	0.52	0.95	**1.00**	0.22
miner (51)	**1.00**	0.25	0.98	**1.00**	0.00
tire-spiky (11)	**1.00**	0.09	0.91	0.91	0.18
tire-truck (74)	**1.00**	0.28	0.99	0.65	0.20
TOTAL (800)	**17.75**	13.65	8.23	11.34	10.33

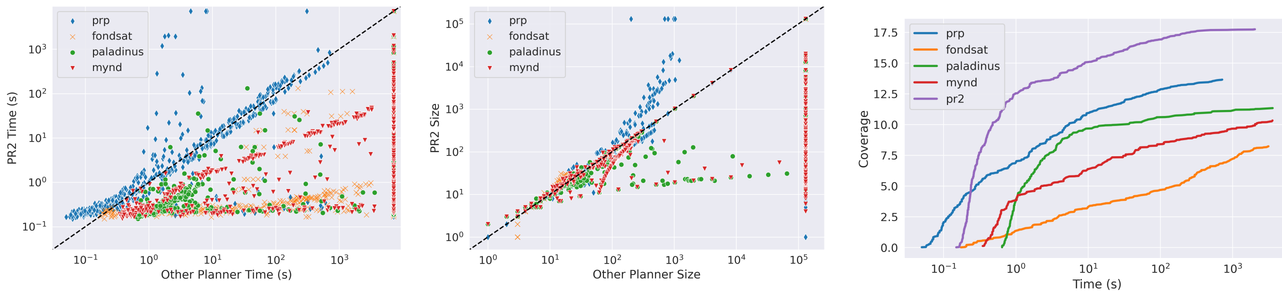
Table 1: Normalized Coverage for all planners and domains.

Ablation Studies

We briefly report on the performance impact of several of the features of PR2. Corresponding to Sections 12 - 15, we disabled each feature and measured the impact on normalized coverage in the PR2 planner:

(section) **Disabled Feature**	**Drop in coverage**
(12) Full Strong-Cyclic Marking	-0.757
(15) Poisoning	-0.743
(15) FSAP Heuristic	-2.839
(15) Object Sampling	-0.743
(15) Forcing 1-safe Plans	-1.570

First, we note that no instance unsolved by PR2 was solved by disabling one of the features. Thus, their presence never hurts the planner in terms of coverage. To place the reduction of coverage in perspective, a drop of 1.0 indicates (roughly) a full domain's worth of instances that cannot be solved. In the supplementary material, we provide a full table similar to Table 1, but the key takeaways are: (1) removing any one of the features drops the coverage in the *tireworld-truck* domain substantially (from nearly 1.0 down to 0.2-0.3); (2) the drop in coverage for poisoning, object sampling, and full strong-cyclic marking is entirely due to this one domain; (3) disabling the FSAP heuristic and forcing 1-safe plans reduced performance greatly in the *triangle tireworld* domain; and (4) disabling the FSAP heuristic degraded performance in the *islands* and *miner* domains.



(a) Time comparison. (b) Size comparison. (c) Coverage over time.

Figure 4: Comparison of time (left), size (middle), and coverage (right). Time: a log-log plot showing the PR2 runtime against the other planners, over all problems. Anything below the x=y line is an improvement. Size: similar to time, but measured in the solution size produced by each planner. Coverage: "Survival plot" that shows the normalized coverage achieved as a function of time for each of the planners. Note the log scaling on the x-axis.

We can conclude that every one of the introduced features provides a net benefit to the planner, which (at times) is crucial to achieving peak performance. Moreover, the FSAP-aware heuristic appears to have the biggest impact on the results of the PR2 planner. We note that removing any single feature leaves PR2 with a planner that still outperforms all previous FOND planners.

Related Work

The closest related work is the FOND planner, PRP (Muise, McIlraith, and Beck 2012). PR2 falls under the same category of FOND planner approach as PRP, along with FIP and NDP (Fu et al. 2011; Kuter et al. 2008). The strategy PR2 uses to leverage regression is similar to PRP, including the handling of conditional effects (Muise, McIlraith, and Belle 2014), while many of the generalization methods are shared across a broader set of planners (e.g., the deadend relaxation introduced by SixthSense (Kolobov, Mausam, and Weld 2010)). Concretely, very little implementation is shared between PRP and PR2. Beyond the implementation, the solution representation (interplay between CONTROLLER and REACHABLE) and the algorithms surrounding them are also novel. From the custom FSAP-aware heuristic to the procedures for strengthening and marking CONTROLLER, all techniques detailed in Sections 19 - 15 are novel.

The CONTROLLER representation is shared by the FONDSAT and GRENDEL planners (Geffner and Geffner 2018; Ramírez and Sardiña 2014). Having a controller with regressed conditions associated with the nodes strikes a powerful balance between representation size and generality. Nonetheless, the three planners (PR2, FONDSAT, and GRENDEL) take strikingly different approaches to computing the solution. Whereas PR2 uses the replanning approach pioneered by NDP and extended by FIP and PRP, FONDSAT uses a SAT encoding of a structure very close to the CONTROLLER, and GRENDEL uses backwards search through the space of controllers. Finally, there is a similarity between the fixed-point regression performed by PR2 on the CONTROLLER and GRENDEL's approach to re-establish the conditions needed for strong-cyclic solutions.

The final category of FOND solvers is the policy-space search of MyND and Paladinus (Mattmüller et al. 2010; Pereira et al. 2022). By many measures, Paladinus represents the state of the art for this class of solution strategy. Particularly on the newer domains, it outperforms MyND by a large margin. These planners operate by searching through the space of reachable policies, incrementally adding to them until a strong cyclic policy is found.

Concluding Remarks

The release of PRP in 2012 marked a significant jump in our ability to solve FOND problems, opening the door to new application areas, and inspiring the development of contingent (Muise, Belle, and McIlraith 2014), probabilistic (Camacho, Muise, and McIlraith 2016) and LTL-FOND (Camacho et al. 2017) variants. We present a new FOND planner, PR2, that similarly advances the state of the art in FOND planning. The PR2 planner was designed from the ground up using many of the best-known techniques for FOND planning. We introduced a suite of novel techniques that help PR2 achieve its high performance, and the planner itself is amenable to extension and further research.

A promising direction for such work is in the further development of novel heuristics for FOND planning. The FSAP-aware heuristic proposed here (cf. Section 15) is a significant improvement over previous methods, but it is only one idea in the arena of FOND-aware heuristics. From a more theoretical viewpoint, there is a growing recognition that the solution representation of a FOND problem plays a critical role in computing and executing plans (Armstrong and Muise 2023; Messa and Pereira 2023). The merits of the representational power of the CONTROLLER as compared to the partial state-action pairs of PRP or the complete state-action mappings found in other FOND planners, deserves further study. Finally, our results show that FOND planning has achieved a notable leap in performance. New benchmark problems, perhaps from adjacent fields such as reactive synthesis (De Giacomo and Vardi 2015; Camacho et al. 2017), should be assembled in service of a modern suite of challenge domains, afforded by these new capabilities.

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Vector Institute for Artificial Intelligence and the Canada CIFAR AI Chair program.

References

Andrés, I.; de Barros, L. N.; and Delgado, K. V. 2020. Human-aware Contingent Planning. *Fundam. Informaticae*, 174(1): 63–81.

Armstrong, V.; and Muise, C. 2023. The Generalizability of FOND Solutions in Uncertain Environments. In *2023 Workshop on Integrated Acting, Planning and Execution (IntEx) at the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*.

Bonet, B.; De Giacomo, G.; Geffner, H.; Patrizi, F.; and Rubin, S. 2020. High-level Programming via Generalized Planning and LTL Synthesis. In Calvanese, D.; Erdem, E.; and Thielscher, M., eds., *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, 152–161.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Camacho, A.; Muise, C. J.; and McIlraith, S. A. 2016. From FOND to Robust Probabilistic Planning: Computing Compact Policies that Bypass Avoidable Deadends. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, (ICAPS)*, 65–69.

Camacho, A.; Triantafillou, E.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 3716–3724. AAAI Press.

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

Daniele, M.; Traverso, P.; and Vardi, M. Y. 1999. Strong Cyclic Planning Revisited. In *Recent Advances in AI Planning, Proc. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Computer Science*, 35–48. Springer.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 1558–1564.

Fritz, C.; and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *17th International Conference on Automated Planning and Scheduling*, 144–151.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I.-L. 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *22nd International Joint Conference On Artificial Intelligence*, 1949–1954.

Geffner, T.; and Geffner, H. 2018. Compact Policies for Fully Observable Non-Deterministic Planning as SAT. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M. T. J., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, 88–96. AAAI Press.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool. ISBN 9781627058759.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J. 2001. FF: The Fast-Forward Planning System. *AI Mag.*, 22(3): 57–62.

Illanes, L.; and McIlraith, S. A. 2019. Generalized Planning via Abstraction: Arbitrary Numbers of Objects. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, 7610–7618.

Kolobov, A.; Mausam; and Weld, D. S. 2010. SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs. In Fox, M.; and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Kuter, U.; Nau, D.; Reisner, E.; and Goldman, R. P. 2008. Using classical planners to solve nondeterministic planning problems. In *18th International Conference on Automated Planning and Scheduling*, 190–197.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *20th International Conference on Automated Planning and Scheduling*, 105–112.

Messa, F.; and Pereira, A. G. 2023. A Best-First Search Algorithm for FOND Planning and Heuristic Functions to Optimize Decompressed Solution Size. In *The 33rd International Conference on Automated Planning and Scheduling*, The 33rd International Conference on Automated Planning and Scheduling.

Muise, C.; Chakraborti, T.; Agarwal, S.; Bajgar, O.; Chaudhary, A.; Lastras-Montano, L. A.; Ondrej, J.; Vodolan, M.; and Wiecha, C. 2019. Planning for Goal-Oriented Dialogue Systems. arXiv:1910.08137.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *The 22nd International Conference on Automated Planning and Scheduling*, The 22nd International Conference on Automated Planning and Scheduling.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2023. PRP Rebooted: Advancing the State of the Art in FOND Planning. arXiv:2312.11675.

Muise, C.; McIlraith, S. A.; and Belle, V. 2014. Non-Deterministic Planning With Conditional Effects. In *The 24th International Conference on Automated Planning and Scheduling*.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing Contingent Plans via Fully Observable Non-Deterministic Planning. In *Proceedings of the Twenty-*

Eighth AAAI Conference on Artificial Intelligence (AAAI), 2322–2329.

Pereira, R. F.; Pereira, A. G.; Messa, F.; and De Giacomo, G. 2022. Iterative Depth-First Search for Fully Observable Non-Deterministic Planning. In *The 32nd International Conference on Automated Planning and Scheduling*.

Ramírez, M.; and Sardiña, S. 2014. Directed Fixed-Point Regression-Based Planning for Non-Deterministic Domains. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*.

Reiter, R. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. The MIT Press. ISBN 0262182181.

Waldinger, R. 1977. Achieving several goals simultaneously. *Machine Intelligence*, 8: 94–136.