Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

# Non-Deterministic Conformant Planning Using a Counterexample-Guided Incremental Compilation to Classical Planning

**Enrico Scala,**[1] **Alban Grastien**[2]
[1]University of Brescia, Italy
[2]Australian National University, Australia
enricos83@gmail.com, alban.grastien@anu.edu.au

## Abstract

We address the problem of non-deterministic conformant planning, i.e., finding a plan in a non-deterministic context where the environment is not observable. Our approach uses an unsound but complete reduction from non-deterministic conformant planning to classical planning to find a candidate plan; the validity of this plan is then verified by a SAT solver; if the plan is invalid, the reduction is revised to guarantee that the invalid plan will not be valid in the classical planning problem. This procedure is executed until a valid plan is found, or it is shown that there is no plan. Experiments show that this approach provides a nice trade-off between fast but unsound, and complete but slow approaches.

## Introduction

Conformant planning is the problem of finding a sequence of actions that enables an agent to reach some goal despite uncertainty on the initial state or the effects of the actions,[1] and without the help of observations (Smith and Weld 1998). This problem has applications in contexts where observations are either hard to make or unreliable, and where the mission is expensive or potentially dangerous, and therefore requires some guarantees on success.

Conformant planning can be formulated as a search problem over the belief space, where a belief is a set of possible states (Bonet and Geffner 2000). As a consequence, conformant planning is a computationally expensive problem, specifically EXPSPACE-complete (Haslum and Jonsson 1999). Different approaches have been proposed to tame this complexity (see Related Work section). One of the main issues of the existing approaches is that they either try to solve the problem in a complete manner (which is hard) or they use an incomplete method.

Instead, we follow the general principles of Counter-Example Guided Abstraction Refinement (CEGAR (Clarke et al. 2000)). Our approach uses an unsound but complete method to generate candidate plans, which means that the candidate plan may be invalid (however, if a valid plan exists, then a candidate plan will be produced). The validity of this plan is then tested. If the plan is invalid, this information is added to our method and a new candidate plan is generated that is guaranteed to be different. The procedure continues until either a valid plan is found or the problem is proved unsolvable.

More precisely, the candidate plan is found by solving a planning problem defined as the combination of two objects:

1. A "one-outcome determinisation" of the conformant planning problem, i.e., a planning problem in which the uncertainty is broken at random.
2. A "counter-automaton", which stores the reasons why the previous candidates are invalid.

The resulting planning problem is a relaxation of the original one in the sense that it accepts more solutions. Importantly, this relaxed problem is a classical one, i.e., it does not include any uncertainty, and it is therefore easy to solve, relatively to the original problem.

The validity of the plan is verified by a SAT solver. Given a plan, the goal is to find a sequence of states that could result from the execution of the plan, such that one of the actions is inapplicable in the corresponding state.

This approach aims at finding a sweet spot between 1) searching in the belief space, which is very hard because this space is exponentially larger than the state space and does not have good heuristics, and 2) compiling the conformant planning problem into a deterministic one, which can fail at the pre-processing phase. By incrementally learning the aspects of the planning problem that are relevant, we avoid unnecessary work during the search.

In the next section we present the background definitions on conformant planning. We then give an example and present a high-level view of our algorithm. We present how we reduce conformant planning to classical planning and, in particular, the notion of an adversarial NFA. Next, we explain how to verify the validity of a plan and how this information is stored in the adversarial NFA. We then show the correctness of the algorithm. We discuss how to exploit the structure to improve the generalisation. After a discussion on related work, we provide an experimental analysis.

## Background

Our definition of the conformant planning problem with non-deterministic effects is similar to other definitions that



[1]The term "non-deterministic" is used to emphasise the fact that uncertainty also applies to the actions.

can be found in the literature. We only report here the salient aspects. A *conformant planning problem* $\mathbb{P}$ is the tuple $\langle A, I, G\rangle$, where $A$ are actions, $I$ is the initial state[2], $G$ is a *goal action* name[3]. All structures in $\mathbb{P}$ predicate over some universe of propositional facts $F$ that we leave implicit; with $L$ we indicate all literals from $F$, and with $\Gamma$ the set of propositional formulas over $L$. An action $a \in A$ is a tuple $\langle \text{name}(a), \text{pre}(a), \text{eff}(a)\rangle$ where:

- $\text{name}(a)$ is a label identifying the action name. When obvious, this part of the action will be ignored.
- $\text{pre}(a)$ maps the action into a precondition from $\Gamma$ and
- $\text{eff}(a)$ maps the action into a set of conditional effects $\{CE_1, \ldots, CE_m\}$ such that each $CE_i$ is a relation $\phi \rhd \{e_1, \ldots, e_k\}$ where $\phi$ is a formula in $\Gamma$ and $e_1, \ldots, e_k$ are subsets of $L$.

An action is said to be *deterministic* if $\forall(\phi \rhd E) \in \text{eff}(a)$. $|E| = 1$. The all-outcome determinisation of an action $a$ is a set of actions $O(a)$ where each $o \in O(a)$ is a particular choice for the non-deterministic effects over $a$.

**Semantics.** A *state* $s$ is a set of facts, i.e., $s \in 2^F$. A fact $f$ is true in $s$ if $f \in s$, false otherwise (Closed World Assumption). A *belief* is a set of states, i.e. $\mathcal{B} \in 2^S$ where $S = 2^F$ is the set of states. A state $s$ satisfies the formula $\psi$ if $eval(\psi, s)$ evaluates to true, where $eval(\psi, s)$ is the evaluation of the formula obtained by substituting with true or false all facts that are true or false in $s$.

An action $a$ is applicable in a state $s$ if $s$ satisfies $\text{pre}(a)$. The result of applying $a$ in $s$ is a state $s'$ such that there exists a deterministic action $o \in O(a)$ and $s' = s[o]$, where $s[o]$ is the state obtained by executing $o$ following the semantics of classical planning. Notice how $s'$ is not decided deterministically. A *plan* $\pi$ is a sequence of action names $\pi = \langle \text{name}(a_1), \ldots, \text{name}(a_n)\rangle$, sometimes more simply replaced by the sequence of actions $\langle a_1, \ldots, a_k\rangle$.[4] An *execution* of $\pi$ from state $s_0$ is a sequence of states $s_1, \ldots, s_n$ where each $s_i$ is one of the states resulting from the execution of $a_i$ in $s_{i-1}$. The execution is *valid* iff $a_i$ is applicable in each state $s_{i-1}$ and $a_n$ is the goal action.

An action $a$ is said to be applicable in a belief $\mathcal{B} \in 2^S$ iff all states in $\mathcal{B}$ satisfy $\text{pre}(a)$. The execution of an action $a$ in a belief $\mathcal{B}$, $\mathcal{B}[a]$, is a new belief that is obtained as follows: Let $\langle \mathcal{B}_1, \ldots, \mathcal{B}_n\rangle$ be the beliefs obtained by applying iteratively the actions named in $\pi$ starting from $\mathcal{B}_0 = I$, we say that $\pi$ is *valid* iff each $a_i$ is applicable in $\mathcal{B}_{i-1}$ and its last action is the goal action. The set of valid plans to the problem $\mathbb{P} = \langle A, I, G\rangle$, i.e., valid from the belief $\mathcal{B} = \{I\}$, is denoted $\Pi(\mathbb{P})$.

[2]Because the uncertainty in the initial state can be simulated by an initial non-deterministic action, we assume in the definitions that there is only one initial state.

[3]$G$ is usually a set of goal states, but it is convenient for us to interpret it as an action whose precondition is satisfied precisely by the goal states.

[4]Our approach reduces non-deterministic conformant planning to classical planning and, consequently, modifies the precondition and effects of each action; to be able to compare the sets of valid plans of either problem, we need to define a plan as a sequence of action names rather than a sequence of actions.

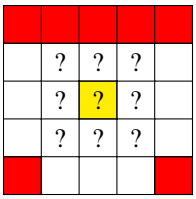

Figure 1: A graphical representation of the TRICKYGRID for a $5 \times 5$ grid. The robot can move in all 4 directions: down (D), up (U), left (L), right (R), but going right might move it up as well. Red cells are forbidden; the yellow cell is the target; the question marks are the possible starting states. A possible solution that can be generalised to larger maps is: $D \times 3, U, L \times 3, (R, D) \times 2, D, U \times 2$.

## Example

We illustrate the definitions with an example that is the basis for the new domain TRICKYGRID, which we also use in the experiments.

In this example (Figure 1), a robot is traveling on an $5 \times 5$ grid. The top row of the grid, as well as the other two corners are dangerous locations and, for this reason, the robot is forbidden to enter them. The robot can move in all four directions (*down*, *up*, *left*, *right*). The robot however is slightly impaired. As a consequence, the *right* move has a chance of turning into a diagonal *up-right* move. If the robot hits the edge of the grid, the move does not change its position. The initial position of the robot is one of the 9 central cells. The goal is to get the robot to the center of the map. The robot is unable to observe its position.

We now show how this problem is modeled in our formalism. The state of the robot is modeled through the facts $x(1), \ldots, x(5)$ and $y(1), \ldots, y(5)$ that represent the $x$ and $Y$ position of the robot.

An action such as *left* will have four conditional effects such as: $x(3) \rhd \{e_1\}$ where $e_1 = \{x(2), \neg x(3)\}$. This conditional effect is deterministic (it has only one set of effects, namely $e_1$), and has two consequences: if the fact $x(3)$ is true is the current state, this fact becomes false after the action is applied and $x(2)$ becomes true instead.

The action *right* affects the $x$ facts in a manner similar to *left*, but it also includes conditional effects such as $y(2) \rhd \{e_1, e_2\}$ where $e_1 = \emptyset$ and $e_2 = \{y(3), \neg y(2)\}$. This means that performing the *right* move when $y(2)$ is true has the following non-deterministic effect: either the fact remains true, or $y(2)$ becomes false while $y(3)$ becomes true.

The verification that the robot did not enter a forbidden cell is made by forcing the robot to perform a *check* action after every move action (thanks to the facts `can_move` and `has_to_check`). The precondition of the *check* action is precisely that the robot is not in a forbidden cell.

A possible solution (only listing the move actions) is presented in Figure 1.

## Presenting the Algorithm

We give a short description of our algorithm, and explain the details of each subroutine in the next sections.

**Algorithm 1** Complete Non-Deterministic Conformant Planning Algorithm

```
1: procedure ND-CPCES
2:     input: conformant planning problem ℙ
3:     output: a Plan π for ℙ, or UNSAT
4:     𝒜 := 𝒜^∅
5:     loop
6:         π := Plan(ℙ_D^𝒜)
7:         if π is ⊥ then
8:             return UNSAT                ▷ Plan can't be found
9:         end if
10:        τ := check(π, ℙ)
11:        if τ = ⊥ then
12:            return π                    ▷ Plan found
13:        end if
14:        𝒜 := Ref(𝒜, τ)
15:    end loop
16: end procedure
```

Our procedure is presented in Algorithm 1. Following the CEGAR framework, it constructs an abstraction of the conformant planning problem that it gradually refines until a solution is found or the problem is proved unsolvable. Specifically, the abstraction is a determinisation of the planning problem; the refinement is obtained by synchronising this determinised problem with a non-deterministic finite automaton (NFA) that prevents the solutions that are known to be invalid. Importantly, the resulting planning problem is deterministic, which means it can be solved with a classical planner.

The procedure starts with an empty NFA and creates the deterministic planning problem $\mathbb{P}_D^{\mathcal{A}}$, which is the synchronisation of the determinisation with the NFA. If no plan is found, it proved that there is no solution. Otherwise, it searches for an execution that invalidates the plan returns it if it is proved valid. If not, the execution is added to the NFA and a new iteration begins.

## Determinisation and Adversarial NFA

We now make explicit the elements of the algorithm. We first discuss the discretisation which allows us to use deterministic (classical) planners. Then we present the notion of adversarial NFA which is used to exclude some solutions. How this NFA is built in practice is discussed in the following section.

### One-Outcome Determinisation of $\mathbb{P}$

The determinisation of a planning problem that we are interested in is a classical planning problem obtained by arbitrarily picking only one effect from the non-deterministic effects. We call such a determinisation the One-Outcome Determinisation and formally define it as follows:

**Definition 1** (One-Outcome Determinisation)**.** *Let $\mathbb{P} = \langle A, I, G\rangle$ be a conformant planning problem and let $\mathbb{P}_D = \langle A_D, I_D, G_D\rangle$ be a deterministic planning problem defined over the same set of facts. $\mathbb{P}_D$ is a* determinisation *of $\mathbb{P}$ if $I = I_D$, $G = G_D$, and $A_D$ consists exactly of one determinisation of a ($\forall a \in A.\ \exists! a_D \in A_D.\ a_D \in O(a)$).*

Note that the set of valid plans of the resulting classical planning problem is a superset of the valid conformant plans of the original conformant problem, i.e., the one-outcome determinisation is a proper relaxation of the conformant planning problem.

**Lemma 1** ($\mathbb{P}_D$ is a relaxation of $\mathbb{P}$)**.** *Any plan of $\mathbb{P}$ is a plan of $\mathbb{P}_D$, yet there may be plans of $\mathbb{P}_D$ that are not plans of $\mathbb{P}$.*

For brevity we will refer to the One-Outcome Determinisation as just Determinisation.

### Adversarial NFA

The second element of our reduction is a finite state machine that is used to disqualify invalid plans. We write $\mathcal{A} = \langle Q, \Sigma, T, q_I, F\rangle$ a *non-deterministic finite automaton* (NFA) where

- $Q$ is a set of states, $q_I \in Q$ is the initial state, $F \subseteq Q$ is the set of *failure states*,
- $\Sigma$ is a set of labels identifying transitions,
- $T \subseteq (Q \times \Sigma \times Q)$ is the set of state transitions.

An NFA *accepts* a sequence of transition labels if such a sequence links the initial state to one of the failure states. The set of accepted sequences is represented by $\mathcal{L}(\mathcal{A})$.

Given the NFA $\mathcal{A}$ and some label $a$, $pred_{\mathcal{A},a}$ is the function that, given a state $q'$ returns its predecessor states, i.e., $pred_{\mathcal{A},a}(q') = \{q \in Q \mid \langle q, a, q'\rangle \in T\}$.

We use the NFA to represent the non-deterministic and adversarial nature of our non-deterministic planning problem. The construction of the NFA is described in the next section, but we now give a short interpretation: i) the NFA labels correspond to actions in our planning problem; ii) the failure state is used to represent the fact that an action was executed in a state that does not satisfy its precondition. So given a sequence of action names $\sigma_1, \ldots, \sigma_k$ that leads to a failure state, this sequence is an invalid plan.

**Definition 2** (Adversarial Automaton)**.** *An NFA $\mathcal{A} = \langle Q, \Sigma, T, q_I, F\rangle$ is said to be* adversarial *w.r.t. a conformant planning problem $\mathbb{P} = \langle A, I, G\rangle$ iff i) $\Sigma = \{\text{name}(a) \mid a \in A\}$ is the set of action names, ii) all failure states are sink states ($\forall q \in F.\ \forall a \in \Sigma.\ \langle q, a, q\rangle \in T$), and iii) $\mathcal{L}(\mathcal{A})$ includes only sequences of actions that are not valid in $\mathbb{P}$, i.e., $\Pi(\mathbb{P}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$.*

For now, we assume the NFA is given and focus on how we encode such an NFA using a classical planning problem formulation. In other words, we show how we can encode a classical planning problem in a way that the obtained plan is not disproved by the NFA.

Given a (classical) planning problem $\mathbb{P} = \langle A, I, G\rangle$ (for instance, $\mathbb{P}_D$) and given an NFA $\mathcal{A} = \langle Q, \Sigma, T, q_I, F\rangle$, we generate a new classical planning problem $\mathbb{P}^{\mathcal{A}} = \langle A', I', G\rangle$ whose valid plans are $\Pi(P) \setminus \mathcal{L}(\mathcal{A})$. The main idea is to i) encode each state of the automaton $q$ using a fact $p_q$, ii) disallow actions going to failure states, and iii) advancing the automaton in parallel to the action execution. The synchronisation of the planning problem with the automaton can be defined formally as the following:

**Definition 3** (Adversarial Automaton Synchronisation)**.** *Let $\mathbb{P} = \langle A, I, G\rangle$ be a planning problem and $\mathcal{A} = \langle Q, \Sigma, T, q_I, F\rangle$ be an NFA whose transition labels are the names of the actions in $\mathbb{P}$. The* adversarial automaton synchronisation *of $\mathbb{P}$ is denoted by $\mathbb{P}^{\mathcal{A}}$ and is a new classical planning problem $\langle A', I', G\rangle$ defined as the following:*

- $A' = \{a^{\mathcal{A}} \mid a \in A\}$
- $I' = I \cup \{p_{q_I}\}$

*where* $a^{\mathcal{A}} = \langle \mathrm{name}(a), \mathrm{pre}(a) \wedge \mathrm{pre}^{\mathcal{A}}(a), \mathrm{eff}(a) \cup \mathrm{eff}^{\mathcal{A}}(a)\rangle$ *such that*

- $\mathrm{pre}^{\mathcal{A}}(a) = \bigwedge_{q' \in F} \bigwedge_{q \in pred_{\mathcal{A},a}(q')} \{\neg p_q\}$
- $\mathrm{eff}^{\mathcal{A}}(a) = \bigcup_{q' \in Q} \left\{ \bigvee_{q \in pred_{\mathcal{A},a}(q')} p_q \rhd p_{q'} \right\} \cup \bigcup_{q' \in Q} \left\{ \neg \bigvee_{q \in pred_{\mathcal{A},a}(q')} p_q \rhd \neg p_{q'} \right\}$.

The definition simulates the progress in the NFA by changing the values to the predicates that represent the automaton state. Each action $a$ features a conditional effect for each state in $q \in Q$ that systematically establishes whether $a$ can reach some other state from this state or not (i.e. $\bigcup_{q' \in Q} \left\{ \bigvee_{q \in pred_{\mathcal{A},a}(q')} p_q \rhd p_{q'} \right\}$). If this is indeed the case, the successor $q'$ through $a$ is set to true (it suffices to have just one state satisfying this); yet this does not make $q$ false. The only way to make a state $q'$ not part of the belief of the agent is when there is no transition from some other state $q$ to it. This last aspect is encoded by using the second set of conditional effects (i.e., $\bigcup_{q' \in Q} \left\{ \neg \bigvee_{q \in pred_{\mathcal{A},a}(q')} p_q \rhd \neg p_{q'} \right\}$). Each of these conditional effects indeed ensures that there is no way of making the relative $p_q$ hold in the next state.

**Lemma 2.** *The plans in $\Pi(\mathbb{P}^{\mathcal{A}})$ are solutions for $\mathbb{P}$, and are not accepted by the NFA $\mathcal{A}$. That is $\Pi(\mathbb{P}^{\mathcal{A}}) = \Pi(\mathbb{P}) \setminus \mathcal{L}(\mathcal{A})$.*

*Proof Sketch.* (Reductio ad absurdum) Let $\pi = \langle a_0, \cdots, a_n\rangle$ be a plan solution for $\mathbb{P}^{\mathcal{A}}$ and assume this sequence can be mapped into a failing trajectory of the NFA $A$. From $\pi$ we can extrapolate a sequence of states in $\mathcal{A}$ such that $\langle s_0, \cdots, s_m\rangle$ with $m \leq n+1$ and $s_m = \bot$. Note that, by construction, $\mathbb{P}^{\mathcal{A}}$ keeps track of any transition occurring in the NFA. Therefore, the only way for this to happen is because the action applied in $s_{m-1}$ had its precondition not satisfied. And this obviously is a contradiction with $\pi$ being a solution for $\mathbb{P}^{\mathcal{A}}$. □

## Incremental Adversarial NFA Construction

In this section, we discuss how the adversarial automaton can be constructed from a set of counter-examples. We then show how this adversarial automaton can be built incrementally, until it is precise enough that it allows us to solve the conformant planning problem.

**Definition 4** (Counter-example)**.** *A* counter-example *for $\mathbb{P}$ is a sequence of states actions $\tau = q_0 \xrightarrow{a_1} \ldots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} \bot$ such that:*

- *$q_0$ is the initial state,*
- *each action $a_i$ is applicable in $q_{i-1}$ for $i < k$,*
- *$q_0, \ldots, q_{k-1}$ is an execution of $a_1, \ldots, a_{k-1}$,*
- *the action $a_k$ is not applicable in $q_{k-1}$.*

*If $a_1, \ldots, a_k$ is a prefix of a plan $\pi$, we also say that $\tau$ is a counter-example to $\pi$.*

**Definition 5.** *Let $\mathbb{P} = \langle A, I, G\rangle$ be a planning problem, and let $\mathcal{C} = \{\tau_1, \ldots, \tau_\ell\}$ be a set of counter-examples of $\mathbb{P}$. The* counter-automaton *for $\mathbb{P}$ based on $\mathcal{C}$ is the NFA $\mathcal{A}^{\mathcal{C}} = \langle Q, \Sigma, T, q_I, F\rangle$ defined as:*

- *$Q = \{I, \bot\} \cup \bigcup_{\tau \in \mathcal{C}} states(\tau)$, where $states(\tau)$ is the set of states in the counter-example $\tau$;*
- *$\Sigma = \{\mathrm{name}(a) \mid a \in A\}$;*
- *$T = \{\langle \bot, \mathrm{name}(a), \bot\rangle \mid a \in A\} \cup \bigcup_{\tau \in \mathcal{C}} trans(\tau)$ where $trans(\tau)$ is the set of transitions in the counter-example $\tau$;*
- *$q_I = I$; and $F = \{\bot\}$.*

While the language $\mathcal{L}(\mathcal{A}^{\mathcal{C}})$ includes all the words that label the counter-examples in $\mathcal{C}$, it generally contains additional elements when the same states are visited multiple times (in the same counter-example or in different ones). Still, one can prove that all the words in the language of the counter-automaton are adversarial:

**Lemma 3.** *A counter-automaton for $\mathbb{P}$ is an adversarial automaton wrt $\mathbb{P}$.*

*Proof Sketch.* Consider a trajectory $q_0 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_k} q_k \xrightarrow{\sigma_{k+1}} \bot$ from the NFA. This trajectory is one of the possible executions of $\langle \sigma_1, \ldots, \sigma_{k+1}\rangle$ since every transition $q_{i-1} \xrightarrow{\sigma_i} q_i$ appears in one of the counter-examples from $\mathcal{C}$. This execution ends in $\bot$, which proves that the execution is not valid. QED. □

Interestingly, the counter-automaton $\mathcal{A}^{\mathcal{C}}$ can be built iteratively, starting with $\mathcal{A}^{\emptyset}$, and adding the states and transitions of each counter-example incrementally. Given the counter-automaton $\mathcal{A}^{\mathcal{C}}$ and a counter-example $\tau$ that is not represented by the language $\mathcal{L}(\mathcal{A}^{\mathcal{C}})$, we call *refinement*, denoted $Ref(\mathcal{A}^{\mathcal{C}}, \tau)$, the operation that computes the NFA $\mathcal{A}^{\mathcal{C} \cup \{\tau\}}$. This operation is indeed a refinement if the counter-automaton is considered as an abstraction of the NFA that represents the invalid plans. Notice that it is not possible to perform an arbitrary number of refinement to the NFA, as each refinement adds at least one transition (and the number of transitions is $O(|S|^2 \times |A|)$).

### The Check Procedure: Finding Counter-examples

The procedure to find counter-examples uses a translation to SAT. The SAT encoding collocates states and actions along a discrete timeline that starts with index 0, that is the initial state, and terminates with the last state produced by the last action. The role of the SAT encoding is to find an assignment for the action effects that makes one of the actions' precondition unsatisfied.

Our SAT encoding is based on a series of substitutions and manipulations of the formulas belonging to the action preconditions, and effect. To frame a given SAT variable to a particular instant of time, we substitute each fact $f$ in some

formula with the SAT variable whose name is obtained by concatenating $f$ with $@(i)$. We attach the symbol $@(i)$ to any formula where this substitution is performed.

Let $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ be the plan to be checked. Our encoding uses SAT variables $V = \{f@i \mid f \in F, i \in \{0, \ldots, n\}\}$ and $D = \{d^{c,e}@i \mid i \in \{0, \ldots, n-1\}, \langle c, E \rangle \in \text{eff}(a_i), e \in E\}$. The SAT variable $d^{c,e}@i$ indicates that if $c$ is satisfied in the $i$th state, then $e$ is the set of effects that are executed as part of the non-deterministic effects (if $c$ is not satisfied, the SAT variable is simply ignored). Then we denote with $ach(\ell, i)$ what needs to hold in order for literal $\ell$ (positive or negative) to be achieved in the next step:

$$ach(\ell, i) = \bigvee_{\substack{c,e \text{ such that} \\ \exists E.\ (c,E) \in a_i \wedge e \in E \wedge \ell \in e}} \big(c@i \wedge d^{c,e}@i\big).$$

Similarly to other works in finding counter-examples via SAT (Grastien and Scala 2020), our encoding is based on 3 main axioms: the initial state axiom ($IA$), the frame axiom ($FA$), and the effect axiom ($EA$). Let $\mathbb{P} = \langle A, I, G \rangle$ be our non-deterministic conformant planning problem, the axioms are shaped as following:

$$IA = \bigwedge_{f \in I} f@0 \ \wedge \bigwedge_{f \in F \setminus I} \neg f@0$$

$$FA = FA^+ \wedge FA^- \text{where}$$

$$FA^+ = \bigwedge_{\substack{i \in \{0,\ldots,n\} \\ f \in F}} \{(\neg f@(i) \wedge f@(i+1)) \rightarrow ach(f, i)\}$$

$$FA^- = \bigwedge_{\substack{i \in \{0,\ldots,n\} \\ f \in F}} \{(f@(i) \wedge \neg f@(i+1)) \rightarrow ach(\neg f, i)\}$$

$$EA = \bigwedge_{\substack{i \in \{0,\ldots,n\} \\ f \in F}} ach(f, i) \rightarrow f@(i+1) \wedge ach(\neg f, i) \rightarrow \neg f@(i+1)$$

Intuitively the formula looks for some allocation of the $d$ variables for all actions in the plan that makes some precondition unsatisfied. This is represented by $UP = \bigvee_{i \in \{1,n\}} \neg \text{pre}(a_i)@i$.

**Lemma 4.** *There exists a counter-example to the plan $\pi$ iff the SAT formula described $IA \wedge FA \wedge EA \wedge UP$ is satisfiable.*

The lemma derives naturally from the definition of a counter-example: Given a counter-example, it is easy to see to derive a satisfying assignment to the formula. Conversely, from a satisfying assignment, it is easy to derive a counter-example.

## Correctness of the Algorithm

Our procedure has already been presented in Algorithm 1. We discuss its correctness.

**Theorem 1** (Soundness and Completeness)**.** *If the underlying classical planner is sound and complete, and the check procedure is sound, Algorithm 1 produces a valid conformant plan when there is one, and terminates with UNSAT after a finite number of steps otherwise.*

*Proof Sketch.* First notice that if Algorithm 1 returns a plan, then this plan is valid since it passed the check (Lemma 4).

From Lemmas 1, 2, and 3, we know that the $\mathbb{P}_D^{\mathcal{A}}$ is a relaxation of $\mathbb{P}$, which means that Algorithm 1 returns UNSAT only when there is no valid plan for $\mathbb{P}$.

Finally, each step of the iteration refines the counter-automaton, which, as we already noticed, can only be done a bounded number of times. Therefore, the procedure always terminates given enough time.

□

## Decomposing Automaton States through Contexts

As we have seen in the previous section, the number of iterations of our algorithm is bounded by the number of trajectories that can prove our classical planning attempts failing. Albeit finite, this number can be quite large. In order to tackle this problem, this section show how to use the structure of the planning problem (in particular, through the notion of context (Bonet and Geffner 2014)) to improve the convergence.

**Definition 6.** *The* context *$ctx(f)$ of a fact $f$ is the smallest set that satisfies:*

- *$f \in ctx(f)$;*
- *For all $f' \neq f$, if there is some action $a$ with conditional effect $\phi \rhd E \in \text{eff}(a)$, such that i) $f'$ appears in $\phi$ and ii) there exists $e \in E$ where either $f \in e$ or $\neg f \in e$, then $f' \in ctx(f)$;*
- *If $f'' \in ctx(f')$ and $f' \in ctx(f)$ then $f'' \in ctx(f)$.*

Given a counter-example, it is possible to determine the context of the precondition that was not satisfied. We can then project the counter-example on the context, and define a counter-example for each automaton.

In practice, if the counter-example $\tau$ is $\tau = q_0 \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} \bot$, and the precondition of the last action is $\text{pre}(a_k) = \varphi_1 \wedge \cdots \wedge \varphi_\ell$ (where $\ell$ may equal 1), let $i$ be one of the indices such that $q_{k-1}$ does not satisfy $\varphi_i$. Then the context of this precondition is $ctx(\varphi_i) = \bigcup_{f \in \mathcal{V}(\varphi_i)} ctx(f)$ where $\mathcal{V}(\varphi_i)$ is the list of facts of $\varphi_i$.

The *projection* of $\tau$ on the context of $\varphi_i$ is $ctx(\varphi_i) \cap q_0 \xrightarrow{a_1} \ldots \xrightarrow{a_{k-1}} ctx(\varphi_i) \cap q_{k-1} \xrightarrow{a_k} \bot$. It is possible to maintain a counter-automaton for each context and to add the projected counter-example onto the right counter-automaton.

**Example** Consider the example of TRICKYGRID displayed on Figure 1. Assume that the candidate plan is DUG (down, up, then goal action). This plan is invalid since, e.g., when starting in position $\langle 2, 2 \rangle$ (bottom-left-most possible starting location), the robot is unable to perform the two conditions of the G action ($x$ should be 3, is currently 2; $y$ should be 3, is currently 2). If we focus on the precondition $\varphi = x(2)$, the context of $\varphi$ is $\{x(1), \ldots, x(5)\}$ since the $y$ position never affects the $x$ position of the robot (that would

be different if there were walls for instance). The counter-example is therefore

$$\{x(2), y(2)\} \xrightarrow{D} \{x(2), y(1)\} \xrightarrow{U} \{x(2), y(2)\} \xrightarrow{G} \bot,$$

and its projection on $\{x(1), \ldots, x(5)\}$ is

$$\{x(2)\} \xrightarrow{D} \{x(2)\} \xrightarrow{U} \{x(2)\} \xrightarrow{G} \bot.$$

The projected counter-example is much more useful than the non-projected one for two simple reasons. First, it helps generalising the result. In particular, the counter-example cannot be avoided by simply using more D and U actions: it is necessary to perform other actions, such as L and R. The second benefit is that fewer additional facts $f_q$ need to be introduced when synchronising the adversarial automaton with the deterministic domain.

## Related Work

There are mainly two approaches to solve conformant planning problems that have been investigated so far. The first approach casts the problem as a search over the belief state (Bonet and Geffner 2000) that is carried out by progressing (or regressing) a frontier of beliefs each given by a particular choice of actions (e.g., (Cimatti, Roveri, and Bertoli 2004; Hoffmann and Brafman 2006; To, Son, and Pontelli 2015; Albore, Ramírez, and Geffner 2011)). The second is a translation based approach, i.e., it transforms the problem into one which is easier to solve (Albore, Palacios, and Geffner 2010; Nguyen et al. 2012; Palacios and Geffner 2007).

A direct search over the belief space has to deal with the problem of efficiently representing and manipulating the belief. In this regard, several approaches have been proposed, among which Binary Decision Diagrams (Cimatti, Roveri, and Bertoli 2004; Bryant 1986) and SAT formulations (Hoffmann and Brafman 2006; Albore, Ramírez, and Geffner 2011). Among these works, to the best of our knowledge, only the system presented by Cimatti, Roveri, and Bertoli (2004) supports non-deterministic effects.

Translation-based approaches can take the powerful reasoning techniques developed for solving a specific targeted problem directly off-the-shelf; however, this comes at the cost of providing guarantees, such as completeness and soundness. In particular, completeness can usually be ensured only under severe restrictions on the structure of the problem (Palacios and Geffner 2007; Albore, Palacios, and Geffner 2010). There exist several translations from conformant to classical planning and from non-deterministic conformant to (deterministic) conformant planning. Yet, the only translation that supports the use of non-deterministic conformant is the one described by Albore, Palacios, and Geffner (2010). The main idea pursued by the authors is that of anticipating the non-deterministic choices of the actions into uncertainty in the initial belief of the agent. The authors present different encodings starting from this idea, but none of them is complete. Indeed, it is difficult to anticipate how many non-deterministic choices the agent will need to deal with; and this problem is related to the difficulty of anticipating reasonable upper-bounds on the number of action occurrences in a plan. Our work is similar in the spirit to the work by Albore, Palacios, and Geffner (2010): we also perform a translation from one problem to the other. Yet, we translate the problem directly into a classical planning formulation. This allows us a much finer control of the implicit belief state, and results in a schema that can be proved not only sound but also complete.

Another related approach to our technique, is the CPCES planning system (Grastien and Scala 2020). They provide a counter-example guided mechanism to incrementally solve the conformant planning problem, but restrict themselves to the case with only deterministic effects. Our approach extends it in a substantial manner. We indeed propose a novel reduction to classical planning, and a novel notion of a counter-example, with profound implications. Our counter-examples are plan executions, not just initial states.

As us, other people have used automata in planning, but with quite a different purpose. For instance, the work by Torres and Baier (2015); Patrizi, Lipovetzky, and Geffner (2013) do so for synchronising the planning model with LTL formulas. The main difference with us is that we build the automaton from counter-examples, i.e., we do not require to provide any user specifications that need to be met.

In the much more general problem of planning under uncertainty, related are all the extensions that assume some form of observability of the agent state, e.g., (Muise, Belle, and McIlraith 2014; Geffner and Geffner 2018) or works that rely on an online replanning step (Yoon, Fern, and Givan 2007; Scala and Torasso 2015). In particular, the work by Geffner and Geffner (2018) uses a SAT encoding to find compact policies in a fully-observable scenario. Understanding synergies between this SAT formulation and ours to extend the reach of our approach is definitely interesting for future work. Replanning methods use determinisation as we do although completeness in that case can only be ensured under very restrictive assumptions, e.g., ergodic state-space (Brafman and Shani 2012).

The work by Bonet and Geffner (2014) has been also of great inspiration for us. The authors indeed propose the notion of contexts for planning to make explicit potentially independent sets of variables. Indeed, we exploit to great extent this definition, and use it as a means to simplify belief tracking problem in our system, too. The state space of our adversarial automaton can indeed be made smaller exactly for the reason that we do need to consider all the combinations of variable values, but can decompose the state space in a context-based manner. As we will see in the experiments, this greatly speeds up our approach.

## Implementation and Experimental Analysis

We implemented the ideas of this paper in a new non-deterministic conformant planner. Our planner makes use of two components: an SMT solver playing the role of our SAT solver[5], and as a classical planner a PDDL Planner. We tried several SMT solvers, among which MathSat (Cimatti, Roveri, and Bertoli 2004) and Z3 (De Moura and Bjørner

[5]We use SMT as we find the language supported by many SMT planners more convenient to use for our purposes. We use PYSMT (Gario and Micheli 2015) to wrap the SMT solvers.

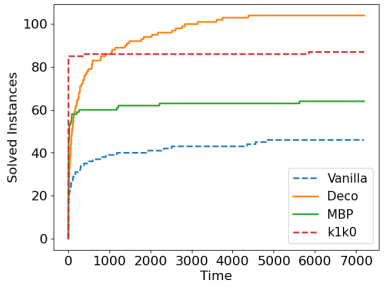


Figure 2: Number of solved instances over time.

| Domain | I | Contexts | Complete Methods VANILLA | DECO | MBP | k1k0 |
|---|---|---|---|---|---|---|
| UTS | 3 | (9-15) | 0 | 1 | 1 | **3** |
| COINS | 3 | (21-45) | 0 | 2 | 2 | **3** |
| TRAIL | 3 | (2) | 1 | **3** | 0 | 0 (3*) |
| MOVE-PKG | 4 | (18-31) | 3 | **4** | **4** | 0 (4*) |
| BOMB-1-T | 40 | (2) | 21 | **40** | 19 | **40** |
| BOMB-N-T | 40 | (4) | 14 | **40** | 17 | **40** |
| MOUSE-CAT | 3 | (1925-7845) | **3** | **3** | 0 | 2 |
| TRICKYGRID | 24 | (5) | 4 | 12 | **21** | 0 |
| Total | 120 | | 46 | **105** | 64 | 88 (95*) |

Table 1: Coverage Analysis. ''Bold'' stands for best performer. ''I'' stands for number of instances, ''Contexts'' stands for number of contexts using (min,max) when they differ across the instances of a given domain.

2008). We opted for MathSat as it turned out to be the most efficient solution for our problems. We also investigated a number of PDDL planners, among which FF, LAMA, BFWS and Madagascar (Hoffmann and Nebel 2001; Rintanen 2014; Richter and Westphal 2010; Lipovetzky and Geffner 2017). Despite being the oldest among such systems, FF was the only solution that worked well and robustly with our expressive formulations.

Our system reads as an input a PDDL description of the problem as for the specification of the last IPC on planning under uncertainty[6]. The interaction with the classical planner is also done using PDDL, of which we use many powerful constructs in the action representation. Recall that a PDDL problem is defined compactly using two different representations: the domain theory and the planning instance. We used both universal and existential quantifiers in the action description, and this gave us the possibility to define the domain file just once; the actual structure of the actions in our implementation is in fact determined by the objects and the variables instantiated into the problem file.

Our experimental analysis aims at understanding whether the approach that we have presented in this paper can be practical. To verify this we tested both the version without the context-decomposition, hereinafter called VANILLA, and the version with it, hereinafter called DECO. We measured: coverage as the number of instances solved by the system, the number of iterations done by the CEGAR loop before finding a solution, and run-time. Quality wise, we measured the length of the resulting plans. Note that the time for both VANILLA and DECO is the time spent by the SMT solver, the PDDL planner and all the various modules. The system is implemented in Python, and can be downloaded at `https://bit.ly/38na9Rq`.

As a comparative analysis, we considered another provably complete conformant planner, that is the Model Based Planner by Cimatti, Roveri, and Bertoli (2004), and the fastest translation-based approach by Albore, Palacios, and Geffner (2010), that is the k1k0 translation. The k1k0 translation uses FF as classical planner, as in our case.

Our benchmark suite encompasses domains and instances from the work by Albore, Palacios, and Geffner (2010). To the best of our knowledge, these are the more recent benchmark domains. For the sake of fairness, all the planners used in our comparison run over the very same problem representation, which is, as hinted at above, written in non-deterministic PDDL[7]. For BOMB-1-TOILET and BOMB-N-TOILET we generated smaller instances (the larger ones were only solvable by the k1k0). These instances scale on the number of bombs, from 1 to 40. For the BOMB-N-TOILET version, which uses multiple toilets, we fixed the number of toilets to 3. All the other instances are exactly the same problems used by Albore, Palacios, and Geffner (2010). Ultimately, we remove from our comparison both GRIPPER and the NON-DET-RING domain because none of the planners was able to solve any instance. We also consider the new domain TRICKYGRID introduced earlier.

All the experiments have run on an Intel Xeon Gold 6140M CPUs with 2.30 GHz. For each instance we set a cut-off time of 7200 seconds, and memory was limited to 8 GB.

**Comparative Analysis Results**. Table 1 reports the number of instances solved by all the systems under comparison, on a per domain basis. As it is possible to observe, only the techniques that exploit, to some extent, the structure of the problem, k1k0 and DECO, are able to scale-up over the instances from BOMB-1-TOILET and BOMB-N-TOILET. The advantage of coverage of DECO over VANILLA is pronounced even when the decomposition induced by the contexts is not prominent. For instance, in both the versions of BOMB-1-TOILET and BOMB-N-TOILET, DECO scales much better than VANILLA even though we have only 2 and 5 contexts, respectively (Figure 3). Interestingly, DECO translation takes the advantage of both a complete approach and that of an approach that exploits the power of classical planners. Indeed, it managed to solve all instances of TRAIL and MOVE-PKG, where instead k1k0 was not able to provide any solution. Actually k1k0 reports that all these instances unsolvable, and this can unfortunately be expected because k1k0 is not complete in general. On the other hand, when k1k0 solves a problem, it is order of magnitude faster than all the other systems under comparison. This is evident if we

[6]https://ipc08.icaps-conference.org/probabilistic/wiki/

[7]This way, our experimental findings depart from what observed by Albore, Palacios, and Geffner (2010). In their experiments (personal communication by Alexandre Albore) all planners make use of a custom problem representation. More precisely, MBP is run with its native language, and several domains used by the k1k0 translation delegate to some handcrafted special action the reasoning over the uncertain variables.

| | | Cpu-Time | | Iterations | | Plan Length | | NFA States | |
|---|---|---|---|---|---|---|---|---|---|
| Domain | I | VANILLA | DECO | VANILLA | DECO | VANILLA | DECO | VANILLA | DECO |
| TRAIL | 1 | 4556.48 | 568.45 | 105.00 | 10.00 | 149.00 | 198.00 | 448.00 | 162.00 |
| MOVE-PKG | 3 | 775.82 | 94.42 | 233.33 | 96.00 | 14.33 | 14.67 | 103.33 | 41.67 |
| BOMB-1-TOILET | 21 | 320.49 | 11.60 | 93.33 | 73.33 | 22.00 | 22.00 | 20.00 | 11.90 |
| BOMB-N-TOILET | 14 | 595.88 | 20.44 | 751.07 | 141.14 | 15.00 | 15.00 | 51.71 | 12.07 |
| MOUSE-CAT | 3 | 375.16 | 367.92 | 0.00 | 0.00 | 57.00 | 57.00 | 0.00 | 0.00 |
| TRICKYGRID | 4 | 880.61 | 40.74 | 138.75 | 68.25 | 40.50 | 42.00 | 107.50 | 66.75 |

Table 2: Focus on VANILLA and DECO, over intersection of instances solved by both configurations. Column 'I' indicates the number of instances under consideration. All results, for each indicator, are an average of the performance.

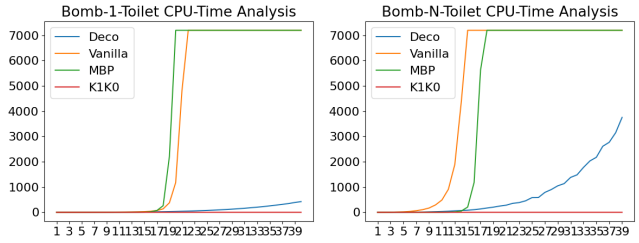


Figure 3: BOMB-1-TOILET (left), BOMB-N-TOILET (right). Instances scale with the number of bombs. We fix to 3 the number of toilets in the multiple toilets instances.

have a look at the survival plot presented in Figure 2, and the instance by instance analysis over BOMB-1-TOILET and BOMB-N-TOILET (Figure 3). Interesting is also the comparison between the VANILLA schema and MBP. Indeed, they seem to scale quite similarly, and to some extent this is also expected. Both VANILLA and the MBP techniques make few assumptions on the structure of the problem. Ultimately, Table 1 provides information (next to a few domain, in parenthesis) on how many instances would be solved by k1k0 with extra-handcrafted knowledge in the PDDL file. Without that information, k1k0 does not manage to solve any instance in those domains.

Our results should be nevertheless be considered with extremely care. On the one hand, MBP is a much more general planning system, which handles far beyond only non-deterministic conformant planning problem. And on the other hand, the k1k0 is known to be incomplete, so the zero coverage in several domains are not due to the technique per se, but to the system that employs it. It would be interesting to see as a future work, how these three quite different approaches can be put in harmony in a unified approach aimed at really getting the best of all these three worlds.

**In-Depth: VANILLA vs DECO**. Our experimental analysis concludes with an in-depth look at the performance of our two translations. In particular, Table 2 reports averaged performance over those instances that have been solved by our compilation-based planner with both the VANILLA and the DECO setting. Obviously we focus on domains where there is at least one instance solved by both configurations. As it is possible to observe, the technique that exploits the decomposition of the automaton through contexts is able to decrease substantially the number of iterations done by Algorithm 1; this indeed correlates with an average smaller number of automaton states that are needed to solve the problem. Less iterations translate in speed-ups up to one order of magnitude. We also notice that the plans are similar, except for TRAIL where the decomposition leads the planner to find a plan which is substantially longer. Interesting are the results for domain MOUSE-CAT. In this domain in fact, the one-outcome determinisation alone is able to find the plan at the very first iteration, so there is no need to refine the automaton and the process can terminate earlier. Therefore there is no difference between the two techniques.

## Conclusion and Discussion

In this paper, we introduce a novel approach to non-deterministic conformant planning. The main idea of the approach is to compute candidate plans based on a simplified (abstracted) domain, and to refine the domain in order to reject the invalid candidate plans. The simplified domains are classical (i.e., with no uncertainty) so that powerful classical planners can be used to generate the candidate plans.

The experimental evaluation shows that our approach is a nice trade-off between two existing types of planners: the planners that search over the belief space and the planners that use the structure of the problem.

This approach can be further extended, and we work in this direction. We have made many decisions that need to be explored. For instance, we select the one-outcome determinisation at random. However, in order to avoid considering plans that are "obviously" invalid, it would be beneficial to start with the "worst" possible outcome for the actions. Similarly, we presented an operation to incorporate the learnt information into the planning model (specifically, the adversarial automaton synchronisation); are there better ways to include this information? For instance, the current operation create a large number of conditional effects, which limits the list of heuristics function that can be used. We also used the `forall` and `exists` constructs from PDDL, but these constructs prevent us from using more recent planning systems such as LAMA because the pre-processor struggles to ground the model. It should be possible to bypass this issue, which would give us access to many more heuristics. Finally, there is the question of finding the "best" counter-example as suggested by Zhang, Grastien, and Scala (2020), where the best counter-example is one that makes the planning task terminate faster.

## Acknowledgements

We thank the reviewers for their great feedback.

## References

Albore, A.; Palacios, H.; and Geffner, H. 2010. Compiling Uncertainty Away in Non-Deterministic Conformant Planning. In *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 465–470. IOS Press.

Albore, A.; Ramírez, M.; and Geffner, H. 2011. Effective Heuristics and Belief Tracking for Planning with Incomplete Information. In *ICAPS*. AAAI.

Bonet, B.; and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In *AIPS*, 52–61. AAAI.

Bonet, B.; and Geffner, H. 2014. Belief Tracking for Planning with Sensing: Width, Complexity and Approximations. *J. Artif. Intell. Res.* 50: 923–970. doi:10.1613/jair.4475. URL https://doi.org/10.1613/jair.4475.

Brafman, R. I.; and Shani, G. 2012. Replanning in Domains with Partial Information and Sensing Actions. *J. Artif. Intell. Res.* 45: 565–600.

Bryant, R. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computer (TC)* 35(8): 677–691.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artif. Intell.* 159(1-2): 127–206.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Twelfth International Conference on Computer-Aided Verification (CAV-00)*, 154–169.

De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

Gario, M.; and Micheli, A. 2015. pySMT: a Solver-Agnostic Library for Fast Prototyping of SMT-Based Algorithms. In *SMT Workshop*.

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

Grastien, A.; and Scala, E. 2020. CPCES: A planning framework to solve conformant planning problems through a counterexample guided refinement. *Artif. Intell.* 284: 103271.

Haslum, P.; and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *Fifth European Conference on Planning (ECP-99)*, 308–318.

Hoffmann, J.; and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.* 170(6-7): 507–541.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14: 253–302.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *AAAI*, 3590–3596. AAAI Press.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing Contingent Plans via Fully Observable Non-Deterministic Planning. In *AAAI*, 2322–2329. AAAI Press.

Nguyen, K.; Tran, V.; Son, T. C.; and Pontelli, E. 2012. On computing conformant plans using classical planners: a generate-and-complete approach. In *22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, 190–198.

Palacios, H.; and Geffner, H. 2007. From conformant into classical planning: efficient translations that may be complete too. In *Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07)*, 264–271.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL Synthesis for Non-Deterministic Systems using Strong Cyclic Planners. In *IJCAI*, 2343–2349. IJCAI/AAAI.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.* 39: 127–177.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. *Proceedings of the 8th International Planning Competition (IPC-2014)* 21.

Scala, E.; and Torasso, P. 2015. Deordering and Numeric Macro Actions for Plan Repair. In *IJCAI*, 1673–1681. AAAI Press.

Smith, D.; and Weld, D. 1998. Conformant graphplan. In *Fifteenth Conference on Artificial Intelligence (AAAI-98)*, 889–896.

To, S. T.; Son, T. C.; and Pontelli, E. 2015. A generic approach to planning in the presence of incomplete information: Theory and implementation. *Artif. Intell.* 227: 1–51.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *IJCAI*, 1696–1703. AAAI Press.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*, 352. AAAI.

Zhang, X.; Grastien, A.; and Scala, E. 2020. Computing Superior Counter-Examples for Conformant Planning. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 10017–10024. AAAI Press.