Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

Agent Planning Programs as Non-deterministic Planning under Fairness

Nitin Yadav[1], **Sebastian Sardiña**[2], **Hector Geffner**[3]

[1]The University of Melbourne, AUSTRALIA
[2]RMIT University, AUSTRALIA
[3]RWTH Aachen University, GERMANY
nitin.yadav@unimelb.edu.au, sebastian.sardina@rmit.edu.au, hector.geffner@ml.rwth-aachen.de

Abstract

We propose an approach for solving Agent Planning Programs (APP) based on a reduction to (strong-cyclic) Fully Observable Non-Deterministic (FOND) planning. APPs represent a middle-ground between automated planning and agent-oriented programming, in which the space of possible agent behavior is "programmed" as a network of declarative goals wrt an underlying planning domain. Each transition in an APP represents a local planning problem that may need to be addressed by the agent executing the APP. APPs allow the specification of continuous goal-driven behavior in which the "next" goal is externally chosen, thus going beyond one-shot planning. Two methods have been proposed for solving APPs: a principled but inefficient LTL reactive synthesis technique; and a more efficient but arguably ad-hoc approach that relies on multiple "local" classical planning and meta-level backtracking. We demonstrate how APPs can be solved in a principled manner by developing an elegant reduction to non-deterministic planning under fairness assumptions, and show experimentally with existing FOND solvers its practical value. We also provide a new solution concept that is simpler and closer to mainstream planning than the existing one.

Introduction

We present a new approach to solving *Agent Planning Programs* (APPs) via Fully Observable Non-Deterministic (FOND) planning. Compared with previous techniques, our approach is conceptually simpler and elegant, performs well computationally, and accommodates non-deterministic planning tasks. Additionally, our reduction provides further evidence of the expressiveness of FOND planning.

Agent Planning Programs (APPs) is, in a nutshell, an approach to represent and synthesize complex behaviors based on *sequences of goals*, *external (user) decisions*, and *repetition of tasks* for continuous long-term operation (De Giacomo, Patrizi, and Sardina 2010; De Giacomo et al. 2016). Technically, APPs represent a middle ground between first-principled planning and agent-oriented programming, where the space of possible behavior is "programmed" via a network of declarative goals. From a planning perspective, an APP can be seen as a generalization of AI planning, in which different planning problems are put together in "stages", and where the "next" goal to be achieved is determined at runtime by the agent running the agent program. From an agent-oriented standpoint, APPs realize the idea of programming an agent fully based on *declarative* goals (de Boer et al. 2007; Hübner, Bordini, and Wooldridge 2006; van Riemsdijk, Dastani, and Meyer 2005; Sardina and Padgham 2011; Porfirio, Roberts, and Hiatt 2024), without the need to specify how those goals are to be realized at a specific time. The advantages of using APPs are two-fold; first they enable control specification at a high-level, while accommodating external decision-making (e.g., by a human expert), and second they accommodate non-terminating operations.

Solving APPs is hard; it has been proven to be EXPTIME-complete even under deterministic planning domains (De Giacomo et al. 2016). Two effective techniques have so far been developed to solve APPs. The first one is based on reactive synthesis, where a (carefully designed) GR(1) specification (Bloem et al. 2012) is used to synthesize a controller that can achieve the goals in the APP (De Giacomo, Patrizi, and Sardina 2010). The second one is based on calling a classical planner repeatedly to solve each *local* planning task in the network, with the resulting plans "stitched" together to form an overall solution for the APP (Gerevini, Patrizi, and Saetti 2011; De Giacomo et al. 2016). While this meta-planning based approach outperforms the synthesis one, it relies on ad-hoc optimizations to "glue" local plans together, uses complex backtracking at the meta-level, and is restricted to classical planning domains.

We propose here a reduction of APPs to FOND planning under (strong) fairness constraints and strong-cyclic solutions. The reduction is elegant and linear in size, and leverages on state-of-the-art FOND planners. It can also directly handle non-deterministic planning domains, which is a significant advantage over the meta-search based approach.

Besides providing the technical details of the reduction, together with correctness results, we provide experimental results showing that the proposed approach is competitive, faring well against the state-of-the-art meta-planning based approach (De Giacomo et al. 2016). Importantly, we also develop an alternative solution concept definition that is conceptually simpler (and still equivalent!) than the existing one in the literature (based on non-standard simulation relations) and closer to planning solution concepts.

FOND Planning

Fully-observable non-deterministic planning (Daniele, Traverso, and Vardi 2000; Geffner and Bonet 2013) is an elaboration of classical planning in which actions can have non-deterministic effects under strict uncertainty. All specified effects are meant to have non-zero probability, and the actual one ensuing is observable at run-time.

A ***FOND planning domain*** is a pair $\mathcal{D} = \langle V, O \rangle$ consisting of a set of Boolean state variables V and an operator set O. A ***state*** $s \in 2^V$ is the set of propositional variables that are true in the state. We use S to denote the set of all states and $\bar{l}$ to denote the complement of literal l.

An ***operator*** (or action) is a tuple $\langle o, \mathrm{Pre}_o, \mathrm{Eff}_o \rangle$, where o is a unique name, and Pre_o and Eff_o describe the *preconditions* and *effects* of operator o as a boolean combination of literals from V.

In particular, disjunctions in effects encode non-deterministic effects, that is, effects under the environment's control. Under full-observability, though, the actual effects that ensue are known after execution. While, in principle, any (satisfiable) boolean formula could be used as an effect, the PDDL language restricts to CNF-type effects via the `(oneof E1 ... En)` construct (Gerevini, Bonet, and Givan 2006) which specifies that *one* of the E_i effects ensues non-deterministically, where each E_i is a deterministic effect (conjunction of literals).[1]

A ***FOND planning problem*** is a tuple $\mathcal{P} = \langle \mathcal{D}, s_I, G \rangle$, where $\mathcal{D}$ is a FOND planning domain, $s_I \in S$ is the initial state and $G \subseteq 2^V$ is the goal conditions to be achieved. Generally speaking, a solution is a policy (or plan) that when executed in domain $\mathcal{D}$ from state s_I will bring about a state s_g such that $s_g \models G$. A ***policy*** is a partial function $\pi : S \mapsto O$ that maps state $s \in S$ to a domain action $\pi(s)$. We use $\mathit{Reach}_{\mathcal{D}}(\pi, s)$ to denote the set of states potentially reachable by executing π form state s. We observe that whereas in a deterministic domain, all reachable states come from the same, linear, execution trace; under a non-deterministic domain, states may be reachable through different execution traces. A ***terminal*** state is one where π is undefined—no action is prescribed. When an execution traces reaches a terminate state, the execution stops and the trace cannot be further extended. We use $\mathit{End}_{\mathcal{D}}(\pi, s)$ to denote the set of terminal states reachable by executing π form state s, that is, all the final states in full executions of $\mathcal{D}$ from s.

In the context of non-determinism, several solution concepts are available (Cimatti et al. 2003). A policy is a ***weak solution*** for a FOND task $\mathcal{P}$ if there exists $s \in \mathit{End}_{\mathcal{D}}(\pi, s_I)$ such that $s \models G$. In turn, a policy is a ***strong-cyclic solution*** for a FOND task $\mathcal{P}$ if for every $s \in \mathit{Reach}_{\mathcal{D}}(\pi, s_I)$, there exists $s' \in \mathit{End}_{\mathcal{D}}(\pi, s)$ such that $s' \models G$. In words, every finite execution of π can be further extended to reach a terminating state where the goal holds. This notion is, arguably, the most used one, as it embodies natural and effective enough behavior in the context of plausible *fairness* assumptions on the environment (Sardina and D'Ippolito 2015): all possible outcomes of an action have probability greater than zero. A more demanding requirement is that of ***strong policy solutions*** which, in addition, requires no execution to pass along a same state twice: no cycles are permitted in executions and the goal is obtained in a *bounded* number of steps. Strong solutions are preferred when non-determinism is adversarial (that is, some possible state outcomes can be skipped forever); while strong cyclic solutions are suitable for fair non-determinism (that is, a possible state outcome cannot be skipped forever).

[1] This formalization of (non-deterministic) actions corresponds to 1ND Normal Form (Rintanen 2003) with no nested conditional (deterministic) effects, and to the usual `(oneof e1 ... en)` clauses in PDDL (Gerevini, Bonet, and Givan 2006) to model non-deterministic actions.

Agent Planning Programs

Agent Planning programs (De Giacomo, Patrizi, and Sardina 2010; De Giacomo et al. 2016) generalize planning problems, like the ones above, by accommodating multiple "interactive" goals in temporally structured manner—as a "network" of goals that are "requested" by the external user or agent running the program. So, instead of one-shot task, APPs can model *non-terminating* behavior that arises from achieving several goals in sequence continuously. In addition, which goals and in which order are determined by factors external to the system, that is, by the "user" of the planning program. Essentially, APPs are high-level representations of interactive, goal-oriented, extended behavior in a given domain. The APPs are also attractive in that they bring ideas and techniques that are normal in declarative goal agent-oriented programming (de Boer et al. 2007; Hübner, Bordini, and Wooldridge 2006; van Riemsdijk, Dastani, and Meyer 2005; Sardina and Padgham 2011) to the realms of automated planning.

Formally, APPs are like planning problems but where the goal is replaced by a network of goals—the planning program—with nodes representing decision points and transitions representing possible guarded goals.

Definition 1. An ***agent planning program problem*** is a tuple $\mathcal{A} = \langle \mathcal{D}, s_I, P, q_I \rangle$, where $\mathcal{D} = \langle V, O \rangle$ is a planning domain, s_I is the initial domain state, $P = \langle Q, T \rangle$ is the program graph with transition edges of the form $q \xrightarrow{\gamma:\phi} q'$ where γ and ϕ are formulas over V, and $q_I \in Q$ the initial node of the program. ■

An edge transition $q \xrightarrow{\gamma:\phi} q'$ in P states that ϕ is a potential goal to be requested for achievement when the planning program is in state q, provided that condition γ holds.[2] Ultimately, an active (i.e., one where the guard holds) legal transition $q \xrightarrow{\gamma:\phi} q'$ in an APP corresponds to a typical planning problem over the same shared domain $\mathcal{D}$, formula ϕ as the goal, and initial state being the world state in which the transition is being executed form.

Figure 1 depicts a simplified APP for an academic everyday-life routine with only achievement goal transitions (example taken from (De Giacomo, Patrizi, and Sardina 2010). Note that the APP only mentions (goal) conditions

[2] In its original formulation, an edge also included a maintenance goal; for simplicity we assume they are complied away.

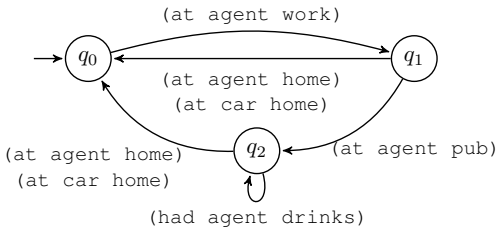

Figure 1: An APP for the daily routine of an academic, meant to be solved relative to a given planning domain.

over the domain, not actions. What actions are available and how they affect those conditions will be given by some planning domain $\mathcal{D}$ (not shown here). Also observe the difference between program states and world states; the former corresponds to the "program counter" of the APP, while the latter is the usual state of the environment (specified by domain $\mathcal{D}$) the APP is running over.

At any point in time, the system is in a certain program state of the APP, and the *user* (e.g., the academic) issues a transition goal request, among the possible ones in such a state. In the example, the academic can at the start only aim to be at work (transition from q_0 to q_1). The system then *deploys a plan* that achieves the requested goal, and the APP then evolves to its next program state; in our case node q_1. After that, the user is able to issue the next goal request. Now, from program state q_1, the academic may request to return home (transition p_1 to p_0) *or* alternatively aim to go to the pub with colleagues (transition p_1 to p_2). Once again, depending on the goal requested, the system is meant to deploy an adequate plan, always. As one can see, each goal decision choice is *decided at run-time*, by an external "user" of the APP (e.g., a human expert or another software). Importantly, the system ought to be prepared to meet *any* legal sequence of goals. Observe also that these sequences are unbounded as the APP is cyclic and encodes repetition of tasks, thus accounting for continuous behavior. In the example, the academic always comes back home and, importantly, with the car (if it has been used).

Solving Agent Planning Programs

The original solution concept for APPs is based on the notion of *realization* (De Giacomo, Patrizi, and Sardina 2010), which is a strategy that guarantees the successful execution of the program, no matter the sequence of goals that are requested. Our first contribution amounts to an alternative, but provably equivalent, solution concept that is significantly simpler and technically directly based on standard planning solution concepts.

Definition 2 (APP solution). Let $\mathcal{A} = \langle \mathcal{D}, s_I, P, q_I \rangle$ be an APP problem, where $P = \langle Q, T \rangle$. A pair of functions $\sigma = \langle \sigma_S, \sigma_\pi \rangle$, where $\sigma_S : Q \mapsto 2^S$ and $\sigma_\pi : T \mapsto (S \mapsto O)$, is an *APP-solution* iff

1. $s_I \in \sigma_S(q_I)$; and
2. for every state $s \in \sigma_S(q)$ and transition $q \xrightarrow{\gamma:\phi} q'$ in P such that $s \models \gamma$, policy $\pi' = \sigma_\pi(q \xrightarrow{\gamma:\phi} q')$ is a solution[3] for planning task $\langle \mathcal{D}, s, \phi \rangle$ and $End_{\mathcal{D}}(\pi', s) \subseteq \sigma_S(q')$. ■

Intuitively, σ_S specifies the states the domain could be at a given program state, whereas σ_π specifies a policy that is guarantee to work on all those states. Note this definition is independent of the specific solution concept (such as strong or strong-cyclic plans) used for planning problems.

We show that Definition 2 is equivalent to the notion of APP realization (as defined by De Giacomo, Patrizi, and Sardina (2010)). Technically, a ***realization*** for an APP is a partial function $\Omega : S \times T \mapsto (S \mapsto O)$ that yields the domain policy to be carried out when a given transition $q \xrightarrow{\gamma:\phi} q'$ is requested in a particular domain state $s \in S$. A ***realization***—a solution—must induce a special simulation relation R_Ω, referred as $PLAN$-simulation, between planning program states and domain states. Formally, ***plan simulation*** (De Giacomo et al. 2016) is a relation $R_\Omega \subseteq Q \times 2^V$ such that $\langle q, s \rangle \in R_\Omega$ implies that, for every transition $q \xrightarrow{\gamma:\phi} q'$ in P and $s \models \gamma$, there exists a planning solution π such that: (i) plan π achieves goal ϕ from s; and (ii) for all $s' \in End_{\mathcal{D}}(\pi, s)$, it is the case that $\langle q', s' \rangle \in R_\Omega$, that is, the plan π preserves relation R. Intuitively, $\langle q, s \rangle \in R_\Omega$ states that, for each possible transition request arising from program state q, function Ω prescribes a policy that is guarantee to achieve the transition's goal from state s and leave the system state—program state and domain state—within relation R_Ω again. A node $q \in Q$ is said to be $PLAN$-simulated by a domain state s, denoted by $q \preceq_{PLAN} s$, if there exists a $PLAN$ simulation relation R_Ω such that $\langle q, s \rangle \in R_\Omega$.

Theorem 1. *There exists a APP solution for APP problem* $\mathcal{A} = \langle \mathcal{D}, s_I, P, q_I \rangle$ iff $q_I \preceq_{PLAN} s_I$.

Proof (Sketch). Take R to be a $PLAN$ simulation relation such that $\langle q_I, s_I \rangle \in R$. Let $S_{q,\gamma} = \{s \mid \langle q, s \rangle \in R, s \models \gamma, q \xrightarrow{\gamma:\phi} q' \in P\} = \{s_1, \ldots, s_n\}$, for $n \geq 0$. Wlog, for each state $s_i \in S_{q,\gamma}$, let π_i denote a plan that achieves goal ϕ from s_i and preserves relation R—we know it exists by assumption. Note these policies address the same transition request but for different (initial) domain states. We combine them all into a single policy π' as follows: $\pi'(s) = \pi_i$, if $\pi_j(s) = \bot$, for all $j < i$.
The policy π' is well-defined since the domain states are fully observable. We construct APP solution $\langle \sigma_S, \sigma_\pi \rangle$ from R as per: $\sigma_S(q) = \{s | \langle q, s \rangle \in R\}$, and $\sigma_\pi(q \xrightarrow{\gamma:\phi} q') = \pi'$.
One can show that for this combined policy π' the following properties hold. Given a state $s \in \sigma_S(q)$ and a transition $q \xrightarrow{\gamma:\phi} q'$ in P:

$$Reach_D(\pi', s) \subseteq \bigcup_{i \leq |S_{q,\gamma}|} Reach_D(\pi_i, s_i) \quad (1)$$

$$End_D(\pi', s) \subseteq \bigcup_{i \leq |S_{q,\gamma}|} End_D(\pi_i, s_i) \quad (2)$$

[3] Weak, strong, or strong-cyclic.

To show the validity of $\langle \sigma_S, \sigma_\pi \rangle$ consider the following. First, since $\langle q_I, s_I \rangle \in R$ we have that $s_I \in \sigma_S(q_I)$. Second, we show that for a transition $q \xrightarrow{\gamma:\phi} q'$ in P, policy π' achieves the solution ϕ for all $s \in S_{q,\gamma}$. For strong-cyclic solution, take $s \in S_{q,\gamma}$ such that $s' \in \mathit{Reach}_D(\pi', s)$. We are to show that goal ϕ is "live" at s', that is, that there exists a $s'' \in \mathit{End}_D(\pi', s')$ such that $s'' \models \phi$. Since each π_i, where $i \leq |S_{q,\gamma}|$, is a solution, it follows that for each $s' \in \mathit{Reach}_D(\pi_i, s_i)$ there exists a $s'' \in \mathit{End}_D(\pi_i, s')$ such that $s'' \models \phi$. The combined policy π' prescribes an action to a state s' as per the smallest i such that $\pi_i(s) \neq \bot$, where $i \leq |S_{q,\gamma}|$. Following the reasoning behind the second property, it is the case that $\mathit{End}_D(\pi', s') \subseteq \bigcup_{i \leq j} \mathit{End}_D(\pi_i, s')$ where π_j is the policy that is mimicked by π' at state s'. This combined with $Reach_D(\pi', s) \subseteq \bigcup_{i \leq |S_{q,\gamma}|} Reach_D(\pi_i, s_i)$ is sufficient to show that for each $s' \in Reach_D(\pi', s)$ there exists a state $s'' \in End_D(\pi', s')$ such that $s'' \models \phi$.

Finally, from the definition of $PLAN$ simulation relation we have that for all $s' \in End_{\mathcal{D}}(\pi_i, s_i)$, where $i \leq |S_{q,\gamma}|$, it is the case that $\langle q', s' \rangle \in R$. Since $End_D(\pi', s) \subseteq \bigcup_{i \leq |S_{q,\gamma}|} End_D(\pi_i, s_i)$, we have that $End_{\mathcal{D}}(\pi', s) \subseteq \sigma_S(q')$. Proof for weak and strong solution uses similar arguments.

For the other direction, assume there exists a APP solution $\langle \sigma_S, \sigma_\pi \rangle$. We construct the plan simulation as $R = \{\langle q, s \rangle | \sigma_S(q) = s\}$. Hence, $\langle q_I, s_I \rangle \in R$ and by construction, for every $\langle q, s \rangle \in R$, and transition $q \xrightarrow{\gamma:\phi} q'$ in P, plan $\sigma_\pi(q \xrightarrow{\gamma:\phi} q')$ achieves goal ϕ from state s, where $s \models \gamma$, and for all states $s' \in \mathit{End}_{\mathcal{D}}(\sigma_\pi(q \xrightarrow{\gamma:\phi} q'), s)$ it is the case that $\langle q', s' \rangle \in R$. Thus we get, $q_I \preceq_{PLAN} q s_I$. □

Note that policy $\sigma_\pi(q \xrightarrow{\gamma:\phi} q')$ is not *any* solution to the "local" planning problems induced by the transition; it is one that ought to be compatible with all policies solving the induced planning problems in the next program state (and so on). Thus, σ is *not* simply solving multiple (local) planning problems in isolation.

Two approaches have been developed to check (and compute) existence of realizations. The first approach was proposed in (De Giacomo, Patrizi, and Sardina 2010) and resorts to reactive synthesis techniques for GR(1) specification (Bloem et al. 2012). While very powerful in doing so, it outputs *universal* solutions containing all possible realizations and despite being optimal wrt computational complexity (i.e., EXPTIME), it turns out to be computationally demanding, as shown in (De Giacomo et al. 2016).

The second approach was first proposed by Gerevini, Patrizi, and Saetti (2011) and it involves iteratively solving each local planning task encoded in the APP, and "gluing" the local solution plans together. To do so, a combination of preference-based planning and backtracking at the meta-level is used. While it has been shown to scale up better (De Giacomo et al. 2016), it results in an overall ad-hoc algorithm, can only handle deterministic domains (as it uses classical planners to solve each transition problem), and relies on an incomplete underlying classical planner (thus providing no completeness guarantees).

In the next section, we develop a third strategy by fully leveraging on existing planning technology, but this time on strong-cyclic FOND planning.

Planning Programs via FOND Planning

In this section, we show how to reduce an arbitrary APP to a FOND planning task. The advantages are twofold. First, it provides a simple semantics of APP that demonstrates that APP is no more expressive than FOND planning. The reduction to be developed below is significantly simpler—syntactically and conceptually—than the one provided in (De Giacomo, Patrizi, and Sardina 2010) to GR(1) synthesis. Second, it will allow us to use existing, and upcoming, off-the-shelf FOND planners, which will prove as effective as the more complex ad-hoc solution techniques (discussed later in empirical evaluation).

Consider an APP $\mathcal{A} = \langle \mathcal{D}, s_I, P, q_I \rangle$, with $P = \langle Q, T \rangle$. In what follows, we define its corresponding FOND planning problem $\mathcal{P}_\mathcal{A} = \langle \mathcal{D}^+, s_I^+, G^+ \rangle$. For simplicity, we first restrict to the cases in which all transition guards are true (i.e., of the form $q \xrightarrow{\mathsf{true}} q'$) and, hence, always enabled.

Domain variables. The *set of propositional variables* V^+ of $\mathcal{D}_\mathcal{A}$ is obtained by extending the set of variables V in $\mathcal{D}$'s domains with the following additional variables basically encoding the program goal graph P:

- q: for each node q in P, that will be used to keep a track of the current state of the APP;
- next_q^ϕ: for each transition of the form $q' \xrightarrow{\phi} q$ in P, that will be used to represent the request of a goal ϕ to evolve to planning program state q;
- q_0 and $\mathit{next}_{q_I}^{\mathsf{true}}$: as a dummy initial program state and request; and
- *goal*: as a distinguished "dummy" goal predicate.

When s^+ is a state in the extended language (over variables V^+), we shall use s to denote its projection over the variables in V—the corresponding domain state over $\mathcal{D}$.

Initial state & goal condition. The *initial state* and *goal* condition of $\mathcal{P}_\mathcal{A}$ are straightforward:

$$s_I^+ = s_I \wedge q_0 \wedge \mathit{next}_{q_I}^{\mathsf{true}};$$
$$G^+ = \mathit{goal}.$$

That is, the initial state encodes the original initial domain state in the APP together with the dummy proposition q_0 used to signal the "start" state of the APP program graph itself. As in (Patrizi, Lipovetzky, and Geffner 2013), we make the goal of $P_\mathcal{A}$ to be the dummy goal predicate *goal*, inserted as an additional possible outcome of the actions that can achieve a transition's goals of the APP.

Domain operators. The set of operators in $\mathcal{D}_\mathcal{A}$ is obtained by extending the set of domain operators in $\mathcal{D}$ (whose preconditions are extended with $\neg q_0$) to model the execution of the planning graph P. Intuitively, every transition in the planning program will be modeled with a distinguished operator encoding the satisfaction of the goal embodied in the transition. The uncertain nature of the planning program's execution—the goal-transition requests are

not controllable—is encoded via the non-deterministic effects of these operators.

Concretely, for every transition edge $q \xrightarrow{\phi} q'$ in P, as well for a "dummy" transition $q_0 \xrightarrow{true} q_I$, a *non-deterministic* operator $t^{\phi}_{q,q'}$ is included to represent the *fulfillment* of the corresponding request transition:

- $\mathrm{Pre}_{t^{\phi}_{q,q'}} = q \wedge \mathit{next}^{\phi}_{q'} \wedge \phi$; and
- $\mathrm{Eff}_{t^{\phi}_{q,q'}} = \neg q \wedge q' \wedge \big(\bigvee_{q' \xrightarrow{\phi'} q'' \text{ in } P} (\neg \mathit{next}^{\phi}_{q'} \wedge \mathit{next}^{\phi'}_{q''}) \vee \mathit{goal} \big)$.

These operators, one per "request" transition $q \xrightarrow{\phi} q'$ in the APP, constitute the core of the reduction. For an operator $t^{\phi}_{q,q'}$ to be executable, the transition $q \xrightarrow{\phi} q'$ must be "pending" and its goal ϕ must have been achieved. The effects of fulfilling the request include transitioning the APP from its current state q to q', and *no-deterministically* simulating the possible next requests at program state q', or alternatively just achieving the distinguished *goal* atom.

The key idea is that whenever the non-deterministic effect encoding the next request transition $q' \xrightarrow{\phi'} q''$ ensues, the only way to (potentially) achieve the goal of the FOND task (i.e., atom "*goal* ") is to execute the fulfillment operator for such transition, namely, operator $t^{\phi}_{q',q''}$. Due to its precondition, this would require the execution of a plan bringing about ϕ, the goal of the request.

This concludes the reduction from an APP $\mathcal{A}$ to its corresponding FOND $\mathcal{P}_{\mathcal{A}}$. It is straightforward to see that the reduction is linear in the number of states of the planning goal graph. Indeed, the number of variables in the underlying planning domain is increased by $|Q| + |T| + 2$ and the number of operators by $|T| + 1$.

Next, we show the correctness of our reduction for APPs, when used under strong-cyclic underlying solution concept in planning transitions (c.f., Definition 2).[4] In particular, the reduction is able to capture exactly all the solutions for agent planning programs. Recall that we use s^+ to denote states in the extended planing domain $\mathcal{D}^+$ and s to denote its corresponding projection over the language of $\mathcal{D}$.

Theorem 2 (Soundness). *Let $\mathcal{A}$ be an APP and $\mathcal{P}_{\mathcal{A}}$ is corresponding FOND planning task encoding as described above. If π is a strong-cyclic solution for $\mathcal{P}_{\mathcal{A}}$, then $\sigma = (\sigma_S, \sigma_\pi)$ is an APP-solution for APP $\mathcal{A}$ where:*

- $\sigma_S(q) = \{s \mid s^+ \in \mathit{Reach}_{\mathcal{D}^+}(\pi, s^+_I), \pi(s^+) = t^{\phi}_{q',q}\}$, *for every $q \in Q$; and*
- $\sigma_\pi(q \xrightarrow{\phi} q')(s) = \pi(s \wedge q \wedge \mathit{next}^{\phi}_{q'} \wedge \neg \mathit{goal})$, *for every state $s \in 2^V$ in planning domain $\mathcal{D}$.*

Proof. First, $s_I \in \sigma_S(q_I)$ because:

- s^+_I is trivially reachable by π ($s^+_I \in \mathit{Reach}_{\mathcal{D}^+}(\hat{\pi}, s^+_I)$);
- operator $t^{\mathsf{true}}_{q_0,q_I}$ is the only executable action at s^+_I w.r.t. $\mathcal{P}_A$ (recall the original domain actions where extended with precondition $\neg q_0$); so $\pi(s^+_I) = t^{\mathsf{true}}_{q_0,q_I}$.

Take now a state $s \in \sigma_S(q)$, for some $q \in Q$, and transition $q \xrightarrow{\phi} q'$ in P. Due to σ_S's construction, there exists $s^+ \in \mathit{Reach}_{\mathcal{D}^+}(\pi, s^+_I)$ such that $\pi(s^+) = t^{\alpha}_{q'',q}$ (action to transition to program state q). Due to the effects of operator $t^{\alpha}_{q'',q}$, state s^+ must have one possible successor state $s^{+\prime}$ such that $s^{+\prime} \models q \wedge \mathit{next}^{\phi}_{q'}$—a state where the transition of interest $q \xrightarrow{\phi} q'$ is "pending." Note that the projection of $s^{+\prime}$ over $\mathcal{D}$ is still s, as transition operators do not change the underlying domain. Since π is a strong-cyclic plan for $\mathcal{P}_{\mathcal{A}}$, every fair execution $s^{+\prime} s^+_1 s^+_2 \cdots$ will reach a state s^+_k such that $s^+_k \models q' \wedge \mathit{goal}$ and $\pi(s^+_k) = \bot$. This, in turn, can only happen when action $t^{\phi}_{q,q'}$ is executed in s^+_{k-1}, which means that, due to $t^{\phi}_{q,q'}$'s preconditions, the execution also reaches an (extended) state where ϕ holds (namely, s^+_{k-1} in $\mathcal{P}_{\mathcal{A}}$). Now, because how σ_π was defined to mimic π at every step, each of those fair executions correspond *one-to-one* to executions produced by policy σ_π for $\mathcal{D}$ when started in domain state s, namely, executions $s s_1 s_2 \cdots s_{k-1} s_k$, with s_i being the projection of s^+_i over the language of $\mathcal{D}$. Because $s^+_{k-1} \models \phi$ and $s^+_{k-1} = s^+_k$, it follows that $s_{k-1} = s_k \models \phi$, and hence every fair execution of π from s reaches a state where ϕ is true (in domain $\mathcal{D}$), and policy π is a strong-cyclic solution for planning task $\langle \mathcal{D}, s, \phi \rangle$. □

In addition, we can always reconstruct a solution policy for the encoding from a solution to the APP task.

Theorem 3 (Completeness). *Let $\mathcal{A}$ be an APP and $\mathcal{P}_{\mathcal{A}}$ is corresponding FOND planning task encoding. If σ is an APP-solution of $\mathcal{A}$, then the following policy is a strong cyclic solution for $\mathcal{P}_{\mathcal{A}}$:*

$$\pi(s^+) = \begin{cases} o & \text{if } s^+ \models q \wedge \mathit{next}^{\phi}_{q'}, \sigma_\pi(q \xrightarrow{\phi} q')(s) = o; \\ t^{\phi}_{q,q'} & \text{if } s^+ \models q \wedge \mathit{next}^{\phi}_{q'}, \sigma_\pi(q \xrightarrow{\phi} q)(s) = \bot; \\ t^{\mathsf{true}}_{q_0,q_I} & \text{if } s^+ \models q_0; \\ \bot & \text{if } s^+ \models \mathit{goal} \end{cases}$$

Proof. We will prove, by induction on its length, that any finite execution λ of π from (extended) state $s^+_1 = s^+_I$ and ending in state $s^+ \models q \wedge \mathit{next}^{\phi}_{q'}$ is such that *(i)* λ can be extended to a terminating execution of π where *goal* holds; and *(ii)* $\sigma_\pi(q \xrightarrow{\phi} q')$ is a strong-cyclic plan solution for planning task $\langle \mathcal{D}, s, \phi \rangle$.

Base case ($|\lambda| = 1$): Take $\lambda = s^+_I = s_I \wedge q_0 \wedge \mathit{next}^{\mathsf{true}}_{q_I}$. By construction, $\pi(s^+_I) = t^{\mathsf{true}}_{q_0,q_I}$. Due to the operator $t^{\mathsf{true}}_{q_0,q_I}$'s (non-deterministic) effects, $\lambda_1 = s^+_I s^{+\prime}_I$ is an execution of π where $s^{+\prime}_I = s_I \wedge q_I \wedge \mathit{goal}$, and in fact a terminating one with $\pi(s^{+\prime}_I) = \bot$, as $s^{+\prime}_I \models \mathit{goal}$.

Induction case ($|\lambda| = k + 1$): Consider an execution $\lambda = s^+_1 s^+_2 \cdots s^+_k s^+_{k+1}$ of π from $\mathcal{P}_{\mathcal{A}}$'s initial (extended) state $s^+_1 = s^+_I$. Suppose $s^+_k = s_k \wedge q \wedge \mathit{next}^{\phi}_{q'}$, for some

[4] To use other fairness notions in each planning transition, conditional fairness approaches would be required, as in (Rodriguez et al. 2022), as operators $t^{\phi}_{q,q'}$ ought to remain fair.

$q \in Q$ and $q \xrightarrow{\phi} q'$ in P—we know, by the structure of $\mathcal{D}^+$. By induction hypothesis, $\sigma_\pi(q \xrightarrow{\phi} q')$ is a strong-cyclic plan solution for planning task $\langle \mathcal{D}, s_k, \phi \rangle$.

Consider first the case where $\sigma_\pi(q \xrightarrow{\phi} q')(s_k) = \bot$. Due to HI, it must be the case that $s_k \models \phi$, and so it follows that $\pi(s_k^+) = t^{\phi}_{q,q'}$ (by definition of π). From $t^{\phi}_{q,q'}$'s effects, it follows that $s_{k+1} = s_k$ and one of two cases applies:

1. $s_{k+1}^+ \models \mathit{goal}$ and therefore $\pi(s_{k+1}^+) = \bot$. Hence, λ itself is a terminating execution of π reaching $\mathcal{P}_\mathcal{A}$'s goal. In addition, $\sigma_\pi(q \xrightarrow{\phi} q')$ is a strong-cyclic plan solution for planning task $\langle \mathcal{D}, s_{k+1}, \phi \rangle$, as $s_{k+1} = s_k$.
2. $s_{k+1}^+ \models s_{k+1} \wedge q' \wedge \mathit{next}^{\phi'}_{q''}$. The key now is to observe that $s_{k+1} = s_k \in \sigma_S(q')$, because π has reached s_k^+ by mimicking policy $\sigma_\pi(q \xrightarrow{\phi} q')$ and all terminating states produced by such policy ought to be in $\sigma_S(q')$—the set of "safe" domains states at APPP node q'. Then, $\sigma_\pi(q' \xrightarrow{\phi'} q'')$ must strong-cyclic solution for planning task $\langle \mathcal{D}, s_{k+1}, \phi' \rangle$, which implies that there exists an execution $s_{k+1}s_{k+2}\cdots s_{k+n}$, with $n \geq 1$, such that $\sigma_\pi(q' \xrightarrow{\phi'} q'')(s_{k+n}) = \bot$ and $s_{k+n} \models \phi'$. Again, by definition of π and domain $\mathcal{D}^+$, sequence $s_{k+1}^+ s_{k+2}^+ \cdots s_{k+n}^+ s_{k+n+1}^+$ will be a (terminating) execution of π from s_{k+1}^+, with $\pi(s_{k+n}^+) = t^{\phi'}_{q',q''}$, $s_{k+n+1}^+ \models \mathit{goal}$, and $\pi(s_{k+n+1}^+) = \bot$. Putting it all together, $\lambda \cdot s_{k+2}^+ \cdots s_{k+n}^+ s_{k+n+1}^+$ is a terminating execution of π from $s_1 = s_I^+$ that reaches a state where *goal* is true. Furthermore, $\sigma_\pi(q' \xrightarrow{\phi'} q'')$ is a strong-cyclic plan solution for planning task $\langle \mathcal{D}, s_{k+n+1}, \phi' \rangle$ trivially, because $s_{k+n+1} = s_{k+n}$ (recall that transition operators do not alter the underlying domain state) and so $\sigma_\pi(q' \xrightarrow{\phi'} q'')(s_{k+n+1}) = \bot$ and $s_{k+n+1} \models \phi'$.

Next, consider the case in which $\sigma_\pi(q \xrightarrow{\phi} q')(s_k) = o$, that is, a domain operator o is prescribed at s_k. By its definition, $\pi(s_k^+) = o$ and since $\mathcal{D}^+$ is just an extension of $\mathcal{D}$, s_{k+1} ought to be a possible successor state of s_k in $\mathcal{D}$ after execution operator o. Because $\sigma_\pi(q \xrightarrow{\phi} q')$ is a strong-cyclic policy for $\langle \mathcal{D}, s_k, \phi \rangle$ (HI) and s_{k+1} is reachable by the policy, there is an execution $s_k s_{k+1} \cdots s_{k+n}$, $n \geq 1$, of $\sigma_\pi(q \xrightarrow{\phi} q')$ from s_k that is terminating ($\sigma_\pi(q \xrightarrow{\phi} q')(s_{k+n}) = \bot$) on goal state $s_{k+n} \models \phi$. By construction of π, then, $s_k^+ s_{k+1}^+ \cdots s_{k+n}^+ s_{k+n+1}^+$, with $s_i^+ = s_i \wedge q \wedge \mathit{next}^{\phi}_{q'}$, $\pi(s_{k+n}^+) = t^{\phi}_{q,q'}$, and $s_{k+n+1}^+ = s_k \wedge \mathit{goal} \wedge \mathit{next}^{\phi'}_{q''}$, for some $q' \xrightarrow{\phi'} q''$ in P, and $\pi(s_{k+n+1}^+) = \bot$ is a goal-reaching terminating execution of π from s_k^+. Hence, $\lambda \cdot s_{k+2}^+ \cdots s_{k+n}^+ s_{k+n+1}^+$ is a terminating execution of π from $s_1 = s_I^+$ that reaches a state where *goal* is true. Furthermore, $\sigma_\pi(q \xrightarrow{\phi} q')$ is a strong-cyclic plan solution for planning task $\langle \mathcal{D}, s_{k+n+1}, \phi \rangle$ trivially, because $s_{k+n+1} = s_{k+n}$ (again, $t^{\phi}_{q,q'}$ do not alter the underlying domain state) and so $\sigma_\pi(q' \xrightarrow{\phi} q')(s_{k+n+1}) = \bot$ and $s_{k+n+1} \models \phi$. □

Our reduction demonstrates that APP framework is not more expressive than FOND planning. This adds to the existing literature that shows how FOND planning can capture many apparent more expressive synthesis problems (e.g., (Patrizi et al. 2011; Camacho et al. 2016)). Importantly, this reduction allows us to resort to existing FOND technology to solve agent planning programs. Note our reduction relies on strong-cyclic solution concept for FOND planning, as the non-determinism modelling the APP's transitions ought to be *fair* to appropriately handle cycles in the APP's graph. If some domain actions are not fair, then one would require *dual* FOND planning (Geffner and Geffner 2018; Rodriguez et al. 2022; Yadav and Sardiña 2023) to ensure that the non-determinism is appropriately handled. Next, we extend the above encoding to accommodate guarded transitions.

Incorporating Guard Conditions

Transitions in the planning program are, in general, of the form $q \xrightarrow{\gamma:\phi} q'$, where γ is the guard condition (over the planning domain variables) of the transition. We introduce additional propositions to modify the encoding, ensuring that the guard condition is verified before fully committing to the next transition step (represented by atoms *next*).

- $\mathit{req}^{\gamma}_{q',\phi}$, for each transition of the form $q \xrightarrow{\gamma:\phi} q'$ in P, representing the request, provided the guard γ holds, of a goal ϕ to evolve to planning program state q'.

Next, we slightly modify operator $t^{\phi}_{q,q'}$ to achieve operator $t^{\gamma:\phi}_{q,q'}$ and operate on these propositions (rather than on *next*):

$$\mathrm{Eff}_{t^{\gamma:\phi}_{q,q'}} = \neg q \wedge q' \wedge \Big(\bigvee_{q' \xrightarrow{\gamma':\phi'} q'' \text{in } P} (\neg \mathit{next}^{\phi}_{q'} \wedge \mathit{req}^{\gamma'}_{q'',\phi'}) \vee \mathit{goal} \Big).$$

Execution of this operator guarantees that commitment to next transition request $q' \xrightarrow{\gamma':\phi'} q''$ is conditional on γ' applying in the current domain state. Two additional "checking" operators are then used to either confirm or dismiss a (potential) request. For a transition $q \xrightarrow{\gamma:\phi} q'$ in P, the deterministic operator $\mathit{check}^{\gamma:\phi}_{q,q'}$ *confirms* a request whose guard holds:

- $\mathrm{Pre}_{\mathit{check}^{\gamma:\phi}_{q,q'}} = q \wedge \mathit{req}^{\gamma}_{q',\phi} \wedge \gamma$, that is, planning program is in state q, and a transition to state q' by achieving goal ϕ has been issued, whose guard γ applies in the domain.
- $\mathrm{Eff}_{\mathit{check}^{\gamma:\phi}_{q,q'}} = \mathit{next}^{\phi}_{q'} \wedge \neg \mathit{req}^{\gamma}_{q',\phi}$, that is, the request is confirmed (i.e., $\mathit{next}^{\phi}_{q'}$ is made true).

In turn, the non-deterministic operator $\mathit{check}^{\neg\gamma:\phi}_{q,q'}$ is used to "dismiss" potential requests whose guards do *not* hold by just re-issuing a new one:

- $\mathrm{Pre}_{\mathit{check}^{\neg\gamma:\phi}_{q,q'}} = q \wedge \mathit{req}^{\gamma}_{q',\phi} \wedge \neg\gamma$, that is, as above but where the guard γ does not hold.
- $\mathrm{Eff}_{\mathit{check}^{\neg\gamma:\phi}_{q,q'}} = \mathit{goal} \vee (\bigvee_{q' \xrightarrow{\gamma':\phi'} q'' \text{in } P} \mathit{req}^{\gamma'}_{q'',\phi'})$, that is, a new potential request is issued at current state q'.

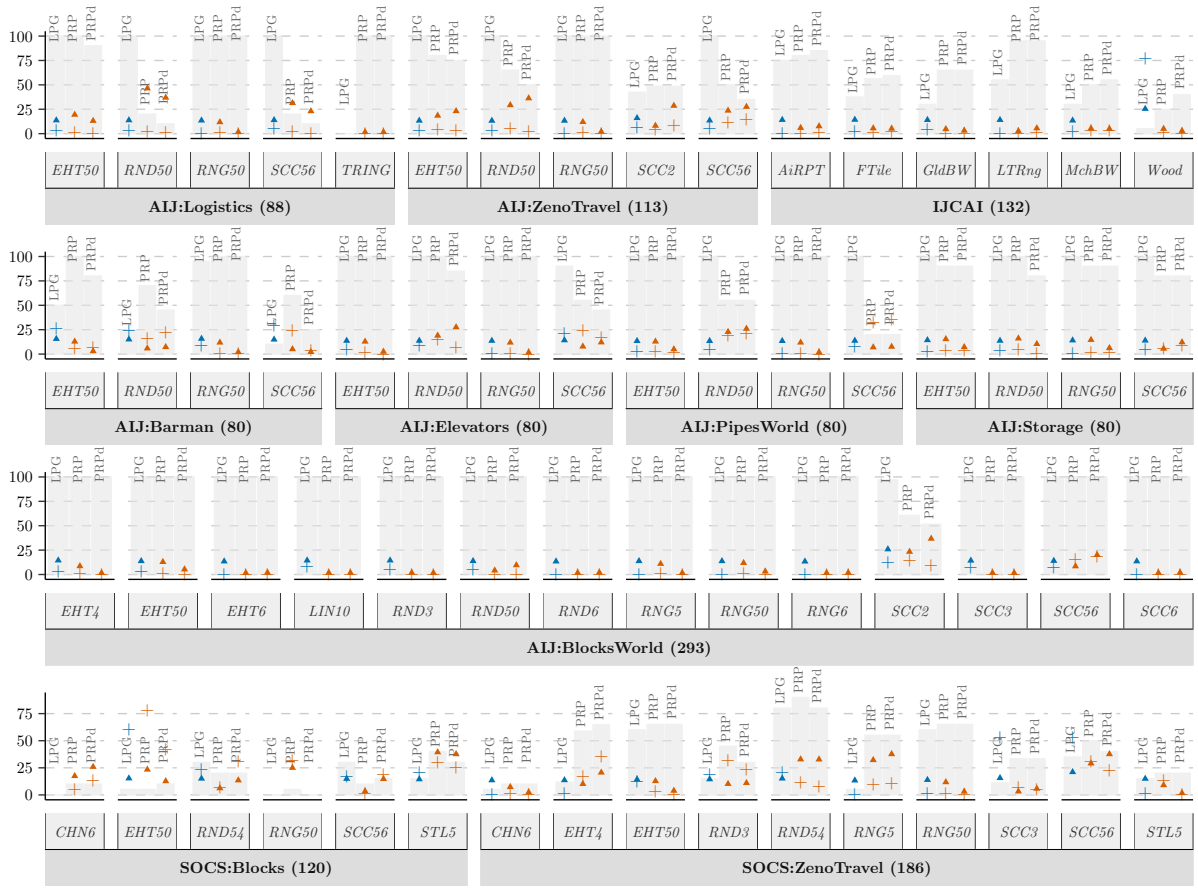

Figure 2: Comparison of LPG and FOND based approach on APP benchmarks. The y-axis shows results for coverage, *normalised* time (cputime divided by 1800s) and *normalised* memory (peak memory divided by 2Gb) for solved instances. Coverage is denoted by grey bars, whereas + and ▲ denote normalised time and normalised memory, respectively.

Note that, because *goal* is always a possible effect when issuing requests, this encoding works properly in the special case in which no transition is applicable from a program state: the user of the APP is not able to request any further goal and the program "terminates" successfully.

Experimental Evaluation

To evaluate our approach we conducted an empirical analysis between our FOND-based technique and the state-of-the-art search-based method for solving APPs (Gerevini, Patrizi, and Saetti 2011; De Giacomo et al. 2016). This method employs a specialized meta-search algorithm built upon the LPG classical preference-based planner (Gerevini, Saetti, and Serina 2003). We excluded the GR(1) reactive synthesis technique from (De Giacomo, Patrizi, and Sardina 2010) from our comparisons, as it was shown to be outperformed by the search-based approach.

We utilized three FOND planners: PRP (Muise, McIlraith, and Beck 2012), Paladiuns (Pereira et al. 2022), and PR2 (Muise, McIlraith, and Beck 2024). However, we report only PRP's results, as it performed best among the planners.[5] Specifically, we evaluated two variants of PRP: with and without invariant analysis in the pre-processor (denoted as PRP and PRPd, respectively).

We used three established benchmark sets: AIJ (De Giacomo et al. 2016), IJCAI (Chrpa, Lipovetzky, and Sardina 2017), and SoCS (Falcone, Gerevini, and Saetti 2017) for the evaluation. These benchmarks contain diverse structures for the program graph (modelled as $P = \langle Q, T \rangle$ in the APP definition) within *deterministic* planning domains from previous international planning competitions. Specifically, the AIJ and SoCS benchmarks contain program graph structures with different shapes, such as single cycles forming rings, multiple binary cycles resembling figure-eights, and various random structures. Additionally, the APP instances in the IJCAI and SoCS benchmarks are designed to incorporate "global" dead-ends—dead-ends occurring at the level of the overall APP program graph rather than at the level of individual transition planning goals.

All APP solvers were allocated 1800 seconds and 2 GB

[5] We are in conversations with PR2's authors to understand why it did not outperform PRP in our instances.

of memory using the hardware setup: a high-performance computing (HPC) system featuring a 2246 MHz processor and 62 GB of RAM. To get reliable results, we used BenchExec (Beyer, Löwe, and Wendler 2019),[6] a standalone benchmarking framework that ensures precise and reliable management of computational resources.[7]

Figure 2 provides an overview of our benchmarking results, where each panel is organized into two hierarchical levels. For the AIJ and SoCS benchmarks, the first level represents the planning domain, while the second level specifies the structure of the program graph (e.g., AIJ:Logistics). In contrast, the IJCAI instances do not feature varying program structures; thus, the second level corresponds to the planning domain. The numbers at the first level indicate the number of instances within each category. The y-axis illustrates normalized performance scores (ranging from 0 to 100) for three metrics: coverage, normalized time, and normalized memory. Time and memory are normalized by expressing them as percentages of their respective maximum allocated values (that is, 1800 seconds and 2 Gb, respectively).

We omit four program graph structures from the SoCS benchmark as they yielded zero coverage for all solvers, resulting in a total of 62 benchmarks for a detailed evaluation of our approach. For each benchmark, we display *coverage* (shown as grey bars labelled with the solver), *average time* (+), and *average memory* (▲) consumed by instances that were successfully solved within the given resource limits. The results for the LPG based solver (LPG) are shown first, followed by our FOND based approach (PRP and PRPd).

In terms of coverage, the LPG-based approach achieves the highest coverage in 22.6% (14 out of 62) of the cases, while the FOND-based approach (PRP and PRPd) attained the highest coverage in 42% (26 out of 62) of the benchmarks. Both approaches exhibit equal coverage in 35.4% (22 out of 62) of the benchmarks. We note that the FOND-based approach outperforms the LPG-based approach in benchmarks featuring global dead-ends (IJCAI and SoCS), whereas the LPG-based approach does better in the AIJ benchmarks (which do not exhibit dead-ends).

We observe that while the LPG-based approach includes optimizations tailored for APPs, the FOND approach does not have any, as it's based on a general reduction and FOND planners used off-the-shelf. Among the 22 benchmarks where the LPG search-based and FOND-based approaches achieved the same coverage, the FOND-based approach had statistically significantly faster solve times (p-value < 0.001)[8] in 17 benchmarks (PRP was 2 to 145 times faster than LPG). In contrast, the LPG-based approach demonstrated statistically significantly faster solve times in just two benchmarks (LPG was 1.5 to 2 times faster than PRP). In the remaining benchmarks, the differences in solve times were not statistically significant (p-value > 0.001).

Furthermore, to demonstrate the applicability of our approach to *non-deterministic* planning domains, we modified the Blocksworld domain from AIJ by introducing non-deterministic variants of the pick, stack, and unstack actions. We assume fair non-determinism, analogous to most work in FOND planning, where strong cyclic solutions apply. Figure 3 presents the computational performance of PRP[9] on these benchmarks. In contrast to the deterministic Blocksworld planning domain (see Figure 2), the program graph structures significantly influence problem difficulty. In particular, coverage was 100% in 11 out of 12 benchmarks for the deterministic variant, whereas in the non-deterministic case, coverage ranged from 0% (for SCC2) to 100% (in 5 out of 12 benchmarks). Second, in non-deterministic domains, the difference in average solve time between PRP and PRPd was not practically significant (the differences in average times were below 5%).

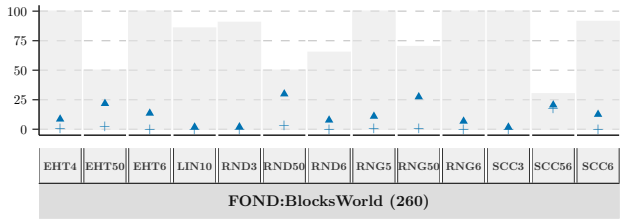

Figure 3: Experimental results for PRP on non-deterministic blocksworld domain.

Conclusion

Agent planning programs extend planning problems by replacing the single goal with a network of goals. The ability to model complex, non-terminating, goal-oriented behaviors in APPs makes them an attractive framework for declarative goal-oriented "programming." We have shown here that APPs can be translated efficiently into FOND planning tasks that can be then solved with planners off-the-shelf. We have established the correctness of the translation as well as empirically shown its competitiveness in several APP benchmarks. In addition, we have provided a solution concept for APP problems that is equivalent to the standard solution concept but is arguably simpler and directly related to planning.

Our technique is also amenable to optimizations, such as incorporating reasoning about global dead-ends (Chrpa, Lipovetzky, and Sardina 2017) and goal "lookaheads" (Falcone, Gerevini, and Saetti 2017). Wrt the former, the so-called "traps" (Lipovetzky, Muise, and Geffner 2016) for goals in the APP can be directly added to the corresponding FOND problem, rather than to each transition planning task. In turn, we can achieve goal *lookaheads* by removing (some) dummy goals in the translation, thus forcing the PRP planner to build weak plans for *sequences* of APP goals. The hypothesis is that more robust weak plans will be generated, and hence backtracking search would be reduced within the search in PRP. Our preliminary testing supports the hypothesis: the addition of lookaheads resulted in a notable increase in coverage in LIN50 (by 30%) and RNG50 (by 15%) benchmarks for the SOCS:Blocks domain.

[6] https://github.com/sosy-lab/benchexec

[7] Benchmarks & code: https://github.com/ssardina-research/app

[8] Using Wilcoxon test to assess differences in solve times.

[9] LPG solver is valid only for deterministic domains.

Acknowledgments

We acknowledge the support of the Nectar Research Cloud Australian research platform and the Benchexec team for our experiments, and thank the anonymous reviewers for their comments.

References

Beyer, D.; Löwe, S.; and Wendler, P. 2019. Reliable benchmarking: requirements and solutions. *International Journal on Software Tools for Technology Transfer*, 21(1): 1–29.

Bloem, R.; Jobstmann, B.; Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2012. Synthesis of Reactive(1) designs. *Journal of Computer and System Sciences*, 78(3): 911–938.

Camacho, A.; Triantafillou, E.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2016. Non-Deterministic Planning with Temporally Extended Goals: Completing the Story for Finite and Infinite LTL. In *Proc. of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KNOWPROS)*.

Chrpa, L.; Lipovetzky, N.; and Sardina, S. 2017. Handling non-local dead-ends in Agent Planning Programs. In *Proc. of IJCAI*, 971–978.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2): 35–84.

Daniele, M.; Traverso, P.; and Vardi, M. 2000. Strong Cyclic Planning Revisited. *Recent Advances in AI Planning*, 35–48.

de Boer, F. S.; Hindriks, K. V.; van der Hoek, W.; and Meyer, J.-J. 2007. A Verification Framework for Agent Programming with Declarative Goals. *Journal of Applied Logic*, 5(2): 277–302.

De Giacomo, G.; Gerevini, A.; Patrizi, F.; Saetti, A.; and Sardina, S. 2016. Agent Planning Programs. *Artificial Intelligence*, 231: 64–106.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2010. Agent Programming via Planning Programs. In *Proc. of AAMAS*, 491–498. IFAAMAS.

Falcone, F.; Gerevini, A.; and Saetti, A. 2017. On Realizing Planning Programs in Domains with Dead-End States. *Proc. of SOCS*, 8(1): 20–28.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geffner, T.; and Geffner, H. 2018. Compact Policies for Fully Observable Non-Deterministic Planning as SAT. In *Proc. of ICAPS*, 88–96.

Gerevini, A.; Bonet, B.; and Givan, B., eds. 2006. *Booklet of 4th International Planning Competition*. Lake District, UK.

Gerevini, A.; Patrizi, F.; and Saetti, A. 2011. An Effective Approach to Realizing Planning Programs. In *Proc. of ICAPS*, 323–326.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning Through Stochastic Local Search and Temporal Action Graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20: 239 – 290.

Hübner, J. F.; Bordini, R. H.; and Wooldridge, M. 2006. Plan patterns for Declarative Goals in AgentSpeak. In *Proc. of AAMAS*, 1291–1293.

Lipovetzky, N.; Muise, C. J.; and Geffner, H. 2016. Traps, Invariants, and Dead-Ends. In *Proc. of ICAPS*, 211–215.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *Proc. of ICAPS*, 172–180.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2024. PRP Rebooted: Advancing the State of the Art in FOND Planning. In *Proc. of AAAI*, 20212–20221.

Patrizi, F.; Lipovetzky, N.; De Giacomo, G.; and Geffner, H. 2011. Computing Infinite Plans for LTL Goals Using a Classical Planner. In *Proc. of IJCAI*, 2003–2008.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL Synthesis for Non-Deterministic Systems using Strong Cyclic Planners. In *Proc. of IJCAI*.

Pereira, R. F.; Pereira, A. G.; Messa, F.; and Giacomo, G. D. 2022. Iterative Depth-First Search for FOND Planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Porfirio, D.; Roberts, M.; and Hiatt, L. M. 2024. Goal-oriented end-user programming of robots. In *Proc. of HRI*, 582–591.

Rintanen, J. 2003. Expressive Equivalence of Formalisms for Planning with Sensing. In *Proc. of ICAPS*, 185–194.

Rodriguez, I. D.; Bonet, B.; Sardiña, S.; and Geffner, H. 2022. Flexible FOND Planning with Explicit Fairness Assumptions. In *Journal of Artificial Intelligence Research (JAIR)*. JAIR.

Sardina, S.; and D'Ippolito, N. 2015. Towards Fully Observable Non-deterministic Planning as Assumption-based Reactive Synthesis. In *Proc. of IJCAI*, 3200–3206.

Sardina, S.; and Padgham, L. 2011. A BDI Agent Programming Language with Failure Recovery, Declarative Goals, and Planning. *Autonomous Agents and Multi-Agent Systems*, 23(1): 18–70.

van Riemsdijk, B.; Dastani, M.; and Meyer, J.-J. 2005. Semantics of Declarative Goals in Agent Programming. In *Proc. of AAMAS*, 133–140.

Yadav, N.; and Sardiña, S. 2023. A Declarative Approach to Compact Controllers for FOND Planning via Answer Set Programming. In *Proc. of ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 2818–2825.