ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230593

A Declarative Approach to Compact Controllers for FOND Planning via Answer Set Programming

Nitin Yadav[a] **and Sebastian Sardina** [b;*]

[a]The University of Melbourne
[b]RMIT University

Abstract. We present an approach to non-deterministic planning under full observability via Answer Set Programming. The technique can synthesise *compact* policies, handle both fair and unfair actions simultaneously, and readily accommodate control knowledge and procedural domain constraints. We show that whereas compact controllers may yield sub-optimal behavior under a naive executor, optimality can be recovered under a smarter executor. The developed planner is succinct, elegant, and directly implementable, thus providing higher confidence of its correctness and ease of elaboration. Experimental results show that its performance is competitive.

1 Introduction

Automated AI planning [12, 16] studies representational and computational techniques for the synthesis of plans for autonomous agents to achieve their goals. The planning system—a solver—takes as input a model of a dynamic system, in terms of how actions can change the system, and automatically builds a strategy for the agent to achieve its goal from an initial world state. In *fully-observable non-deterministic* (FOND) planning [8, 6], actions with alternative possible effects are allowed at the representational level: the agent does not know which outcome will occur when it executes an action, but it will be able to observe the outcome once executed. Despite its conceptual simplicity, FOND planning has proven to be a very powerful framework, and it has been connected to very expressive problems in AI and CS, like generalized planning [31], service/behavior composition [25], and even general reactive synthesis [5].

Solving FOND planning problems is hard—it is EXPTIME-complete [27]. To address this, some of the best performing FOND planners (e.g., [22, 24, 23, 10, 19]) involve sophisticated search-based algorithms implementing a collection of procedural techniques. This makes it difficult to assess their correctness or elaborate them, for example, to accommodate different types of fairness notions. In addition, as argued in [13], planners that rely on classical planners do not address the non-determinism directly, thus struggling in settings where "risky" non-determinism can result in the computation of a high number of linear plans not leading to a full solution.

In this paper, we join efforts to develop *declarative* solvers for FOND planning (e.g., [2, 13, 29]), and develop a solver whose logic (and hence correctness) is directly accessible, is tolerant to elaboration [21], and can reason globally about the impact of non-deterministic decisions. Our planner is realised in the Answer Set Programming (ASP) paradigm [14, 20], and is arguably the simplest FOND planning solver to date. We draw mostly from the SAT-based approach in FOND-SAT [13], but also take ideas from the search-based PRP approach [22]. From the former, we take the idea of computing (compact) automata-type solutions via model construction; from the latter, we share the use of regression to extract the relevant aspects at each decision point in a controller. The use of ASP in lieu of SAT solvers allow us to separate specification of the actual planner from that of an instance problem, and to offload the grounding to the ASP solver, thus yielding a lifted encoding of the planner and taking advantage of advances in grounding techniques.

As with FOND-SAT, our planner aims to construct *compact* controllers. We show, however, that more compact controllers may yield sub-optimal behavior at execution time under a naive executor—runs may be unnecessarily longer. We show then how to address this via a smarter executor. This yields a more complete understanding of compactness, optimality, and execution for FOND planning. We also show that, due to its declarative nature, our planner can easily be extended to, for example, accommodate unfair actions, specify domain control knowledge, or add optimisation features. We perform experiments to compare our simple planner with PRP, PALADINUS, and FOND-SAT. Those experiments confirm the findings [13] that model-based solvers, like ours and FOND-SAT, perform worse on problems where non-determinism does not impose a "risk," but outperform search-based solvers on problems where there are many miss-leading classical plans that cannot be extended to a full policy solution.

2 FOND Planning

We present here the non-deterministic planning framework to be used in the rest of the paper. Wlog, we use a STRIPS notation, in which goals and action preconditions are positive atoms.

A ***Fully-observable non-deterministic (FOND) planning problem*** is a tuple $\mathcal{P} = \langle At, I, Act, G \rangle$, where At is the set of propositional atoms describing the domain, $I, G \subseteq At$ are the sets of atoms describing the initial state and goal condition, resp., and Act is the set of domain actions. An action $a \in Act$ is defined by its precondition $Prec(a) \subseteq At$ and non-deterministic effects $Eff(a) = e_1 \mid \ldots \mid e_k$, with $k \geq 1$, where each $e_i = \langle Add(e_i), Del(e_i) \rangle$, with $Add(e_i), Del(e_i), \subseteq At$ is a possible deterministic effect of the action (expressed with add and delete sets): when action a is performed, one of these effects will ensue, by the environment's choice.[1]

* Corresponding Author: sebastian.sardina@rmit.edu.au.

[1] This formalization of (non-deterministic) actions corresponds to 1ND Normal Form [26] with no nested conditional (deterministic) effects, and to the usual `(oneof e1 ... en)` clauses in PDDL [15].

The semantics of a FOND problem can be defined in terms of its possible execution *runs* [12]. A ***domain state*** $s \subseteq 2^{At}$ is the set (or conjunction) of propositional atoms that are true in the state. We use S to denote the set of all possible states in $\mathcal{P}$. The successor of a state s given a specific (deterministic) effect e can be defined as $succ(s,e) = s \setminus Del(e) \cup Add(e)$. We can generalize that to actions $a \in Act$, to obtain all possible ***successor states***, one per effect, after executing action a in a state s as follows: if $s \not\models Prec(a)$, then $succ(s,a) = \emptyset$; otherwise $succ(s,a) = \bigcup_{e \in Eff_a} \{succ(s,e)\}$. An ***execution run*** from state $s \in S$ is a, possibly infinite, sequence of states and actions of the form $\tau = s_0 a_0 s_1 a_1 s_1 \cdots s_k a_k s_{k+1} \cdots$ such that $s_0 = s$, and $s_{i+1} \in succ(s_i, a_i)$, for all $i \geq 0$.

A (tabular) ***policy*** $\pi : S \mapsto Act$ is a partial function stating which action ought to be executed at a given state. We use $Exec(\pi, s)$ to denote the set of all possible ***maximal execution runs*** when following policy π from state s, that is, all runs τ from s such that $a_i = \pi(s_i)$, for all $i \geq 0$ and that cannot be further extended. There have been several solution concepts for FOND planning depending on the *fairness assumptions* imposed on the non-deterministic actions [8, 6]. Roughly speaking, a ***fair action*** is one in which (in all runs) all its effects are guaranteed to occur infinitely often when the action is executed infinitely many times in the same state [12, 30]. When all actions are assumed fair, a ***strong-cyclic policy*** guarantees that the agent, by "re-trying" when needed, eventually achieves the goal [6]. In turn, when no fairness can be assumed, a plan with acyclic executions that reaches the goal in a bounded number of steps—a ***strong policy***—is required. The ***Dual FOND*** hybrid variation has recently been introduced to deal with domains that have both fair and unfair actions/effects [4, 13, 29]. In that setting, a solution amounts to a policy whose "fair" executions w.r.t. the actions/effects assumed to be fair (not necessarily all) are goal reaching. Lastly, we note that while planning under non-determinism is EXPTIME-complete [27], effective optimized techniques and solvers have been developed, and is an area of significant active work (e.g., [22, 19, 18, 24, 5, 13, 29, 23]).

3 FOND Planning as ASP

We propose an *answer set logic program* to compute solution policies for FOND planning problems. ASP [20] is closely related to SAT solving in that it allows the specification of a (solution to a) problem as a set of constraints, albeit in a more high-level specification language that includes, among other constructs, negation-as-failure [7]. An ASP solver then computes so-called Answer Sets, which are *stable models* of the logic program [14]. Stable models provide a clean and practical semantics of logic programs with negation as failure with roots in AI non-monotonic reasoning theories, under which a model can be seen as the possible "beliefs" that an agent associated with program may have. As with SAT, there are competitive solvers available, such as Clingo [11] or DLV [1]. Thus, following the ASP declarative *generate-and-check* specification paradigm, our ASP program transparently constraints policy controllers to those that are solutions to the given FOND planning task; and we extract such controllers from the stable models—answer sets—of the program, which we compute using state-of-the-art Clingo [11] solver.

Our proposal borrows and integrates elements from previous works on FOND planning. We follow FOND-SAT [13] SAT-based approach, in that we aim to compute a sort of finite-state automaton that *compactly* encodes a (strong-cyclic) solution policy. However, we use the Answer Set Programming paradigm and the regression-based technique used by PRP [22]. The use of ASP in lieu of SAT solvers allows one to model FOND planning at a higher-level of abstraction. Importantly, our solution separates the encoding of the solver (i.e., the planning system) from that of an instance. Moreover, it offloads the grounding to the ASP solver, thus yielding a lifted encoding of the planner and taking advantage of advances in grounding techniques. In turn, like PRP [22] does, we use the regression technique [28] to focus the controller on the *relevant* domain features at each decision step as its *weakest-precondition*.[2] Finally, we note that while FOND-ASP [29] is also an ASP-based FOND planner, it explicitly operates on the domain state space, which is in practice significantly larger than the controller state space.

From now on, we shall assume a FOND planning problem instance $\mathcal{P} = \langle At, I, Act, G \rangle$. Its encoding Π_P in ASP is a collection of facts using the following atoms:

- `atom(P)`: for each predicate `P` $\in At$.
- `action(A)`: for each action `A` $\in Act$. In addition, to define an action's precondition and effects we use the following terms:
  - `prec(A, P)`: atom `P` is in precondition of action `A`.
  - `effect(A, Ei)`: associates an action with its i-th effect E_i.
  - `add(A, Ei, P)`: effect E_i of action `A` adds atom `P`.
  - `del(A, Ei, P)`: effect E_i of action `A` deletes atom `P`.
- `init(P)`: predicate `P` $\in I$ is true in the initial state.
- `goal(P)`: predicate `P` $\in G$ is in the goal condition.

Note that our encoding for actions does not require the "all outcomes determinisation" approach used in FOND-SAT and PRP: we encode actions directly with their non-deterministic effects.

In a nutshell, following [13], our ASP encoding aims to constrain the way a finite number of controller states are annotated with domain actions and linked to each other, so that the resulting structure amounts to a (strong-cyclic) solution policy for the FOND planning task at hand. Whereas the domain action associated with a controller state indicates the prescribed action to be executed, the links between controller states represent the possible successor states, one per action's effects, of the controller upon execution of the prescribed action. Note the difference between controller states and domain states: the former stand for the state of the executor solving the problem; the latter are concrete states of the planning domain. A key feature is that a controller state can compactly prescribe the same action for (exponentially) many domain states. The idea of using finite automata-like structures for complex sequential decision-making is not new, and has even been applied to decision-theoretic settings [17].

Next, we describe the ASP program—the planning system—encoding the constraints on a controller solution. Unlike SAT-based approaches to planning, the ASP planning system is *generic*, in that it is a *fixed program* that is evaluated together with the facts Π_P encoding a problem instance. The following atoms are used to encode the constraints on a candidate controller with $k+1$ states (remember these are *controller* states, not domain states):

- `state(I)`: denotes a controller state with number `I` where $0 \leq$ `I` $\leq k$. The term `state(0..k)` defines the controller states. States 0 and k are reserved for the initial and final state, resp.
- `holds(S, P)`: atom `P` $\in At$ is required to hold in a state `S`.
- `policy(S, A)`: policy prescribes action `A` in state `S`.
- `next(S1, E, S2)`: State `S2` is the successor of state `S1` as a result of effect `E` (of the action prescribed in `S1`).
- `reachableG(S)`: goal is reachable from state `S`.

[2] This yields an arguably more natural encoding than that of FOND-SAT, which is based on propagating negative atoms forward. Nonetheless, we explain later that a simple change yields the FOND-SAT style approach.

```
1  state(0..k)
2  {policy(S, A): action(A)} = 1:- state(S), S != k.
3  {next(S1, E, S2): state(S2)} = 1 :-
4              policy(S1, A), effect(A, E).
5
6  holds(S, P):- policy(S, A), prec(A, P).
7  holds(S1, P):- next(S1, E, S2), holds(S2, P),
8      policy(S1, A), not add(A, E, P).
9  -holds(S2, P):- next(S1, E, S2),
10     policy(S1, A), del(A, E, P).
11 -holds(0, P) :- atom(P), not init(P).
12 holds(k, P) :- goal(P).
13
14 reachableG(S):- state(S), S=k.
15 reachableG(S):- next(S, _, S1), reachableG(S1).
16 :- not reachableG(S), state(S).
```

Figure 1: The CFOND-ASP planner $\Pi(k)$, parametrized by $k \geq 0$. Atoms `policy/2` in stable models stand for a compact controller.

The Answer Set Program $\Pi(k)$ in Figure 1 is the *generic* FOND planner imposing constraints on a candidate controller with $k \geq 0$ controller states. The program follows the *generate-and-check* methodology, and is therefore succinct and mostly self-explanatory, but let us quickly go over the main parts. First, the choice rules in lines 2 and 3 "generate" candidate controllers, by annotating each non-goal controller state with an domain action and defining a successor (controller) state for each effect of the action, respectively.

The next five rules define when a predicate must hold true in the domain when the controller is in a state. The first rule (line 6) indicates that all preconditions of an action are required in a controller state where the action is executed. The rule in line 7 implements a one-step regression: an atom holds in a (controller) state, if an effect of the action executed in that state does not make the predicate true, and it is (already) required in the successor state. The third rule (line 9) states that a predicate cannot be required to hold true in a controller sate if it may be the resulting state of an effect that deletes that predicate. Informally, these three rules identify the *weakest precondition* at each controller state, that is, the minimum conditions required for the controller (at the state) to guarantee goal reachability. It is interesting to observe that this is realising, albeit in a purely declarative manner, the regression phase performed by PRP on weak plans to obtain policy rules with respect to the *relevant* aspects of the domain states. The last two rules for `holds/2` describe the requirements for the initial and goal controller state.

The final three rules in the planner (lines 14-16) "check" that the controller generated adhere to one of the usual characaterizations of strong-cyclic policies: *the goal is reachable from any policy state*. Like [29], we take advantage of the minimality property of stable models and the first-order features of ASP to define reachability directly, thus dispensing us from the need to encode numbers to capture distances to the goal, as done by FOND-SAT. This will also support elaborations of solution concepts in a more parsimonious manner.

To solve a FOND planning task $\mathcal{P}$, we check for a stable model on program $\Pi_{\mathcal{P}} \cup \Pi(k)$, starting with bound $k = 0$ and incrementally increasing it until one is found. A stable model M defines a controller C_M with $(k+1)$ states $(0 \ldots k)$ that is a solution to $\mathcal{P}$. The controller starts in its state 0, when in state n, it executes the action $C_M(n)$ and evolves to its successor state $succ_{C_M}(n, e)$ depending on the effect $e \in \mathit{Eff}(C_M(n))$ that ensues. As done with policies in Section 2, a controller C from a state s yields a set of ***enacted executions runs*** $\mathit{Exec}(C, s)$ of the form $\tau = (s_0, n_0)a_0(s_1, n_1)a_1(s_2, n_2)\cdots$, such that *(i)* $s_0 a_0 s_1 a_1 s_2 \cdots$ is an execution run; *(ii)* for each $i \geq 0$, $a_i = C(n_i)$ and $n_{i+1} \in succ_C(n_i, e)$, for some $e \in \mathit{Eff}(a_i)$ and $s_{i+1} = succ(s_i, e)$; and *(iii)* if finite, it cannot be further extended with C. Then, as standard, C is a ***strong-cyclic controller solution*** for $\mathcal{P}$ iff every fair enacted controlled run $\tau \in \mathit{Exec}(C_M, I)$ is goal reaching.

Theorem 1 *Let $\mathcal{P}$ be a FOND planning problem and M be a stable model of ASP program $\Pi_P \cup \Pi(k)$, for some $k \geq 0$. Then, C_M is a strong-cyclic controller solution for $\mathcal{P}$.*

Proof idea. By reaching a contradiction if a run $\tau \in \mathit{Exec}(C_M, I)$ would not reach the goal. First, if τ were finite, it would imply that a controller state was reached which prescribes an action (this should always happen due to choice rule in line 2) that is not executable in the domain. This could not happen due to line 6 in $\Pi(k)$ for `holds/2` which requires all preconditions of a prescribed action to be guaranteed at the controller state. Second, if τ were infinite, because τ is fair, it would imply that the controller itself includes a subset of states that are disconnected from the goal state k. This is not possible due to integrity constraint in line 16. □

For completeness, it is straightforward to construct a controller C^π for a strong-cyclic tabular policy solution π such $\mathit{Exec}^-(C^\pi, I) = \mathit{Exec}(\pi, I)$, where $\mathit{Exec}^-(C, s)$ is like $\mathit{Exec}(C, s)$ but with all controller states removed from the enacted runs. To do so, we associate a unique controller state to each domain state reachable by π from initial state I—an explicit controller. However, we show that any strong-cyclic tabular policy π will be represented by some solution controller which uses only *relevant* parts of π and therefore whose size may be smaller than π. The set of execution traces resulting from such a controller will be a subset of the execution traces of the given tabular policy solution. To that end, and following PRP's approach to generalize its policies, consider the ***regression operator*** $\mathcal{R}(s)$ on domain states relative to the FOND task $\mathcal{P}$ [28]: *(i)* $\mathcal{R}(s) = G$, if $s \models G$; and *(ii)* $\mathcal{R}(s) = \mathit{Prec}(a) \cup \bigcup_{e \in \mathit{Eff}(\pi(s))} \mathcal{R}(succ(s, e)) \setminus \mathit{Add}(e)$, otherwise. In our case $\mathcal{R}(s)$ is a partial state defined by the exact set of predicates that must hold in state s for the goal to be reachable from s via π—the relevant predicates.[3] Next, let $N_\pi(\mathcal{P}) = |\{\mathcal{R}(s) : s \in S \text{ is reachable from } \pi \text{ in } \mathcal{P}\}|$ be the number of different relevant partial states for π.

Theorem 2 *Let π be a strong-cyclic tabular policy solution for FOND planning task $\mathcal{P}$. Then, there exists a stable model M of program $\Pi_P \cup \Pi(N_\pi(\mathcal{P}))$, such that $\mathit{Exec}^-(C_M, I) \subseteq \mathit{Exec}(\pi, I)$ and C_M is a strong-cyclic controller solution for $\mathcal{P}$.*

Proof idea. We show how to construct C_M from π. The set of controller's states is $\{\mathcal{R}(s) : s \in S \text{ is reachable from } \pi \text{ in } \mathcal{P}\}$. The action associated with a state is given by: $C(\mathcal{R}(s)) = \pi(s)$, if there are two $s, s' \in S$ such that $\mathcal{R}(s) = \mathcal{R}(s')$ then pick any. The successor of state n from effect e for action $C(n)$ is $\mathcal{R}(succ(s, e))$. By construction, we have that $\mathit{Exec}^-(C_M, I) \subseteq \mathit{Exec}(\pi, I)$. To show C_M is a strong cyclic controller solution for $\mathcal{P}$: *(i)* the regression operator ensures that if an action is applicable in a state s then it is also applicable in partial state $\mathcal{R}(s)$; *(ii)* goal is reachable in (all fair traces) from a state s that is reachable in $\mathcal{P}$ following π. As a result, goal will also be reachable in induced traces following actions as per C_M from a state $\mathcal{R}(s)$ in the controller. □

Observe that a compact controller may have a size even smaller than $N_{\pi(\mathcal{P})}$. In cases where the solution policy π decides on actions

[3] Note this is also analogous to π-reduced states in FOND-SAT [13], but we further restrict relevancy up to until the first step when a future needed proposition (precondition or goal condition) is added (and not later deleted).

based on irrelevant facts a compact controller may be such that it ignores the largest irrelevant part of the policy π. This happens in cases where there are two states s and s' such that $\mathcal{R}(s) = \mathcal{R}(s')$. The completeness proof is indifferent to which state (s or s') is ignored.

While we argue that specifying the constraints of a controller based on weakest-preconditions, as in the spirit of PRP, is more natural and intuitive, one can readily encode FOND-SAT approach by simply replacing the rule in line 7 with the following one:

```
1 -holds(S2, P) :- next(S1, E, S2), -holds(S1, P),
2     policy(S1, A), not add(A, E, P).
```

Under this rule, instead of propagating the weakest-preconditions backwards, atoms that may be false are propagated forward.

Dual FOND: Integrating fair and unfair actions One of the key advantages of realising a solver from its logical specification is that one can more easily extend it to support other features and solution concepts. Integrating fair with unfair actions has proven to be very challenging for algorithm-based techniques [4], and hence most those planners do not support the mixture. In our case, we can directly import the ideas from FOND-ASP [29] to account for domains mixing both types of actions, except now operating on the controller state space rather than in the much larger domain state space. So, assuming unfair actions are specified using the `unfair/1` atom, we simply replace lines 14 to 16 with the following ones:

```
1 terminates(k).
2 terminates(S) :- policy(S, A), not unfair(A),
3     next(S, _, S1), terminates(S1).
4 terminates(S) :- policy(S, A), unfair(A),
5     terminates(S1): next(S, _, S1).
6
7 :- state(S), not terminates(S).
```

The last integrity constraint requires every controller state to be deemed terminating, that is, eventually reaching the goal state. The goal state is trivially terminating. If the action prescribed in a controller state is fair, then the state is guaranteed to terminate if any successor is terminating. On the other hand, if the action prescribed is an unfair one, one cannot rely on all effects arising by re-trying sufficiently often and, hence, termination of the controller state is guaranteed only if *all* successor states are deemed terminating.

4 Controller Compactness and Optimality

We now discuss two different properties of our FOND controllers, namely, *compactness* and *optimality* of execution. One of the key advantages of model-based techniques such as FOND-SAT [13] and the one proposed here is the ability to compute *compact* controllers. These are controllers whose sizes can be exponentially smaller than the reachable state space when enacted in the planning domain; see [13] for an example. Intuitively, compactness results from generalising planning states by only considering facts that must hold in all possible evolutions to a state. For instance, at each location in the Tireworld domain one may end up with a flat tire and may need repair by using a spare tire in that location. However, on reaching the next location one does not need to remember if a spare tire has been used or not in any previous locations. This information is not essential to decide on next action(s) to reach a goal state.

However, it turns out that compact controllers can yield suboptimal behavior under a naive execution: there are runs that could have been done in a shorter manner. To demonstrate that, consider a version of the Tireworld domain in which the action $\mathsf{go}(x, y)$ to drive from location x to location y may not only produce a flat tire, but also leave the fuel tank below an acceptable threshold. Fortunately, actions `change` and `refuel` can fix both issues.

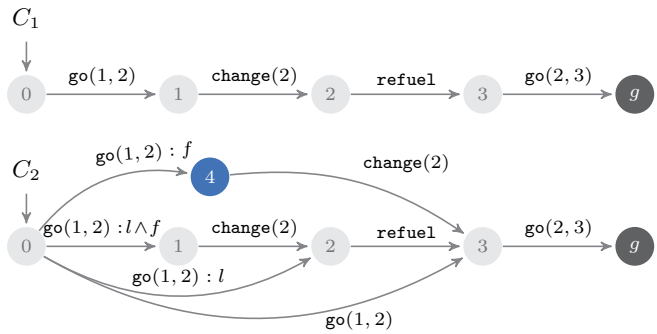

Figure 2: Case where compactness can lead to suboptimality. Controller C_1 is more compact, but can yield longer executions than C_2.

Algorithm 1 SmartExecutor(C, s)

1: $open \leftarrow \{k\}$
2: **loop**
3: $\quad A \leftarrow \{C(n) \mid n \in open, s \models \mathcal{R}(n)\}$
4: $\quad$ **if** $A \neq \emptyset$ **then return** A **else**
5: $\quad\quad open \leftarrow \{n \mid e \in \mathit{Eff}(C(n)), succ_C(n, e) \in open\}$
6: $\quad$ **end if**
7: **end loop**

Consider next a domain with three locations 1, 2 and 3 in a straight line shape, each allowing refueling and one spare tire change. The smallest controller C_1 requires only 5 states; see Figure 2. Observe that such controller does not keep a track of the status of tire and fuel, and hence produces a conformant-type plan that changes the tire and refuels always, regardless whether necessary or not. In turn, while having one more state, controller C_2 tracks the non-deterministic effects of an low fuel tank (l) and a flat tire (f), and can more efficiently decide what is needed or not, if anything. As one can see, if driving goes smoothly, controller C_2 will get to the goal in two driving actions, whereas C_1 will require four, two being unnecessary!

Recovering optimality. The fact that more efficient controller can be obtained with larger number of controller states poses a challenge: there appears to be a trade-off between compactness and execution optimality. It turns out, however, that the more efficient executions can indeed be recovered under a smarter executor. In a nutshell, this smarter executor may "fast-forward" its behavior when possible.

Algorithm 1 presents a smart executor that skips unnecessary actions when possible. In a nutshell, such an executor "jumps" to any controller state n that is *closer* to the goal k and such that the current domain state satisfies the regression $\mathcal{R}(n)$—the *sufficient* conditions that must hold in the domain so that the controller can guarantee goal reachability from its state n. The regression $\mathcal{R}(n)$ with respect to a controller C with $(k + 1)$ states, where n is now a *controller state*, can be defined analogously to how it was defined for domain states in Section 3: *(i)* $\mathcal{R}(k) = G$; and *(ii)* $\mathcal{R}(n) = \mathit{Prec}(C(n)) \cup \bigcup_{e \in \mathit{Eff}(C(n))} \{\mathcal{R}(succ(n, e)) \setminus \mathit{Add}(e)\}$, otherwise.

Procedure SmartExecutor could yield many possible applicable actions when these are possible and equidistant from the goal state in the controller. We say that π is a ***smart*** (tabular) policy for controller C if $\pi(s) \in \mathsf{SmartExecutor}(C, s)$, for all $s \in S$. Informally, a smart policy captures an efficient way of executing the controller.

Now, to compare runs in terms of "efficiency," we say that run τ' is a sub-run of run τ, denoted $\tau' \sqsubseteq \tau$, if τ' can be derived from τ by replacing substring of the form $sa_0s_1 \cdots a_m s$ in τ with just s, that is, by projecting out an unnecessary sub-run in τ.

Theorem 3 *Let C be a strong-cyclic controller solution for $\mathcal{P}$ and $\tau \in Exec^{-}(C, I)$ a run under C. Let τ' be the shortest maximal execution run in $\mathcal{P}$ such that $\tau' \sqsubseteq \tau$ of $\mathcal{P}$. Then, there exists a smart policy π^{+}_{C} for C such that $\tau' \in Exec(\pi^{+}_{C}, I)$.*

Proof idea. To show existence of such a smart policy, take any subsequence that is part of τ but not in τ'. Suppose $s_1a_1s_2$ in τ' maps to a subsequence $s_1a'_1 \ldots s_1a_1s_2$ in τ. Let n_1 and n'_1 be the controller states associated with the same domain state s_1 such that $C(n_1) = a_1$ and $C(n'_1) = a'_1$. Since τ' is maximal and shortest, n_1 is closer to goal state than n'_1. Hence, on domain state s_1 the smart controller will have that $C(n_1) \in \mathsf{SmartExecutor}(C, s_1)$. □

Thus, by executing it smartly, we can keep the smallest controller efficient. Note that executing a controller C smartly is linear on the size of the controller. While our controllers, like those of FOND-SAT, may be exponentially smaller than the explicit tabular policies, they may still be exponential on the original domain in the worst case.

5 Optimisations for compact FOND ASP planner

We improve the core planner in Figure 1 with three diverse efficiency optimisations. First, we add symmetry breaking with respect to transitions in a controller. We then show, theoretically, the existence of a lower bound to the size of a controller, which can be used to reduce the number of steps one has to iterate to look for a solution. Finally, we take the advantage of the declarative nature of CFOND-ASP and show how control knowledge can be easily accommodated.

Symmetry breaking. Observe that the encoding in Figure 1 is free to connect any two controller states with a transition and therefore can produce two different controllers C and C' such that their induced traces are the same (that is, $Exec^{-}(C, I) = Exec^{-}(C', I)$). We define two constraints to breaks this symmetry. The first constraint on line 1 in Figure 3 enforces an ordering on how states are connected via transitions. For an intuitive understanding of this constraint consider three states `n1`, `n2`, and `n3` such that `n1 > n2 > n3`. Suppose that `n3` is a successor of `n1`, then it should not be the case that there is no transition to `n2` from a state with a number lesser than or equal to that of `n1`. In other words, in the case described above, since `n2` is smaller than `n3`, `n1` should have `n2` as the next available successor if available. We note that this constraint was part of the FOND-SAT implementation but not described in the paper. This constraint is unable to break symmetry with respect to transitions due to non-deterministic effects. Hence, we introduce a new second constraint (line 3 in Figure 3) that enforces an ordering between successor states due to different effects of an action. We use a simple lexicographic ordering between multiple effects of an action by using the term `lex` where given two effects Ei and Ej the predicate `lex(Ei,Ej)` is true if $i < j$.

```
1 :- next(I,_,J), J > I, J > 1, not goalState(J),
2          not next(K,_,J-1): state(K), K<=I.
3 :- lex(E1, E2), next(S, E1, J1), next(S, E2,
      J2), S<J2<J1, not goalState(J1), {next(S,
      E3, J2): lex(E3, E2), not goalState(J2)}=0.
```

Figure 3: Symmetry breaking rules.

Weak plan backbones. Current model-based techniques for FOND planning iteratively check for existence of a controller by incrementing the number of available states. The underlying solver (that is, sat solver in case of FOND-SAT and an ASP solver in our case) proves unsatisfiability for iterations 1 to $k-1$ where k is the size of a compact solution controller. One way to improve the speed of these planners is to start the iterative process with a number l where it is always guaranteed that $1 \leq l \leq k$. The number l will be a provable lower bound on k (the size of the controller). We show that such a lower bound always exists and is actually the size of a shortest weak plan for the planning instance. We call such a plan a "backbone". As a consequence of Theorem 1, a shortest weak plan always exists in a controller (it is the shortest path from the initial controller state 0 to its goal state k).

The next result shows that the size of the shortest weak plan is a lower bound on the size of a controller.

Theorem 4 *Let π_w be a shortest weak plan for a FOND problem $\mathcal{P}$. If the length of π_w is l, then the smallest controller C that is a strong cyclic controller solution for $\mathcal{P}$ will have at least $l+1$ of states.*

Proof idea. We will use proof by contradiction. Suppose there exists a strong cyclic controller C^* for $\mathcal{P}$ where C^* has $m \leq l$ states. Next, consider the shortest trace $\tau = (s_0, n_0), a_0, \ldots a_{k-1}(s_k, n_k) \in Exec(C^*, I)$. Therefore, we know that there exists a weak plan of length k in $\mathcal{P}$. Since C^* has m states, we have that $k < m \leq l$. That is, $\mathcal{P}$ has a weak plan shorter than π_w: a contradiction as π_w is a shortest weak plan in $\mathcal{P}$. □

Control knowledge. Another way to boost the efficiency of FOND planning is to embed control knowledge in the solving process. Here we discuss two types of control knowledge. First, constraints on eliminating actions that undo the effect of a previous action from policy consideration. Second, addition of knowledge that arises from the domain itself.

We restrict the modelling of undo to deterministic actions. The atom `det(A)` denotes that action `A` is deterministic. We say that two deterministic actions `A1` and `A2` have a mismatch in the context of undo, if `A1` adds an atom `P` but `A2` does not delete `P` (line 1 in Figure 4), or if `A1` deletes `P` and `A2` does not add `P` (line 2 in Figure 4). Actions where add and delete effects do not mismatch are considered undo actions (line 3 in Figure 4). Given a state n and its successor n', we eliminate policy tuples that prescribe undo actions to n and n' (line 4). We note that this approach is sound but incomplete in identifying all undo actions, as there may exist undo actions involving non-deterministic effects. However, this would require additional domain knowledge on effects that are not considered a failure.

```
1 notUndo(A1,A2):- add(A1,_,P), not del(A2, _,P).
2 notUndo(A1,A2):- del(A1,_,P), not add(A2, _,P).
3 undo(A1, A2) :- det(A1), det(A2), not
      notUndo(A1, A2), not notUndo(A2, A1).
4 :- 1{policy(S2, A2): policy(S1, A1), state(S1),
      state(S2), undo(A1, A2), next(S1,_,S2)}.
```

Figure 4: Rules to model deterministic undo actions.

Next we discuss two examples of including domain specific knowledge in the CFOND-ASP planner. First, a domain may have atoms that should always be false (or true) in every solution. The listing below adds the knowledge that an atom `P` should always be false: a controller should never prescribe an action that *may* make it true. This is relevant, for example, in the MINER domain where a person should never be dead.

```
1 :- state(S), holds(S, P).
2 :- policy(S, A), add(A,_, P).
```

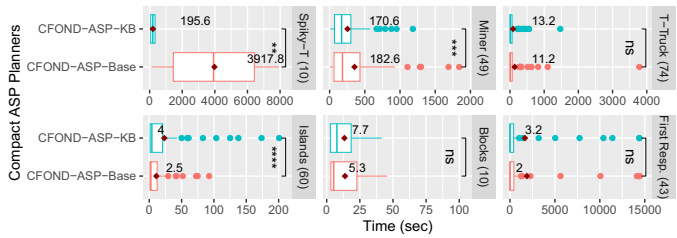

Figure 5: Distribution of solve times for CFOND-ASP-Base (without control knowledge) and CFOND-ASP-KB (with control knowledge). The median of the distributions is labelled in the respective box plots.

Second, we model the ability to repair in cases of failure. In these cases on has to identify the type of failure (e.g., a flat tire) and what is required for repair (e.g., a spare tire). We use the term `possible` to identify actions that may lead to failure. The listing below adds the knowledge that if a policy prescribes an action that may lead to a failure then necessary atoms required for its repair should also hold in that state. This is useful in domains such as SPIKY TIREWORLD and TRIANGLE TIREWORLD, where a tire may become flat.

```
1 :- state(S), policy(S, A), possible(A, F),
     required(P, F), not holds(S, P).
```

6 Experimental Results

To demonstrate the effectiveness of our proposed approach we first show the efficiency gained by incorporation of proposed optimisation techniques, such as estimation of minimum controller size and use of control knowledge. Then, we will compare our ASP-based approach against state-of-the art FOND planners PALADINUS [23], FOND-SAT [13], and PRP [22]. (We do not report FOND-ASP as it often runs out of memory when enumerating the domain state space.) In all experiments we aim to compute strong cyclic solutions and set a maximum controller size of 100 for controller-based approaches.

The set of FOND benchmarks includes the new domains introduced in [13], namely DOORS, ISLANDS, MINER, TIREWORLD SPIKY, and TIREWORLD TRUCK. The other classical FOND domains tested include ACROBATICS, BEAM WALK, BLOCKSWORLD, CHAIN OF ROOMS, EARTH OBSERVATION, ELEVATORS, FAULTS, FIRST RESPONDERS, TIREWORLD, TRIANGLE TIREWORLD, and ZENOTRAVEL. We only considered planning instances that are solvable. The total number of solvable instances are 210 for the new FOND domains and 348 for the classical FOND domains.

Lower bound on controller size: We compare the average size of a backbone with the average size of a compact controller for different domains to estimate how close is this bound. Table 1 shows this information for domains where the compact ASP planner solved at least 10 instances. In our approach, ASP-based classical planner PLASP [9] was used to compute the "backbones" for lower bound on controller size. As one would expect the backbone size was smaller than the controller size for all instances.

We note that the time spent by the ASP solver increases non-linearly with an increase in the number of controller states. Hence, utilising backbone size is not expected to significantly improve the coverage of the planner. This is consistent with the observation that on average the lower bound is closer to the controller size for classical FOND domains than new FOND domains. However, this does not necessarily translate to higher number of solved instances. Instead, it helps in reducing the time spent on proving unsatisfiability in the initial iterations. If the subsequent iterations are expensive the planner would need other optimisation techniques (such as domain knowledge) to improve its performance. Another benefit of estimating minimum controller size is to avoid a large number of unsatisfiability iterations if a classical planner does not return a solution.

Domain	n	$\sum \lvert C\rvert/n$	$\sum \lvert B\rvert/n$
Doors	15	19.00	9.20
Earth observation	10	19.20	10.90
Faults	45	14.93	7.44
First Responders	47	10.57	9.53
Islands	60	8.20	4.00
Miner	50	20.42	7.69
Spiky Tireworld	11	23.00	8.00
Tireworld	11	6.45	4.00
Tireworld Truck	74	15.26	4.81

Table 1: Comparison of the average size of controller ($\sum |C|/n$) and backbone ($\sum |B|/n$).

Control knowledge: At the outset, providing additional domain knowledge is expected to improve efficiency. However, these additional constraints may be redundant (if the solver has learnt them). Moreover, addition of new facts or constraints may lead to increase in memory consumption. We used two versions of the ASP planner, with (named CFOND-ASP-KB) and without control knowledge (named CFOND-ASP-Base). Both were run with 2 threads. The domains for undo actions included ACROBATICS, BLOCKSWORLD (15 instances), ELEVATORS, FIRST RESPONDERS, ISLANDS, MINER, SPIKY TIREWORLD, TIREWORLD TRUCK, and domain specific knowledge was used for MINER and SPIKY TIREWORLD.

Figure 5 shows the distribution of solve times for six of the selected domains across instances that were solved by both these planners. We exclude results of the remaining domains as the coverage was low (see Figure 6). We used Wilcoxon paired test to check significance of the difference in solve times between CFOND-ASP-Base and CFOND-ASP-KB. There was improvement in solve times for the domain of SPIKY TIREWORLD ($p = 0.00391$). As result of this CFOND-ASP-KB solved 2 more instances as compared to CFOND-ASP-Base, thus resulting in improved coverage. Addition of control knowledge for MINER resulted in reduction in solve times ($p = 0.00015$), however, there was a slight reduction in coverage (1 instance out of 50) with addition of knowledge due to the effect on memory. In the domains of TIREWORLD TRUCK, FIRST RESPONDERS, and BLOCKSWORLD adding undo actions had no effect (the difference was not significant, $p > 0.2$). Finally, for ISLANDS addition of control knowledge yielded an increase in solve times ($p < 0.00001$). We note that the ASP solver is already fast on the ISLANDS domain and addition of domain constraints resulted in extra processing for the solver.

Comparison with other planners: We used two versions of our compact ASP planner.[4] The first one named CFOND-ASP1 did not use any optimisations and used a single thread. The second version named CFOND-ASP2 estimated the minimum controller size, made use of control knowledge in certain domains (see above for the selected domains), and used two threads. We used CFOND-ASP1 for a fair comparison with other single threaded planners. We compared our approach with state-of-the art FOND planners PALADINUS (labelled PALAD), FOND-SAT, and PRP (labelled PRP). FOND-SAT relies on the choice of a sat solver and is bundled with minisat as the default. However, for a fair comparison we used FOND-SAT with both minisat (labelled FSMST) and glucose (labelled FSGLU).

All planners were run on Nectar Research Cloud computing machines, each with 64GB memory and 32 AMD EPYC-Rome (2.24 GHz) CPU Processors. A memory limit of 4GB and a time limit of 4 hours was enforced by using the system utility `systemd-run`. In addition, a CPU limit of 100% was used for CFOND-ASP1, FOND-SAT, PALADINUS, and PRP, and a limit of 200% was used for CFOND-ASP2. We also set a heap limit of 3GB (75% of the memory

[4] Available at https://github.com/ssardina-research/cfond-asp

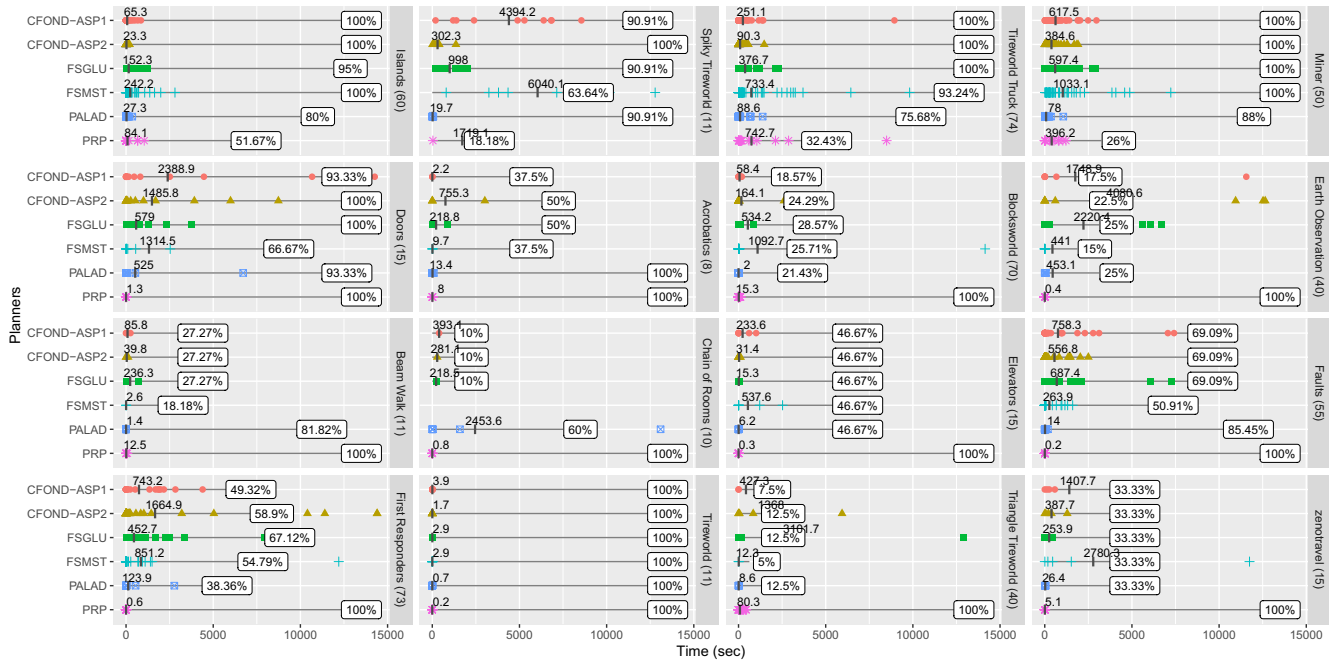

Figure 6: Benchmarking results for planners on solvable FOND instances for strong cyclic planning. Planners CFOND-ASP1, CFOND-ASP2, FSGLU, FSMST, PALAD, and PRP correspond to base compact ASP, compact ASP with 2 threads and control knowledge (where available), FOND-SAT with Glucose/MiniSAT, PALADINUS and PRP, respectively. Coverage is depicted in boxes, distribution of run times is shown as colour coded points, and the mean of the distribution of solve times is denoted marked on a vertical line.

limit) for JVM when running PALADINUS. In absence of this (limit on the JVM heap) PALADINUS exceeded the memory limit on a significantly higher number of instances.

Figure 6 shows the distribution of all instances that were solved by the planners. We evaluate planners first on the coverage and second on differences in solve time if more than one planner had the highest coverage. We used Wilcoxon paired test to check significance of the difference in solve times between planners. Planner CFOND-ASP2 outperformed all other planners in 4 out of the 5 new FOND domains (first row of Figure 6). In ISLANDS CFOND-ASP1, CFOND-ASP2 and FOND-SAT achieved 100% coverage, however, CFOND-ASP2 had the fastest solve times ($p =< 0.0001$ for CFOND-ASP2 vs CFOND-ASP1 and CFOND-ASP2 vs FSMST). CFOND-ASP2 had the highest coverage in the SPIKY TIREWORLD domain. Though in this domain PALADINUS solved most instances quicker than CFOND-ASP2, it does this at the expense of higher memory requirement, and as a result had a lower coverage than CFOND-ASP2. In TIREWORLD TRUCK, CFOND-ASP1, CFOND-ASP2, and FSGLU obtained 100% coverage, however, CFOND-ASP2 had the fastest solve times ($p < 0.0001$ for CFOND-ASP2 vs CFOND-ASP1 and CFOND-ASP2 vs FSGLU). All model-based solvers achieved maximum coverage in the MINER domain. CFOND-ASP2 was the fastest solver ($p < 0.0001$, CFOND-ASP2 vs rest), the difference in solve times was not significant between CFOND-ASP1 and FSGLU ($p = 0.093$). Finally, in the DOORS domain CFOND-ASP2, FSGLU, and PRP achieved 100% coverage however PRP was the fastest ($p < 0.001$ for PRP vs CFOND-ASP2 and PRP vs FSGLU). We note that in ISLANDS, TIREWORLD TRUCK, and MINER CFOND-ASP1 came second in place after CFOND-ASP2. (Note: FOND-ASP [29] ran out of memory in TIREWORLD TRUCK already in the fourth instance.) In the domains where CFOND-ASP2 did well, we also compared it against a version of FOND-SAT with Kissat [3], a state-of-the-art sat solver. The CFOND-ASP2 planner outperformed Kissat-based FOND-SAT in the domains of ISLANDS and SPIKY-TIREWORLD.

In the classical FOND domains PRP outperformed all other planners. In TIREWORLD, though all planners achieved 100% coverage, PRP was the fastest ($p < 0.001$, PRP vs rest). There was no significant difference between solve times of CFOND-ASP2, FSMST, and FSGLU ($p > 0.2$). In ACROBATICS, PRP and PALADINUS performed equally well (there was no significant difference in solve times, $p > 0.2$). PRP achieved 100% coverage in rest of the classical FOND domains. This is partly because these domains allow fast computation of repairable weak plans by PRP [13]. In these domains if we compare within the model-based approaches, FOND-SAT and CFOND-ASP2 achieved similar coverage results.

7 Conclusions

We presented a FOND planning system in Answer Set Programming. While extremely succinct and declarative, the system is competitive against the state-of-the art planners in domains with meaningful nondeterminism [13]. In addition, due to its high-level declarative nature, the ASP-based planner proposed is easily amenable for elaboration, including incorporating optimisation features, control knowledge, and alternative solution concepts (e.g., integrating unfair actions). We also provided lower bounds on the size of any solution controller, and used that to speed up the solving process. Finally, we demonstrated that controllers, while compact in nature, could yield suboptimal behavior—longer than necessary runs—and showed how to overcome this by resorting to a smarter executor.

One can extend the proposed approach in diverse ways. First, one could look at alternate controller specifications that utilise the "multi-shot" solving technique from ASP. One could also look at generating more meaningful weak plans and incorporating knowledge such as essential actions—landmarks—that must be executed in every policy. Finally, one could model user preferences over controllers, by taking advantage of the optimisation features of existing ASP solvers.

Acknowledgements. We acknowledge the use of the Nectar Research Cloud collaborative Australian research platform to perform our experiments, and thank the anonymous reviewers for their insightful comments and suggestions.

References

[1] Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari, 'The ASP system DLV2', in *Proc. of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 10377 of *Lecture Notes in Computer Science (LNCS)*, pp. 215–221. Springer, (2017).

[2] Chitta Baral, Thomas Eiter, and Jicheng Zhao, 'Using SAT and logic programming to design polynomial-time algorithms for planning in non-deterministic domains', in *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pp. 578–583. AAAI Press, (2005).

[3] Armin Biere and Mathias Fleury, 'Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022', in *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pp. 10–11. University of Helsinki, (2022).

[4] Alberto Camacho and Sheila A. McIlraith, 'Strong-cyclic planning when fairness is not a valid assumption', in *Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning*, (2016).

[5] Alberto Camacho, Eleni Triantafillou, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith, 'Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces', in *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pp. 3716–3724. AAAI Press, (2017).

[6] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso, 'Weak, strong, and strong cyclic planning via symbolic model checking', *Artificial Intelligence*, **147**(1-2), 35–84, (2003).

[7] Keith L. Clark, 'Negation as failure', in *Logic and Data Base*, 292–322, Plenum Press, New York, (1978).

[8] M. Daniele, P. Traverso, and M. Vardi, 'Strong cyclic planning revisited', *Recent Advances in AI Planning*, 35–48, (2000).

[9] Yannis Dimopoulos, Martin Gebser, Patrick Lühne, Javier Romero, and Torsten Schaub, 'Plasp 3: Towards effective asp planning', in *Logic Programming and Nonmonotonic Reasoning: 14th International Conference, LPNMR 2017*, pp. 286–300. Springer, (2017).

[10] Jicheng Fu, Vincent Ng, Farokh Bastani, and I-Ling Yen, 'Simple and fast strong cyclic planning for fully-observable non-deterministic planning problems', in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1949–1954, (2011).

[11] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub, 'Multi-shot ASP solving with Clingo', *Theory and Practice of Logic Programming*, **19**(1), 27–82, (2019).

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

[13] Tomas Geffner and Hector Geffner, 'Compact policies for fully observable non-deterministic planning as SAT', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 88–96, (2018).

[14] Michael Gelfond and Vladimir Lifschitz, 'The stable model semantics for logic programming', in *Proc. of the International Conference on Logic Programming (ICLP)*, pp. 1070–1080. The MIT Press, (1988).

[15] Alfonso Gerevini, Blai Bonet, and Bob Givan, eds. *Booklet of 4th International Planning Competition*, Lake District, UK, 2006.

[16] Malik Ghallab, Dana S. Nau, and Paolo Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann Publishers Inc., May 2004.

[17] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra, 'Planning and acting in partially observable stochastic domains', *Artificial Intelligence*, **101**(1-2), 99–134, (1998).

[18] Peter Kissmann and Stefan Edelkamp, 'Solving fully-observable non-deterministic planning problems via translation into a general game', in *Proceedings of the Annual German Conference on AI*, pp. 1–8, (2009).

[19] U. Kuter, S. Nau, D., E. Reisner, and P. Goldman, R., 'Using classical planners to solve nondeterministic planning problems', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 190–197, (2008).

[20] Vladimir Lifschitz, *Answer Set Programming*, Springer, 2019.

[21] John McCarthy, 'Mathematical logic in artificial intelligence', *Daedalus*, **117**(1), 297–311, (1988).

[22] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck, 'Improved non-deterministic planning by exploiting state relevance', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 172–180, (2012).

[23] Ramon Fraga Pereira, André Grahl Pereira, Frederico Messa, and Giuseppe De Giacomo, 'Iterative depth-first search for fond planning', in *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*, (2022).

[24] Miguel Ramirez and Sebastian Sardina, 'Directed fixed-point regression-based planning for non-deterministic domains', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 235–243, Portsmouth, NH, USA, (2014). AAAI Press.

[25] Miguel Ramirez, Nitin Yadav, and Sebastian Sardina, 'Behavior composition as fully observable non-deterministic planning', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 180–188, (2013).

[26] Jussi Rintanen, 'Expressive equivalence of formalisms for planning with sensing', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 185–194, (2003).

[27] Jussi Rintanen, 'Complexity of planning with partial observability', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 345–354, (2004).

[28] Jussi Rintanen, 'Regression for classical and nondeterministic planning', in *Proc. of the European Conference in Artificial Intelligence (ECAI)*, pp. 568–572, (2008).

[29] Ivan D. Rodriguez, Blai Bonet, Sebastian Sardiña, and Hector Geffner, 'Flexible FOND planning with explicit fairness assumptions', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 290–298. AAAI Press, (2021).

[30] Sebastian Sardina and Nicolas D'Ippolito, 'Towards fully observable non-deterministic planning as assumption-based reactive synthesis', in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3200–3206, (2015).

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