Artificial Intelligence 352 (2026) 104482

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

# Symbolic pattern planning

Matteo Cardellini[a,*], Enrico Giunchiglia[a], Marco Maratea[b]

[a] DIBRIS, Università di Genova, Genova, Italy
[b] DeMaCS, Università della Calabria, Rende, Italy

ARTICLE INFO

*Keywords:*
Automated planning
Classical and numeric planning
Symbolic planning

ABSTRACT

In this paper, we propose a novel approach for solving automated planning problems, called Symbolic Pattern Planning. Given a deterministic planning problem Π, we propose to compute a plan by first fixing a pattern –defined as an arbitrary sequence of actions– and then define a formula encoding the state resulting from the sequential execution of the actions in the pattern, starting from an arbitrary initial state. By allowing each action in the pattern to be executed consecutively zero, one or possibly more times, and by imposing the conditions on the initial and goal states, we can check whether the pattern allows determining a valid plan or whether the pattern needs to be extended and the procedure iterated. We ground our proposal in the numeric planning setting, we prove the correctness and also the completeness of the procedure (provided at each iteration the pattern is extended with a complete sequence of actions), and we define procedures for the pattern selection and for computing quality plans. When exploiting the planning as satisfiability approach, we show that our encoding allows to determine a valid plan in a number of iterations which is never higher than the one needed by the state-of-the-art rolled-up or relaxed-relaxed-∃ symbolic encodings. On the experimental side, we run an extensive analysis which included the problems and systems involved in the numeric track of the 2023 International Planning Competition, showing that the results validate the theoretical findings and that our planner PATTY has remarkably good comparative performances.

## 1. Introduction

Automated planning is a model-based approach to the control of autonomous agents. Given a description of the possible initial states, the set of goals to achieve and the set of possible actions, the standard task is to build a strategy allowing to reach a state in which all the goals are satisfied for every possible initial state and action outcome. There are different flavours of planning, each corresponding to the allowed language, initial states, objectives, and strategies. Here we consider the standard deterministic setting in which $(i)$ there is only one initial state, $(ii)$ for each state and action there is at most one state resulting from the execution of the action in the given state, and $(iii)$ the objective is to find a finite sequence of actions whose executions from the initial state result in a state satisfying all the goals. Even with such restrictions, there are many different types of planning problems, each corresponding to the characteristics of the model. For instance, in *classical planning* the model involves only Boolean variables; in *numeric planning* variables can also take numeric values; in *temporal planning* actions are associated with a duration, and the task also involves determining the start time of each action.

* Corresponding author.
*E-mail addresses:* matteo.cardellini@unige.it (M. Cardellini), enrico.giunchiglia@unige.it (E. Giunchiglia), marco.maratea@unical.it (M. Maratea).

https://doi.org/10.1016/j.artint.2026.104482
Received 5 December 2024; Received in revised form 7 January 2026; Accepted 7 January 2026
Available online 16 January 2026

Independently of the considered type of planning, as long as a solution to a planning problem $\Pi$ is a sequence of actions; and given a pattern $\prec$ defined as a finite sequence of actions, we can compute a formula $\Pi^{\prec}$ $(i)$ whose models correspond to valid solutions of $\Pi$ (*correctness of* $\Pi^{\prec}$), and $(ii)$ which is satisfiable if there exists a subsequence of $\prec$ which is a valid solution (*completeness of* $\Pi^{\prec}$), we can solve the planning problem $\Pi$ with a simple Symbolic Pattern Planning (SPP) procedure in which we

1. initially fix a pattern $\prec$,
2. return a solution if $\Pi^{\prec}$ is satisfiable, and
3. extend the pattern and iterate the second step, otherwise.

The correctness of $\Pi^{\prec}$ ensures the correctness of the procedure: any returned solution is valid. If the pattern, upon failure, is extended with a complete pattern, i.e., with a sequence including all the available actions in $\Pi$, the procedure is also complete: if $\Pi$ admits a valid solution, one will be returned.

To ground the proposal, we focus on numeric planning problems specified in PDDL 2.1 level 2 [1], extending our previous work [2]. Given a PDDL numeric planning problem $\Pi$ and an arbitrary pattern $\prec$, we first define a Satisfiability Modulo Theory (SMT) [3] formula $\Pi^{\prec}$ which is both correct and complete, obtaining a correct and complete SPP procedure for PDDL numeric planning problems as a consequence. Then, we consider the pattern selection procedure exploited in [2] based on Asymptotic Relaxed Plan Graph (ARPG) [4], and we improve it to produce patterns which allow the SPP procedure to return a solution in at most the same number of iterations needed when using the original pattern. Finally, we show that our approach may initially yield low-quality solutions, which can be enhanced either by directly searching for higher-quality solutions of $\Pi^{\prec}$ or by refining an initial, possibly low-quality, solution.

Our SPP approach differs from existing symbolic methods that rely on the planning as satisfiability approach [5]. In these methods, a solution is found by: $(i)$ constructing a logical model that captures how actions cause a single transition from one state to another, $(ii)$ defining a *bound* $n \geq 0$ and representing trajectories with $n$ state transitions by duplicating the single transition model $n$ times, and $(iii)$ iteratively checking for the existence of a solution, starting with $n = 0$ and incrementing $n$ after each failure. Our encoding can be considered a generalization and an improvement of the state-of-the-art rolled-up encoding $\Pi^{R}$ proposed in [6] and of the relaxed-relaxed-∃ encoding $\Pi^{R^2\exists}$ [7,8], both exploited in the planning as satisfiability approach. In particular, we prove that our encoding *dominates* both $\Pi^{R}$ and $\Pi^{R^2\exists}$: for any bound $n$, it is never the case that the latter two allow to find a valid plan for $\Pi$, while ours does not. The SPP approach has also been extended in a recent work for temporal planning [9], the fragment of planning where actions may have a duration, are executed concurrently over time, and can affect Boolean and numeric variables at both the start and end of their execution.

In the search-based planning literature, several planners exploit actions sequences, first searching for a goal state employing the full sequence, and then resorting back to single actions if unsuccessful. The classical planner YAHSP [10] employs “look-ahead plans” (i.e., sequences) in the forward search trying to jump to intermediate states closer to the goal. This is similar to the concept of macro-actions [11]. Patterns, however, allow capturing a larger superset of sequences than macro-actions and “look-ahead plan”, since actions in any position of the pattern may not be selected in the plan, while the latter approaches only employ the full sequence. Moreover, in numeric planning, the planner needs also to consider how many times a single action in the sequence has to be consecutively applied (i.e. rolled) which is standard in the SPP approach. Conclusively, all the above-mentioned approaches are implemented only for search-based approaches, while our work moves forward the state-of-the-art for satisfiability-based planning. On the modelling side, Bonassi et al. [12] introduces the concept of “planning with actions constraints” (PAC) for PDDL3, where one can specify *action-trajectory constraints* on the final plan directly in the planning task. Through PAC, one could specify, for example, the expected order of actions in the plan, helping to guide the search for a plan. While this approach is not connected with our ideas of patterns (the plan could be drastically different from the pattern and our procedure would still be complete), it shows that, in many domains, providing an intuition on the order of the actions can be very beneficial.

To experimentally validate the results and show the effectiveness of our proposal, we $(i)$ considered the 2 planners, benchmarks, and settings of the 2023 International Planning Competition (IPC), Agile track [13]; $(ii)$ added 4 other publicly available planning systems for numeric problems; and $(iii)$ considered various versions of our system differing either for the pattern selection procedures and/or for the quality of the returned plan. Overall, our comparative analysis included 7 other planners, 4 of which symbolic and 3 search-based. The experimental results indeed validate our theoretical findings and show that, compared to the other symbolic planners, our planner PATTY has always better performance on every domain, while compared to all the other planners, PATTY has overall remarkably good performances, being the fastest system able to solve most problems on the largest number of domains.

Also based on these results, we believe that our proposal provides a new starting point for symbolic approaches to planning: a pattern $\prec$ can be *any* sequence of actions (even with repetitions) and the pattern needed to effectively solve the problem (e.g., the plan itself) can be symbolically searched and incrementally defined, aiming to more complex SPP procedures bridging the gap between symbolic and search-based planning.

Summarizing, the main contributions of the paper are:

1. we present a novel approach for planning in deterministic domains, that we call Symbolic Pattern Planning, and ground our proposal to numeric planning problems formalized in PDDL 2.1 level 2,
2. we consider various pattern selection procedures and mechanisms for improving the quality of the returned solution,
3. we compare our work to existing planning as satisfiability approaches to numeric planning, showing that our encoding can be considered a generalization of the state-of-the-art rolled-up and $R^2\exists$ encodings, and

4. we experimentally validate our ideas and show that our planner PATTY outperforms the other available numeric planners on the benchmarks of the 2023 IPC.

The above contributions are based on and extend our previous work [2]. Differently from this paper, in [2] $(i)$ we did not explicitly introduce Symbolic Pattern Planning and adopted a planning as satisfiability approach, $(ii)$ we did not study the impact of the pattern selection procedure and mechanisms for improving the quality of the returned plan, and $(iii)$ we performed a more limited experimental analysis.

The paper is structured as follows. After the preliminaries on how to define a numeric planning problem $\Pi$ in PDDL 2.1 and the main concepts behind Satisfiability Modulo Theories (Section 2), we present our SPP encoding, proving how it allows defining correct and complete procedures for $\Pi$ in Section 3. In the same section, we present the outlined pattern selection procedures and address the plan quality problem, respectively. In Section 4 we frame our encoding in the planning as satisfiability approach, and show that our encoding provably dominates the rolled-up and $R^2\exists$ encodings. We end the paper with the experimental analysis and the conclusions. One running example is used throughout the paper to illustrate the formal concepts introduced in the paper.

## 2. Preliminaries

### 2.1. Numeric planning in PDDL 2.1

As briefly outlined in the introduction, there are many languages for specifying planning problems. Here, we specifically consider *numeric planning problems* specified in PDDL 2.1, level 2 [1], standardly defined as a tuple $\Pi = \langle V_B, V_N, A, I, G\rangle$ in which [1]

1. $V_B$ and $V_N$ are finite sets of *Boolean* and *numeric state variables* with domains $\{\top, \bot\}$ for truth and falsity, and the set $\mathbb{Q}$ of rational numbers, respectively;
2. $A$ is a finite set of actions. An *action* $a$ is a pair $\langle \mathrm{pre}(a), \mathrm{eff}(a)\rangle$ in which
   (a) $\mathrm{pre}(a)$ is the union of the sets of *propositional* and *numeric preconditions* of $a$, the former of the form either $v = \top$ or $v = \bot$ and $v \in V_B$, the latter of the form $\psi \trianglerighteq 0$, with $\trianglerighteq \in \{\geq, >, =\}$ and $\psi$ a *linear expression* over $V_N$, i.e., with $\psi$ equal to $\sum_{w\in V_N} k_w w + k$, for some $k_w, k \in \mathbb{Q}$; and
   (b) $\mathrm{eff}(a)$ is the union of the sets of *propositional* and *numeric effects*, the former of the form $v := \top$ or $v := \bot$, the latter of the form $w := \psi$, with $v \in V_B$, $w \in V_N$ and $\psi$ a linear expression.

   We assume that for each action $a$ and variable $v \in V_B \cup V_N$, $v$ occurs in $\mathrm{eff}(a)$ at most once to the left of the operator "$:=$", and when this happens we say that $v$ is *assigned* by $a$. As customary, we write
   (a) $v \mathrel{+}= \psi$ as an abbreviation for $v := v + \psi$ (and similarly for $v \mathrel{-}= \psi$), and
   (b) $\psi < 0$ as an abbreviation for $-\psi > 0$, and similarly for $\psi \leq 0$.
3. $I$ is the *initial state* mapping each variable in $V_B \cup V_N$ to an element in its domain, and $G$ is a finite set of *goal formulas*, each one being a propositional combination of propositional and numeric conditions. Indeed, the set $G$ is interpreted as the conjunction of the formulas in it.

Let $\Pi = \langle V_B, V_N, A, I, G\rangle$ be a numeric planning problem. A *state* $s$ maps each variable $v \in V_B \cup V_N$ to a value $s(v)$ in its domain, and we assume the domain of each state is extended to linear expressions, Boolean/numeric conditions and their propositional combinations in the standard way. An action $a \in A$ is *executable in a state* $s$ if $s$ satisfies all the preconditions of $a$. Given a state $s$ and an executable action $a$, the *result of executing* $a$ *in* $s$ is the state $s' = res(a, s)$ such that for each variable $v \in V_B \cup V_N$,

1. $s'(v) = \top$ if $v := \top \in \mathrm{eff}(a)$, $s'(v) = \bot$ if $v := \bot \in \mathrm{eff}(a)$, $s'(v) = s(\psi)$ if $(v := \psi) \in \mathrm{eff}(a)$, and
2. $s'(v) = s(v)$ otherwise.

Consider a finite sequence of actions $\alpha = a_1; \ldots; a_n$ with $n > 0$. The *state sequence* $s_0; \ldots; s_n$ *induced by* $\alpha$ *in* $s_0$ is such that for $i \in [0, n)$, the state $s_{i+1}$

1. is undefined if either $a_{i+1}$ is not executable in $s_i$ or $s_i$ is undefined, and
2. is $res(a_{i+1}, s_i)$, the result of executing $a_{i+1}$ in $s_i$, otherwise.

If $s_n$ is defined, we say that

1. $\alpha$ is *executable in* $s_0$,
2. $s_n$ is the *result of executing* $\alpha$ *in* $s_0$, which will be denoted also with $res(\alpha, s_0)$, i.e.,

$$s_n = res(\alpha, s_0) = res(a_n, res(a_{n-1}, res(\ldots res(a_1, s_0)\ldots))).$$

We extend the definition to the case $n = 0$, where $\alpha$ reduces to the empty sequence $\epsilon$. We define that in any state $s$, the empty sequence $\epsilon$ is executable in $s$ and $res(\epsilon, s) = s$. Finally, if $res(\alpha, I)$ is defined and satisfies the goal formulas in $G$, we say that $\alpha$ is a *(valid) plan*.

[1] The PDDL language allows for a lifted representation with variables defined over a finite domain. Here, we consider the grounded version in which variables are replaced with the elements in the domain in all possible ways.

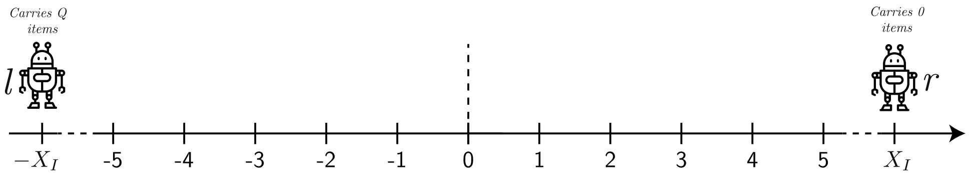

**Fig. 1.** The initial condition of the motivating example of this paper, as described in Example 1.

**Example 1.** As depicted in Fig. 1, there are two robots $l$ and $r$ for left and right, respectively, whose position $x_l$ and $x_r$ on an axis correspond to the integers $\leq 0$ and $\geq 0$, respectively. The two robots can move to the left or to the right, decreasing or increasing their position by 1. The two robots carry $q_l$ and $q_r$ objects, which they can exchange. However, before exchanging objects, the two robots must connect, setting a Boolean variable $p$ to $\top$, and this is possible only if they have the same position. For the sake of simplicity, we impose that they exchange only one object at a time, and, thus, a variable $q$, which can either be $+1$ or $-1$, corresponds to $l$ giving objects to $r$ or vice versa, respectively. Once connected, they must disconnect before moving again. This scenario can be modelled in PDDL 2.1 with $V_B = \{p\}$, $V_N = \{x_l, x_r, q_l, q_r, q\}$ and the following set of actions:

$$\begin{aligned}
\texttt{lftr} &: \langle \{x_r > 0\}, \{x_r \mathrel{-}= 1\}\rangle,\ \texttt{rgtr} : \langle \{p = \bot\}, \{x_r \mathrel{+}= 1\}\rangle,\\
\texttt{lftl} &: \langle \{p = \bot\}, \{x_l \mathrel{-}= 1\}\rangle,\ \texttt{rgtl} : \langle \{x_l < 0\}, \{x_l \mathrel{+}= 1\}\rangle,\\
\texttt{conn} &: \langle \{x_l = x_r\}, \{p := \top\}\rangle,\ \texttt{disc} : \langle \{p = \top\}, \{p := \bot\}\rangle,\\
\texttt{exch} &: \langle \{p = \top, q_l \geq q, q_r \geq -q\}, \{q_l \mathrel{-}= q, q_r \mathrel{+}= q\}\rangle,\\
\texttt{lre} &: \langle \{\}, \{q := 1\}\rangle, \texttt{rle} : \langle \{\}, \{q := -1\}\rangle.
\end{aligned} \tag{1}$$

The action `lftr` models the right robot going left, and similarly for `rgtr`, `lftl` and `rgtl`.

Assume the initial state is $I = \{p := \bot, x_l := -X_I, x_r := X_I, q_l := Q, q_r := 0, q := 1\}$, with $X_I, Q \in \mathbb{N}$. Assuming $G = \{q_l = 0, q_r = Q, x_l = -X_I, x_r = X_I\}$, one of the shortest plans is

$$\texttt{rgtl}^{X_I}; \texttt{lftr}^{X_I}; \texttt{conn}; \texttt{exch}^{Q}; \texttt{disc}; \texttt{lftl}^{X_I}; \texttt{rgtr}^{X_I} \tag{2}$$

where, for each action $a$ and $m \in \mathbb{N}$, $a^m$ denotes the sequence consisting of the action $a$ repeated $m$ times (for $m = 0$, $a^m = \epsilon$). According to the plan (2), the robots go to the origin, connect, exchange the $Q$ items, disconnect, and then go back to their initial positions.

In the rest of the paper, $v, w, x, y$ denote variables, $a, b$ denote actions and $\psi$ denotes a linear expression, each symbol possibly decorated with subscripts. Further, we will handle sequences of actions in different ways, depending on whether we intend each to be

1. a generic sequence of actions, in which case we will use the letter $\alpha$, or
2. a plan, in which case we will use the letter $\pi$, or
3. a pattern, in which case we will use the symbol $\prec$,

each symbol $\alpha, \pi, \prec$ possibly decorated with subscripts and/or superscripts. For any two sequences of actions $\alpha$ and $\alpha'$, $\alpha; \alpha'$ denotes the sequence of actions obtained by concatenating $\alpha'$ to the end of $\alpha$. Finally, we continue to use standard logical terminology using terms like satisfiable, contradictory and valid, taking them for granted.

## 2.2. Satisfiability modulo theories (SMT)

The *satisfiability problem* (SAT) is formally defined as follows. Given a *propositional formula* $f(x_1, \dots, x_n)$, composed of $n$ *propositional variables*, *logical connectives* (e.g., $\wedge$ for *conjunction*, $\vee$ for *disjunction*, $\neg$ for *negation*, and $\rightarrow$ for *implication*), and parentheses for grouping, determine whether there exists a *truth assignment* or *model* to the variables $x_1, \dots, x_n$ that satisfies $f$. A model is a mapping $\mu : \{x_1, x_2, \dots, x_n\} \mapsto \{\top, \bot\}$, where $\top$ and $\bot$ are the symbols for *true* and *false*. The goal is to check whether there exists a model $\mu$ such that $f$ evaluates to true under $\mu$. Formally, this can be written as:

$$\exists \mu : \{x_1, x_2, \dots, x_n\} \mapsto \{\top, \bot\},\ \text{s.t. } f(\mu(x_1), \dots, \mu(x_n)) \equiv \top,$$

where $\equiv$ is the symbol for *logical equivalence* (e.g., $\top \vee \bot \equiv \top$). *Satisfiability Modulo Theories* (SMT) [3] extends the SAT problem by incorporating background theories, enabling reasoning over richer logical structures. In this paper, we will employ SMT with the Quantifier Free Linear Arithmetic theory, which supports reasoning over integers and real numbers under linear constraints. In SMT, one can mix propositional and numeric variables, and thus search for a correct assignment (i.e. a model) to the propositional and numeric variables that solve the SMT formula. This mix of propositional and numeric variables works very naturally in the fragment of numeric planning.

**Example 2.** The preconditions and the effects of the `exch` action of Example 1 can[2] be expressed in SMT using $(i)$ the propositional variable $exch$ to denote whether the action `exch` has been executed or not, $(ii)$ the propositional variable $p$ and the numeric variables

[2] As we will see in the following sections, there are several ways one could express these formulas, here we show the simplest one [14].

$q_l, q_r$ and $q$ to denote the value of the respective variables in $V_B$ and $V_N$ before the application of `exch`, and $(iii)$ the numeric variables $q_l', q_r'$ to represent the value of the respective variables in $V_N$ after the application of `exch`.

$$exch \rightarrow p \wedge (q_l \geq q) \wedge (q_r \geq q),$$
$$exch \rightarrow (q_l' = q_l - q) \wedge (q_r' = q_r + q)$$

## 3. Symbolic pattern planning

Consider a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$, and a *pattern* $\prec = a_1; a_2; \dots; a_k$, defined as an arbitrary finite sequence of actions in $A$ of length $k \geq 0$. From the definition, the pattern can be empty (in which case it reduces to the empty sequence $\epsilon$), or it can contain only some or all of the actions in $A$, possibly multiple times, either consecutively or not. Though the pattern can contain multiple occurrences of a same action $a$, such occurrences will be treated as different copies of $a$. This allows us to treat each action occurrence in the pattern as a variable in our encoding, simplifying the notation and the presentation. When necessary, we will write $a1, a2, \dots$, to mean the first, second, …copy of the action $a$ in the pattern.

In this section, we first formally define the SPP procedure outlined in the introduction (Section 3.1), proving its correctness and completeness assuming the corresponding correctness and completeness of our pattern $\prec$-encoding $\Pi^{\prec}$ of $\Pi$. The formal definition of $\Pi^{\prec}$, together with the proof of its correctness and completeness, is given in Section 3.2. Different procedures to compute patterns and high-quality plans are presented in Sections 3.3 and 3.4, respectively.

### 3.1. *A simple SPP procedure*

The basic idea of Symbolic Pattern Planning (SPP) is to define the value of each state variable in the state resulting from the execution of a subsequence $\alpha$ of the pattern $\prec$ as a function of both the state in which $\alpha$ starts and of the pattern $\prec$. More in details, to each action occurrence $a_i$ in the pattern we associate a distinct numeric action variable whose value denotes the number of times $(\geq 0)$ $a_i$ has been executed after $a_1; \dots; a_{i-1}$. Then, every subsequence $\alpha$ of $\prec$

1. corresponds to one assignment to the action variables in the encoding, and
2. allows expressing the value of each state variable in $s'$ as a function of the starting state $s$ and of the action variables associated to the action occurrences in $\alpha$ (assuming $\alpha$ is executable in a state $s$ and that $s' = res(\alpha, s)$).

Thus, in a SPP encoding, we assume to have the following sets of variables:

1. $\mathcal{X}$, the set of *state variables*, which includes $V_B \cup V_N$, used to impose the initial conditions;
2. $\mathcal{A}^{\prec}$, consisting of one distinct *action variable* for each action occurrence in the pattern $\prec$, used to model which action occurrences in the pattern are executed; and
3. $\mathcal{X}'$, the set of *resulting state variables*, consisting of one variable $x'$ for each state variable $x \in \mathcal{X}$, used to model the values of the state variables in the resulting state and impose the goal conditions.

About the variables in $\mathcal{A}^{\prec}$, we take their domain to be the set of non-negative integers, the value of each variable modelling how many times the action is being consecutively (possibly) executed.

Then, the *(SPP) $\prec$-encoding of* $\Pi$ is the formula

$$\Pi^{\prec} = \mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}') \wedge \mathcal{G}(\mathcal{X}'),$$

in which

1. $\mathcal{I}(\mathcal{X})$ is the *initial state formula*, a formula on the set $\mathcal{X}$ of variables, defined as
$$\bigwedge_{v: I(v)=\top} v \wedge \bigwedge_{w: I(w)=\bot} \neg w \wedge \bigwedge_{x,k: I(x)=k} x = k.$$
2. $\mathcal{G}(\mathcal{X}')$ is the *goal formula*, obtained by making the conjunction of the formulas in $G$, once $(i)$ each variable $v$ is replaced with $v'$, and $(ii)$ $v' = \top$ and $v' = \bot$ are substituted with $v'$ and $\neg v'$, respectively.
3. $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ is a *(pattern) $\prec$-symbolic transition relation*, a formula on the variables $\mathcal{X} \cup \mathcal{A}^{\prec} \cup \mathcal{X}'$ providing a definition of each variable in $\mathcal{G}(\mathcal{X}')$ as a function of the variables in $\mathcal{X} \cup \mathcal{A}^{\prec}$. We will thoroughly discuss the $\prec$-symbolic transition relation in Section 3.2.

Indeed, each $\prec$-encoding of $\Pi$ has to come with a (pattern) $\prec$-decoding function, allowing to associate to each model of $\Pi^{\prec}$ a sequence of actions in $A$, which, for the *correctness* of the $\prec$-encoding, has to be a plan for $\Pi$. For the *completeness* of the $\prec$-encoding, we require that if there exists a subsequence $\alpha$ of $\prec$ which is a plan of $\Pi$, $\Pi^{\prec}$ is satisfiable.

Then, if for any pattern $\prec$ we can define a correct and complete $\prec$-encoding $\Pi^{\prec}$ of $\Pi$, the following simple SPP procedure is guaranteed to return a plan for $\Pi$ if one exists:

1. fix an initial pattern $\prec_I$ including every action in $A$ and start with $\prec = \epsilon$;
2. check whether $\Pi^{\prec}$ is satisfiable,

**Algorithm 1** SPP algorithm. In SPP, the pattern $\prec_I$ is computed once in the initial state and $\prec$ is $\prec_I$ concatenated $n$ times.

```
1:  function SPP(Π)        /* $\Pi = \langle V_B, V_N, A, I, G\rangle$ */
2:      $n \leftarrow 0; \prec \leftarrow \epsilon;$
3:      $\prec_I \leftarrow$ COMPUTEPATTERN(Π);
4:      while (TRUE) do
5:          $\Pi^{\prec} \leftarrow \mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}') \wedge \mathcal{G}(\mathcal{X}');$
6:          $\mu \leftarrow$ SOLVE($\Pi^{\prec}$);
7:          if ($\mu \neq 0$) then
8:              return GETPLAN($\mu, \prec$);
9:          end if
10:         $\prec \leftarrow \prec; \prec_I;$
11:         $n \leftarrow n + 1;$
12:     end while
13: end function
```

3. extend $\prec$ by concatenating $\prec_I$ to it, and iterate the second step upon its failure.

If $\pi$ is a plan of length $n$, $\pi$ will be a subsequence of the pattern $\prec$ generated at the $n$th iteration of the above procedure and the correctness and completeness of the $\prec$-encoding $\Pi^{\prec}$ guarantees the correctness and completeness of the procedure. Notice that the above outlined procedure is guaranteed to terminate at most at the $n$th iteration. Indeed, it will terminate as soon as $\pi$ is a subsequence of the pattern being tested, and even before if the plan $\pi$ contains multiple consecutive occurrences of a same action and our encoding allows modelling such consecutive executions with a single action variable in $\prec_I$.

Algorithm 1 shows the pseudocode of the SPP procedure, in which:

1. COMPUTEPATTERN(Π) returns a *complete pattern*, i.e., a sequence of actions which includes all the actions of the planning task Π.
2. SOLVE($\Pi^{\prec}$) calls a solver which is expected to return a model of $\Pi^{\prec}$ assuming it is satisfiable, and 0 otherwise.
3. GETPLAN($\mu, \prec$) returns the plan corresponding to the model $\mu$ of $\Pi^{\prec}$, i.e., the sequence of actions

$$a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \dots ; a_k^{\mu(a_k)}. \tag{3}$$

For any correct and complete encoding $\Pi^{\prec}$, SPP(Π) is *correct* (any returned sequence of actions is a plan) and *complete* (if a plan exists, SPP(Π) will return one). We thus adopt the standard notion of completeness for search procedures (see, e.g., [15]), which requires the ability to find a solution whenever one exists, and does not require the ability to determine that no solution exists.

**Theorem 1.** *Let Π be a numeric planning problem. If for each pattern $\prec$ $\Pi^{\prec}$ is a correct and complete $\prec$-encoding of Π, then* SPP(Π) *is correct and complete, i.e.,*

1. *any returned plan is valid, and*
2. *a plan is returned when there is a valid one.*

**Proof.** The correctness of SPP(Π) follows directly from the hypothesis of the correctness of the $\prec$-encoding. For the completeness of SPP(Π), let $\pi$ be a plan of length $n$. Then, after the $n$th iteration of the loop in SPP(Π), $\pi$ is a subsequence of $\prec$ and thus the completeness of SPP(Π) follows from the completeness of the $\prec$-encoding of Π. □

The completeness of SPP(Π) essentially relies on the fact that after $n$ iterations, the pattern $\prec$ is *$n$-complete*, i.e., that it contains at least $n$ non-overlapping subsequences in which every action in $A$ occurs. It is then clear that the procedure maintains its correctness and completeness if the SPP(Π) procedure is modified in order to, at each iteration,

1. compute a possibly different complete pattern to be concatenated with the previously used pattern: this modification amounts to remove line 3 and insert the new line of code

   $\prec_I \leftarrow$ COMPUTEPATTERNI($\Pi, \prec$);

   in between lines 9 and 10, in which COMPUTEPATTERNI($\Pi, \prec$) is assumed to return a complete pattern, or
2. compute an $n$-complete pattern to be used in the next iteration, possibly entirely different from the previously used pattern: this modification amounts to removing line 3 and replacing line 10 with the line of code

   $\prec \leftarrow$ COMPUTEPATTERNN($\Pi, \prec$);

   in which COMPUTEPATTERNN($\Pi, \prec$) is assumed to return a $n$-complete pattern.

It is clear that SPP(Π), as in Algorithm 1, can be considered a special case of the SPP(Π) procedure as modified in the first of the above two items, which in turn can be considered a special case of the SPP(Π) procedure as modified in the second of the above two items.

## 3.2. A correct and complete SPP encoding for numeric planning problems

We now formally define a correct and complete $\prec$-encoding $\Pi^{\prec}$ of $\Pi$, which amounts to define the $\prec$-symbolic transition relation $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$. A pattern is *elementary* if the same action doesn't appear multiple times in the pattern. It is *complete* if all the actions in $A$ appear in the pattern.

In the *$\prec$-encoding* $\Pi^{\prec}$ of $\Pi$,

1. $\mathcal{X} = V_B \cup V_N$, and
2. $\mathcal{A}^{\prec}$ contains a distinct action variable with domain in $\mathbb{N}$ for each action occurrence in $\prec$ (thus $|\mathcal{A}^{\prec}| = k$).

As already said, in the following, for each $i \in [1, k]$, we will use $a_i$ to denote both the $i$th action in $\prec$ and the corresponding action variable in $\mathcal{A}^{\prec}$.

Assume the pattern $\prec$ is not empty and consider an arbitrary action $a$ in it (its position is irrelevant at this time).

Intuitively, as proposed by [6], the value assumed by the action variable $a \in \mathcal{A}^{\prec}$ represents the number $\geq 0$ of times the action has to be consecutively executed. Of course, the possibility to have $a > 1$ is an optimization allowing the pattern $\prec$ to model transitions in which actions are also consecutively executed more than once: restricting $a$ in $\{0, 1\}$ neither affects the correctness nor the completeness of the SPP($\Pi$) procedure, but it may affect performance. Though it might be desirable to allow $a$ assuming any possible value, it is not always possible to allow $a > 1$, e.g., because the action $a$ cannot be executed more than once, or it is not easy to compute the effects of executing $a$ more than once, or it is not useful to execute $a$ more than once. To define when it is possible to allow $a > 1$, each effect $v := e$ of the action $a$ is categorized as

1. a *linear increment*, if $e = v + \psi$ with $\psi$ a linear expression not containing any of the variables assigned by $a$, as for the effects of the action `exch` and `lftr` in (1), or as
2. a *general assignment*, if it is not a linear increment. General assignments are further divided into
   (a) *simple assignments*, when $e$ does not contain any of the variables assigned by $a$, as in the effects of the actions `conn`, `disc`, `lre` and `rle` in (1), and
   (b) *self-interfering assignments* (e.g., $\mathrm{eff}(a) = \{x := y, y := x\}$), otherwise.

Then, the action $a$ is *eligible for rolling* if [3]

1. $v = \bot \in \mathrm{pre}(a)$ (resp. $v = \top \in \mathrm{pre}(a)$) implies $v := \top \notin \mathrm{eff}(a)$ (resp. $v := \bot \notin \mathrm{eff}(a)$), and
2. $a$ does not contain a self-interfering assignment, and
3. $a$ contains a linear increment.

Whenever an action $a$ is eligible for rolling, it is possible to determine both the conditions under which it is possible to execute $a$ for $m$ times in a state $s$, and the conditions on the resulting state.

**Theorem 2.** *[Scala et al., (2016)] Let $\Pi$ be a numeric planning problem. Let $a$ be an action which is eligible for rolling. For any two states $s$ and $s'$ and integer $m > 0$,*

$$s' = res(a^m, s)$$

*if and only if*

1. *for each numeric precondition $\psi \trianglerighteq 0$ in $\mathrm{pre}(a)$,*
$$s(\psi) \trianglerighteq 0 \quad \textit{and} \quad s(\psi[m]) \trianglerighteq 0, \tag{4}$$
*where $\psi[m]$ is the linear expression obtained from $\psi$ by substituting each variable $x$ with*
   *(a) $x + (m-1) \times \psi'$, whenever $x \mathrel{+}= \psi' \in eff(a)$ is a linear increment,*
   *(b) $\psi'$, whenever $x := \psi' \in \mathrm{eff}(a)$ is a simple assignment.*
2. *for each variable $v$,*
   *(a) $s'(v) = \top$ (resp. $s'(v) = \bot$) whenever $v := \top \in \mathrm{eff}(a)$ (resp. $v := \bot \in \mathrm{eff}(a)$);*
   *(b) $s'(v) = s(v) + m \times s(\psi)$ whenever $v \mathrel{+}= \psi \in \mathrm{eff}(a)$;*
   *(c) $s'(v) = s(\psi)$ whenever $v := \psi \in \mathrm{eff}(a)$ is a simple assignment;*
   *(d) $s'(v) = s(v)$, otherwise.*

The conditions in (4) ensure that $\psi \trianglerighteq 0$ holds in the states in which the first and the last execution of $a$ happens. The satisfaction of these two conditions ensure that each precondition $\psi \trianglerighteq 0$ of $a$ is satisfied also in the intermediate states $s$ in between the first and the last execution of $a$. This is a consequence of the fact, proved in [6], that the function $\psi[a]$ is monotonic in $a$ if the action is eligible for rolling.

Let $\prec = a_1; \dots; a_k$. Now, for each $i \in [0, k]$, we define the expression $\sigma_i(v)$, representing the value of each variable $v \in V_B \cup V_N$ after the execution of $\prec_i$, as a function of the action variables in $\mathcal{X} \cup \{a_1, a_2, \dots, a_i\}$. Clearly, if $i = 0$, $\sigma_0(v) = v$ while, if $i \in [1, k]$, $\sigma_i(v)$ is recursively defined as follows[4]:

[3] Here, as in [2], we consider just the cases $\alpha = 0$ and $\alpha = 1$ of Theorem 1 in [6], which (quoting) "cover a very general class of dynamics, where rates of change are described by linear or constant equations".

[4] Note that each $\sigma_i$ is not implemented as a auxiliary variable, but is an expression over $\mathcal{X} \cup \{a_1, a_2, \dots, a_i\}$.

1. if $v$ is not assigned by $a_i$, the value of $v$ does not change, no matter whether $a_i$ is executed or not, and thus
$$\sigma_i(v) = \sigma_{i-1}(v);$$
2. if $v := \top \in \mathtt{eff}(a_i)$, $v$ will get the value $\top$ if $a_i$ is executed and will keep the same value otherwise, and thus
$$\sigma_i(v) = (\sigma_{i-1}(v) \vee a_i > 0);$$
3. if $v := \bot \in \mathtt{eff}(a_i)$, $v$ will get the value $\bot$ if $a_i$ is executed and will keep the same value otherwise, and thus
$$\sigma_i(v) = (\sigma_{i-1}(v) \wedge a_i = 0);$$
4. if $v \mathrel{+}= \psi \in \mathtt{eff}(a_i)$ is a linear increment, the value of $v$ will be incremented by the value of $\psi$ multiplied by the number of times $a_i$ is consecutively executed, and thus
$$\sigma_i(v) = \sigma_{i-1}(v) + a_i \times \sigma_{i-1}(\psi),$$
where $\sigma_{i-1}(\psi)$ is the expression obtained by substituting each variable $v \in V_N$ in $\psi$ with $\sigma_{i-1}(v)$;
5. if $v := \psi \in \mathtt{eff}(a_i)$ is a general assignment, suitable "at-most-once" axioms will restrict $a_i$ to range in $\{0, 1\}$ if action $a_i$ is not eligible for rolling, and then executing $a_i$ will cause $v$ getting the value $\sigma_{i-1}(\psi)$, while $v$ will keep the same value if $a_i$ is not executed, and thus [5]
$$\sigma_i(v) = \text{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v)),$$
where $\text{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v))$ returns $\sigma_{i-1}(\psi)$ or $\sigma_{i-1}(v)$ depending on whether $a_i > 0$ is true or not, and belongs to the standard functions defined in SMTLIB [16].

**Example 3.** Consider (1), and assume $\prec$ is

$$\mathtt{lre}; \mathtt{rle}; \mathtt{lftr}; \mathtt{rgtl}; \mathtt{conn}; \mathtt{exch}; \mathtt{disc}; \mathtt{rgtr}; \mathtt{lftl}. \tag{5}$$

The pattern contains all the 9 actions in $A$ exactly once, and the value $\sigma(v)$ of each variable $v$ after executing $\prec$, each action of $\geq 0$ times, is

1. for the Boolean variable $p$,
$$\sigma(p) = (p \vee \mathtt{conn} > 0) \wedge \mathtt{disc} = 0,$$
2. and, for the numeric variables in $V_N = \{x_l, x_r, q_l, q_r, q\}$,
$$\begin{aligned}\sigma(x_l) &= x_l + \mathtt{rgtl} - \mathtt{lftl},\\ \sigma(x_r) &= x_r - \mathtt{lftr} + \mathtt{rgtr},\\ \sigma(q_l) &= q_l - \mathtt{exch} \times q^{\mathtt{rle}},\\ \sigma(q_r) &= q_r + \mathtt{exch} \times q^{\mathtt{rle}},\\ \sigma(q) &= q^{\mathtt{rle}}.\end{aligned}$$
in which $q^{\mathtt{rle}}$ abbreviates the term $\text{ITE}(rle > 0, -1, \text{ITE}(lre > 0, 1, q))$.

Notice that the above definition of $\sigma(v)$ for $v \in V_B \cup V_N$ depends not only on which are the actions in the pattern, but also on their position in the pattern. For instance, if $\prec$ is

$$\mathtt{lftr}; \mathtt{rgtl}; \mathtt{conn}; \mathtt{exch}; \mathtt{disc}; \mathtt{rgtr}; \mathtt{lftl}; \mathtt{lre}; \mathtt{rle},$$

i.e., if we assume we set the value of the state variable $q$ at the end of the pattern, then the value of $\sigma(v)$ remains the same as the one above defined for all the variables except for $q_l$ and $q_r$, about which we now get:

$$\begin{aligned}\sigma(q_l) &= q_l - \mathtt{exch} \times q,\\ \sigma(q_r) &= q_r + \mathtt{exch} \times q,\end{aligned}$$

modelling that now the two robots exchange items at the initially fixed rate $q$.

If we omit the actions `lre` and `rle` from the pattern, and thus if we assume $\prec$ is

$$\mathtt{lftr}; \mathtt{rgtl}; \mathtt{conn}; \mathtt{exch}; \mathtt{disc}; \mathtt{rgtr}; \mathtt{lftl}$$

we will get the same $\sigma(v)$ as the one we just defined for all the variables except for $q$, about which we now get

$$\sigma(q) = q,$$

reflecting the fact that there is no action in the pattern modifying the initial value of the state variable $q$.

[5] The definition of $\sigma_i(v)$ that we give here for this case is different from the one we used in [2], which does not rely on ITE terms and requires the introduction of one additional variable.

If $\prec$ is

$$\texttt{lre1}; \texttt{rle1}; \texttt{lftr}; \texttt{rgtl}; \texttt{conn}; \texttt{exch}; \texttt{disc}; \texttt{rgtr}; \texttt{lftl}; \texttt{lre2}; \texttt{rle2},$$

i.e., if we assume we set the value of the state variable $q$ both at the beginning and also at the end of the pattern, then the value of $\sigma(v)$ remains the same for the Boolean variable $p$ and the numeric variables $x_l$ and $x_r$, while the others become:

$$\begin{aligned}&\sigma(q_l) = q_l - \texttt{exch} \times q^{\texttt{rle1}},\\ &\sigma(q_r) = q_r + \texttt{exch} \times q^{\texttt{rle1}},\\ &\sigma(q) = q^{\texttt{rle2}}.\end{aligned}$$

in which $q^{\texttt{rle1}}$ abbreviates the term $\text{ITE}(rle1 > 0, -1, \text{ITE}(lre1 > 0, 1, q))$, and $q^{\texttt{rle2}}$ abbreviates the term $\text{ITE}(rle2 > 0, -1, \text{ITE}(lre2 > 0, 1, q^{\texttt{rle1}}))$.

The $\prec$-symbolic transition relation $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ of $\Pi^{\prec}$ is simply the conjunction of the formulas enforcing

1. at-most-once axioms for the actions not eligible for rolling; and
2. preconditions axioms enforcing that executing an action is possible only in states in which its preconditions are satisfied; and
3. an explicit definition of each variable $v' \in \mathcal{X}'$ as a function of the variables in $\mathcal{X} \cup \mathcal{A}^{\prec}$, i.e., of the starting state and the variables corresponding to the action occurrences in the pattern.

Formally, $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ is the conjunction of

1. $\text{amo}^{\prec}(A)$ which contains, for each $i \in [1, k]$,

   $$a_i = 0 \lor a_i = 1,$$

   whenever the action $a_i$ is not eligible for rolling. [6]
2. $\text{pre}^{\prec}(A)$, which contains, for each $i \in [1, k]$, and for each $v = \bot$ and for each $w = \top$ in $\text{pre}(a_i)$,

   $$a_i > 0 \to \neg\sigma_{i-1}(v), \quad a_i > 0 \to \sigma_{i-1}(w),$$

   and, for each numeric precondition $\psi \trianglerighteq 0$ in $\text{pre}(a_i)$ we ensure the conditions in (4),

   $$a_i > 0 \to \sigma_{i-1}(\psi) \trianglerighteq 0, \quad a_i > 1 \to \sigma_{i-i}(\psi[a_i]) \trianglerighteq 0.$$
3. $\text{frame}^{\prec}(V_B \cup V_N)$, consisting of, for each variable $v \in V_B$ and $x \in V_N$,

   $$v' \leftrightarrow \sigma_k(v), \quad x' = \sigma_k(x).$$

**Example 4.** Assume the pattern $\prec$ is (5), i.e.,

$$\texttt{lre}; \texttt{rle}; \texttt{lftr}; \texttt{rgtl}; \texttt{conn}; \texttt{exch}; \texttt{disc}; \texttt{rgtr}; \texttt{lftl}.$$

In this case,

1. $\text{amo}^{\prec}(A)$ is

   $$\begin{aligned}&\texttt{lre} = 0 \lor \texttt{lre} = 1, \quad \texttt{rle} = 0 \lor \texttt{rle} = 1,\\ &\texttt{conn} = 0 \lor \texttt{conn} = 1, \quad \texttt{disc} = 0 \lor \texttt{disc} = 1.\end{aligned}$$
2. $\text{pre}^{\prec}(A)$ is equivalent to

   $$\begin{aligned}&\texttt{lftr} > 0 \to x_r > 0,\ \texttt{lftr} > 1 \to x_r - (\texttt{lftr} - 1) > 0,\\ &\texttt{rgtr} > 0 \to \neg((p \lor \texttt{conn} > 0) \land \texttt{disc} = 0),\\ &\texttt{lftl} > 0 \to \neg((p \lor \texttt{conn} > 0) \land \texttt{disc} = 0),\\ &\texttt{rgtl} > 0 \to x_l < 0,\ \texttt{rgtl} > 1 \to x_l + (\texttt{rgtl} - 1) < 0,\\ &\texttt{conn} > 0 \to x_l + \texttt{rgtl} = x_r - \texttt{lftr},\\ &\texttt{disc} > 0 \to (p \lor \texttt{conn} > 0),\\ &\texttt{exch} > 0 \to ((p \lor \texttt{conn} > 0) \land q_l \geq q^{\texttt{rle}} \land q_r \geq -q^{\texttt{rle}}),\\ &\texttt{exch} > 1 \to (q_l \geq q^{\texttt{rle}} - (\texttt{exch} - 1) \times q^{\texttt{rle}}),\\ &\texttt{exch} > 1 \to (q_r \geq -q^{\texttt{rle}} + (\texttt{exch} - 1) \times q^{\texttt{rle}}).\end{aligned}$$

   in which $q^{\texttt{rle}}$ abbreviates the term $\text{ITE}(rle > 0, -1, \text{ITE}(lre > 0, 1, q))$ as before.

[6] If the action is not eligible for rolling then $a$ can be defined as a Boolean variable. However, this will require to change the recursive definition of $\sigma_i(v)$. In particular, if $v \mathrel{+}= \psi \in \text{eff}(a_i)$ the new definition will be:

$$\sigma_i(v) = \text{ITE}(a_i, \sigma_{i-1}(v) + \sigma_{i-1}(\psi), \sigma_{i-1}(v)),$$

and similarly for the other cases.

3. $\mathrm{frame}^{\prec}(V_B \cup V_N)$ is

$$\begin{array}{l} p' \leftrightarrow ((p \vee \mathtt{conn} > 0) \wedge \mathtt{disc} = 0), \\ x'_l = x_l + \mathtt{rgtl} - \mathtt{lftl}, \quad x'_r = x_r - \mathtt{lftr} + \mathtt{rgtr}, \\ q'_l = q_l - \mathtt{exch} \times q^{\mathtt{rle}}, \quad q'_r = q_r + \mathtt{exch} \times q^{rle}, \\ q' = q^{\mathtt{rle}}. \end{array}$$

$\Pi^{\prec}$ is the conjunction of the above formulas together with the formulas encoding the initial and goal states. $\Pi^{\prec}$ is satisfiable, and the plan (2) corresponds to a model of $\Pi^{\prec}$.

Indeed, if $\Pi$ is the domain in the example and COMPUTEPATTERN($\Pi$) in the SPP($\Pi$) procedure in Fig. 1 returns the complete pattern (5), SPP($\Pi$) will return a plan when $n = 1$, i.e., at the first iteration in which $\prec$ is not empty.

**Theorem 3.** *Let $\Pi$ be a numeric planning problem. Let $\prec$ be a pattern. The SPP $\prec$-encoding $\Pi^{\prec}$ is correct and complete.*

**Proof.** Let $\prec = a_1; a_2; \dots; a_k,\ k \geq 0$. For $\Pi = \langle V_B, V_N, A, I, G\rangle$, let $\Pi_\emptyset$ be the numeric planning problem $\Pi$ without goals, i.e., $\Pi_\emptyset = \langle V_B, V_N, A, I, \emptyset\rangle$. Clearly, any executable sequence of actions (even the empty one) is a plan for $\Pi_\emptyset$.

Correctness. We first prove the correctness of the encoding considering the planning problem $\Pi_\emptyset$. Specifically, we first prove that if $\mu$ is a model of $\Pi^{\prec}_\emptyset$ then $\pi = a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \dots; a_k^{\mu(a_k)}$ is a plan of $\Pi_\emptyset$ and $s_k = res(\pi, I)$ is such that, for each state variable $v \in V_B \cup V_N$, $s_k(v) = \mu(\sigma_k(v)) = \mu(v')$, i.e., the value of $v$ in $s_k$ coincides with $(i)$ the value of $v$ computed via the expression $\sigma_k$ at the end of the pattern and $(ii)$ with the value of $v' \in \mathcal{X}'$. The proof is by induction on the length $k$ of $\prec$. If $k = 0$, then $\prec = \pi = \epsilon$ and the thesis follows since the empty sequence of actions is a valid plan, $s_k = I$ and $\Pi^{\prec}_\emptyset$ reduces to

$$\Pi^{\prec}_\emptyset = \mathcal{I}(\mathcal{X}) \wedge \bigwedge_{v \in V_B} v \equiv v' \wedge \bigwedge_{v \in V_N} v = v'.$$

If $k = i + 1 > 0$, let $\pi_i = a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \dots; a_i^{\mu(a_i)}$ and $\pi = \pi_i; a_k^{\mu(a_k)}$. By induction hypothesis, $s_i = res(\pi_i, I)$ is defined and for each state variable $v \in V_B \cup V_N$, $s_i(v) = \mu(\sigma_i(v))$. Then $s_k$, equal to $res(a_k^{\mu(a_k)}, s_i)$, is defined, and for each state variable $v \in V_B \cup V_N$, $s_k(v) = \mu(\sigma_k(v)) = \mu(v')$, holds for every possible value of $\mu(a_k)$. When $\mu(a_k) = 0$, $a_k^{\mu(a_k)} = \epsilon$, $s_k = s_i$ and for each state variable $v \in V_B \cup V_N$, $\mu(v') = \mu(\sigma_k(v)) = \mu(\sigma_i(v))$. When $\mu(a_k) > 0$, the thesis follows from Theorem 2. Now consider a model $\mu$ of $\Pi^{\prec}$. Then, $\mu$ is also a model of $\Pi^{\prec}_\emptyset$ and $\pi$ is a plan of $\Pi_\emptyset$. The fact that the state $s_k = res(\pi, I)$ satisfies $G$ follows from the fact that for each state variable $v \in V_B \cup V_N$, $s_k(v) = \mu(\sigma_k(v)) = \mu(v')$ and $\mu$ satisfies $\mathcal{G}(\mathcal{X}')$.

Completeness. Given the definition of completeness for the $\prec$-encoding, we have to prove that if $\Pi$ admits a plan which is a subsequence of $\prec$, then $\Pi^{\prec}$ is satisfiable. Let $\pi$ be a valid plan of length $n \leq k$ of $\Pi$ which is also a subsequence of $\prec$. Let $s_n$ be the last state induced by $\pi$. Then, if we consider $\pi$ as a pattern and build $\Pi^{\pi}$, the assignment $\mu$ extending $I$, assigning all the actions in $\mathcal{A}^{\pi}$ to 1 and such that, for each variable $v' \in \mathcal{X}'$, $\mu(v') = s_n(v)$ is a model of $\Pi^{\pi}$. The proof that $\mu$ is a valid model of $\Pi^{\pi}$ is by induction on $n$ and analogous to the proof done for correctness, by first considering $\Pi_\emptyset$. Then, going back to the proof of completeness of $\Pi^{\prec}$, we conclude showing that $\Pi^{\pi}$ is equivalent to the formula obtained from $\Pi^{\prec}$ when substituting each action variable not in $\pi$ with 0, and hence $\Pi^{\prec}$ is satisfiable. □

Due to Theorems 1 and 3, for any numeric planning problem $\Pi$, the SPP($\Pi$) procedure in Fig. 1 is correct and complete.

## 3.3. Pattern computation

Consider a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$. Though the SPP($\Pi$) procedure in Algorithm 1 is guaranteed to be correct and complete for any complete pattern $\prec_I$ returned by COMPUTEPATTERN($\Pi$), it is clear that its performance may critically depend on $\prec_I$, as shown also by our running example.

**Example 5.** As already seen, if the pattern $\prec$ is (5), then $\Pi^{\prec}$ is satisfiable. Thus, if $\prec_I$ is (5) then SPP($\Pi$) returns a plan after $n = 1$ concatenations of $\prec_I$. On the other hand, if $\prec_I$ is the sequence obtained reversing (5), i.e.,

$$\mathtt{lftl}; \mathtt{rgtr}; \mathtt{disc}; \mathtt{exch}; \mathtt{conn}; \mathtt{rgtl}; \mathtt{lftr}; \mathtt{rle}; \mathtt{lre} \tag{6}$$

then SPP($\Pi$) returns a plan after $n = 5$ concatenations of $\prec_I$.

Even considering the SPP($\Pi$) procedure in Algorithm 1 modified to compute a (possibly) brand new $n$-complete pattern at each iteration (as discussed at the end of Section 3.1), the problem is how to easily (i.e., in polynomial time) compute a “good” pattern. To address this problem, consider a pattern $\prec = a_1; \dots; a_k,\ k \geq 0$. Our desideratum is to compute a pattern $\prec'$ such that

1. for each action $a \in A$, the number of occurrences of $a$ in $\prec'$ is at most equal to the number of occurrences of $a$ in $\prec$, and
2. $\prec'$ *dominates* $\prec$, i.e., such that $\Pi^{\prec}$ satisfiability implies $\Pi^{\prec'}$ satisfiability.

The first requirement is necessary, as it is easy to satisfy the second one by simply adding action occurrences to $\prec$. Indeed, by adding an action to $\prec$, we obtain a new pattern that strongly dominates the previous one. A pattern $\prec'$ *strongly dominates* $\prec$ if and only if for any planning problem $\Pi'$ possibly differing from $\Pi$ only in the initial state $I$ and goal $G$, $\Pi'^{\prec}$ satisfiability implies $\Pi'^{\prec'}$ satisfiability. Of course, if $\prec'$ strongly dominates $\prec$, then $\prec'$ dominates $\prec$.

**Theorem 4.** *Let $\Pi$ be a numeric planning problem. Let $\prec$ be a pattern. Let $a$ be an action in $\Pi$. Let $\prec + a$ be a pattern obtained inserting $a$ in $\prec$. $\prec + a$ strongly dominates $\prec$.*

**Proof.** Let $\prec = a_1; \dots; a_k$ and $\prec + a = a_1; \dots; a_i; a; a_{i+1}; \dots; a_k$, $0 \le i \le k$. $\prec + a$ strongly dominates $\prec$, since each model $\mu$ of $\mathcal{T}^{\prec}$ can be extended to a model $\mu'$ of $\mathcal{T}^{\prec+a}$ with $\mu'(a) = 0$. □

According to the theorem, removing actions from the pattern $\prec$ produces a new pattern $\prec'$ which, at least theoretically, will not allow us to solve more problems: the best we can get is that $\prec$ and $\prec'$ are equivalent or strongly equivalent. A pattern $\prec'$ is *equivalent* (resp. *strongly equivalent*) to $\prec$ if and only if $\prec'$ dominates (resp. strongly dominates) $\prec$ and vice versa. However, on the practical side, a pattern with fewer actions produces formulas with fewer variables which are likely to be easier to solve.

For the above reasons, we first concentrate on determining sufficient conditions allowing to improve a pattern by removing action occurrences from it. Then, we present conditions allowing to prove when swapping two actions leads to a new pattern which (strongly) dominates the original one. Finally, we show how we can effectively build a pattern based on the previously presented findings.

In the following, for each $i \in [0, k]$ and state $s$, we inductively define the set $R_s^{\prec_i}$ of *states reachable with $\prec_i$ starting from the state $s$*, as

1. $R_s^{\epsilon} = \{s\}$ for $i = 0$, and
2. for $i > 0$, as the smallest set containing the states $res(a_i^m, s)$ whenever $s \in R_s^{\prec_{i-1}}$, $a_i^m$ is executable in $s$, $m \ge 0$ and also $m \le 1$ if $a$ is not eligible for rolling.

Intuitively, $R_s^{\prec_i}$ represents the set of states which are the result of executing each action in $\prec_i$, for 0, 1 or more times (if eligible for rolling), starting from $s$ used as initial state. From the definition, for $i > 0$, it follows that

1. for any state $s$, $R_s^{\prec_{i-1}} \subseteq R_s^{\prec_i}$,
2. for any pattern $\prec'$, if $R_I^{\prec} \subseteq R_I^{\prec'}$ then $\prec'$ dominates $\prec$, and
3. for any pattern $\prec'$ we have that $R_s^{\prec} \subseteq R_s^{\prec'}$ for any state $s$ if and only if $\prec'$ strongly dominates $\prec$.

### 3.3.1. *Improving patterns by removing action occurrences*

Consider an action occurrence $a_i$ in the pattern $\prec$ and the problem of determining when $a_i$ can be removed from $\prec$, still obtaining an equivalent pattern. In general, this is possible if $R_I^{\prec_{i-1}} = R_I^{\prec_i}$, i.e., when the execution of $a_i^m$ in any state in $R_I^{\prec_{i-1}}$ does not lead to any new state, for any $m \ge 0$. Indeed, checking whether this condition holds is far from being trivial, since in general it amounts to check the unsatisfiability of the formula

$$\mathcal{I}(\mathcal{X}) \wedge \exists a_1 \dots \exists a_i . \mathcal{T}^{\prec_i}(\mathcal{X}, \mathcal{A}^{\prec_i}, \mathcal{X}') \wedge \neg \exists a_1 \dots \exists a_{i-1} . \mathcal{T}^{\prec_{i-1}}(\mathcal{X}, \mathcal{A}^{\prec_{i-1}}, \mathcal{X}').$$

Apart from the cases in which we can easily check that executing $a_i$ does not affect the state in which it is executed (as, e.g., in the case of the example where `lre` is the first action in the pattern (5) and $q$ is already equal to 1 in $I$), we can simplify the pattern by removing $a_i$

1. when $a_{i+1}$ is another occurrence of the action $a_i$ and $a_i$ is eligible for rolling, or
2. when $a_i$ is not executable in any state in $R_I^{\prec_{i-1}}$ (assuming it can be easily computed).

**Theorem 5.** *Let $\Pi$ be a numeric planning problem. Let $\prec = a_1; \dots; a_k$ be a pattern, $k \ge 0$. Let $i \in [1, k]$ and assume that*

1. *$i < k$, $a_i = a_{i+1}$ and $a_i$ is eligible for rolling, or*
2. *$a_i$ is not executable in any state of $R_I^{\prec_{i-1}}$.*

*Then, we can remove $a_i$ from $\prec$ and obtain an equivalent pattern.*

**Proof.** We prove the two statements separately.

1. If $a_i$ is eligible for rolling, given a model $\mu$ of $\Pi^{\prec}$, the assignment $\mu'$ differing from $\mu$ only for the interpretation of $a_i$ and $a_{i+1}$ and such that $\mu'(a_i) = \mu(a_i) + \mu(a_{i+1})$ and $\mu'(a_{i+1}) = 0$ is a model of $\Pi^{\prec}$ and hence of $\Pi^{\prec'}$.
2. If $a_i$ is not executable in any state of $R_I^{\prec_{i-1}}$ then $R_I^{\prec_{i-1}} = R_I^{\prec_i}$. Thus, the conclusion follows.

□

**Corollary 1.** *In the hypothesis of the previous theorem, the pattern*

$$a_1; \dots; a_{i-1}; a_{i+1}; \dots; a_j; a_i; a_{j+1}; \dots; a_k$$

*with $i < j \le k$ obtained from $\prec$ by moving $a_i$ after $a_j$, dominates $\prec$.*

**Proof.** From Theorem 5, we can remove $a_i$ from $\prec$ and obtain an equivalent pattern $\prec'$. From Theorem 4, adding an action to $\prec'$ leads to a new pattern dominating $\prec'$ and hence also $\prec$. □

It is relatively easy to check when the first condition of the theorem is met. For instance, if we consider the pattern obtained concatenating the actions in (5) and in (6), we obtain two consecutive occurrences of the action lftl in the resulting pattern: since lftl is eligible for rolling, one of such two occurrences can be safely removed.

About the second condition of the theorem, it is possible in polynomial time to compute a superset of $R_I^{\prec_{i-1}}$ and check $a_i$ executability in any of its states, by using the Asymptotic Relaxed Plan Graph (ARPG) construction [4]. We will denote the superset of $R_I^{\prec_{i-1}}$ computed with ARPG with $R_{\text{ARPG}}^{\prec_{i-1}}$. For a detailed presentation of the ARPG and its properties, see [4]. Here we convey its basic ideas and properties by considering our running example, which will be used also to show an application of the first condition of the theorem.

**Example 6.** In the ARPG construction, each Boolean variable is associated to the set of values it can assume, and each numeric variable is associated to a convex interval representing an overapproximation of the set of values it can assume. A *relaxed state* is then an assignment in which each variable gets a value in the associated set of values. Starting from the representation $R_{\text{ARPG}}^{\epsilon}$ of the initial state,

$$R_{\text{ARPG}}^{\epsilon} = \{\langle p, \{\bot\}\rangle, \langle x_l, [-X_I, -X_I]\rangle, \langle x_r, [X_I, X_I]\rangle, \langle q_l, [Q, Q]\rangle, \langle q_r, [0,0]\rangle, \langle q, [1,1]\rangle\},$$

given the $i$th action $a_i$ whose preconditions are satisfied by some relaxed state in $R_{\text{ARPG}}^{\prec_i}$, $R_{\text{ARPG}}^{\prec_i;a_i}$ is obtained by modifying the interval associated to each variable assigned by $a_i$ to include the possible new values the variable can assume after the consecutive execution of $a_i$ for finitely many times. For instance, considering the set $R_I^{\text{lre;rle;lftr;rgtl}}$ representing the set of states reachable with the initial pattern lre; rle; lftr; rgtl of the pattern (5), the corresponding superset $R_{\text{ARPG}}^{\text{lre;rle;lftr;rgtl}}$ computed with ARPG is

$$\{\langle p, \{\bot\}\rangle, \langle x_l, [-X_I, +\infty)\rangle, \langle x_r, (-\infty, X_I]\rangle, \langle q_l, [Q, Q]\rangle, \langle q_r, [0,0]\rangle, \langle q, [-1,1]\rangle\}.$$

See [4] for more details. Thus, assuming $\prec$ is (5), the $i$th action in the pattern is executable in at least one state in the overapproximation of the set of states $R_I^{\prec_{i-1}}$ computed with ARPG. Further, there are no two consecutive occurrences of the same action, and thus it is not possible to simplify the pattern $\prec$ by removing some action using the two proposed methods.

On the other hand, if $\prec$ is (6), then the set $R_{\text{ARPG}}^{\text{lftl;rgtr}}$, representing the superset of the states reachable from $R_{\text{ARPG}}^{\epsilon}$ by executing the first 2 actions in the pattern (6), is

$$\{\langle p, \{\bot\}\rangle, \langle x_l, (-\infty, -X_I]\rangle, \langle x_r, [X_I, +\infty)\rangle, \langle q_l, [Q, Q]\rangle, \langle q_r, [0,0]\rangle, \langle q, [1,1]\rangle\}.$$

Then, the action disc is not executable in any state represented by $S_{\text{ARPG}}^{\text{lftl;rgtr}}$ (and thus $R_{\text{ARPG}}^{\text{lftl;rgtr;disc}} = R_{\text{ARPG}}^{\text{lftl;rgtr}}$), and similarly for the actions exch, and conn. Thus, we can remove such actions from (6) and obtain an equivalent pattern.

Notice that starting from the representation of the initial state at level $l = 0$, we can

1. extend the ARPG construction by inserting the action level $l$ consisting of the actions whose preconditions are relaxed-satisfied at that level,
2. compute the relaxed representation of the state at level $l + 1$ which are reachable given the execution of the actions at level $l$, and
3. iterate the process until no more new actions can be introduced.

The result is that each action in the ARPG has an associated level, in our case:

$$\begin{array}{ll} level\ 0: & \texttt{lftl}, \texttt{rgtl}, \texttt{lftr}, \texttt{rgtr}, \texttt{rle}, \texttt{lre}, \\ level\ 1: & \texttt{conn}, \\ level\ 2: & \texttt{disc}, \texttt{exch}, \end{array} \tag{7}$$

corresponding to a partial order on actions. By construction, if an action $a_i$ at level $l$ precedes all the actions with level $< l$ in the pattern $\prec$, then $a$ is not executable in any state in $R_I^{\prec_{i-1}}$ and moving $a_i$ after all the actions at level $< l$ leads to a dominating pattern.

As the example makes clear, building the pattern by extending the partial order given by the ARPG construction ensures that no action can be removed based on the results in this section. In [2], patterns were computed with such methodology.

### 3.3.2. Improving patterns by swapping action occurrences

Consider a pattern $\prec' = a_1; \ldots; a_{i-1}; a_{i+1}; a_i; a_{i+2}; \ldots; a_k$, $0 \le i \le k$, differing from $\prec$ only because now $a_{i+1}$ precedes $a_i$ in $\prec'$. As usual, $\prec_{i-1} = \prec'_{i-1} = a_1; \ldots; a_{i-1}$ while $\prec_i = \prec_{i-1}; a_i \neq \prec'_i = \prec'_{i-1}; a_{i+1}$ and $\prec_{i+1} = \prec_i; a_{i+1} \neq \prec'_{i+1} = \prec'_i; a_i$.

We now define sufficient conditions under which $\prec'$ *(strongly) dominates* or is *(strongly) equivalent* to $\prec$. Clearly, for any state $s \in R_I^{\prec_{i-1}}$ if $a_i^m$, $m \ge 1$, (resp. $a_{i+1}^n$, $n \ge 1$) is executable in $s$ then $res(a_i^m, s)$ (resp. $res(a_{i+1}^n, s)$) belongs to both $R_I^{\prec_{i+1}}$ and $R_I^{\prec'_{i+1}}$, and so the possible differences between $R_I^{\prec_{i+1}}$ (resp. $R_I^{\prec}$) and $R_I^{\prec'_{i+1}}$ (resp. $R_I^{\prec'}$) come from executing either $a_i^m; a_{i+1}^n$ or $a_{i+1}^n; a_i^m$ in $s$. From this, it follows that $\prec$ and $\prec'$ are strongly equivalent when $a_i$ and $a_{i+1}$ do not mutually interfere. Given two actions $a$ and $a'$,

1. *$a$ does not interfere with the executability of $a'$* if for each assignment $x := e$ of $a$, $x$ does not occur in the preconditions of $a'$;
2. *$a$ does not interfere with the effects of $a'$* if for each assignment $x := e$ of $a$ $(i)$ $x$ does not occur in the assignments of $a'$, or $(ii)$ both $a$ and $a'$ assign $x$ with a linear increment, or $(iii)$ both $a$ and $a'$ assign $x$ with a simple assignment,
3. *$a$ and $a'$ do not mutually interfere* if $a$ does not interfere with the executability and the effects of $a'$ and also $a'$ does not interfere with the executability and effects of $a$.

**Theorem 6.** *Let $\Pi$ be a numeric planning problem. Let $\prec = a_1; \dots; a_k$ be a pattern, $k \geq 2$. Let $i \in [1, k)$. Let $\prec'$ be the pattern differing from $\prec$ only because $a_{i+1}$ precedes $a_i$ in $\prec'$. If $a_i$ and $a_{i+1}$ do not mutually interfere, $\prec$ and $\prec'$ are strongly equivalent.*

**Proof.** If $a_i$ and $a_{i+1}$ do not mutually interfere then for any state $s$, and for any $m, n \geq 0$, if and only if both $a_i^m$ and $a_{i+1}^n$ are executable in $s$, then

1. $res(a_{i+1}^n, res(a_i^m, s))$ is defined
2. $res(a_i^m, res(a_{i+1}^n, s))$ is defined,
3. $res(a_i^m, res(a_{i+1}^n, s)) = res(a_{i+1}^n, res(a_i^m, s))$.

The above facts follow from the non-mutual interference of $a_i$ and $a_{i+1}$.

Assume $\prec$ and $\prec'$ are not strongly equivalent. Then, for some initial state $s$, there is a goal state, e.g., in $R_s^{\prec}$ which is not in $R_s^{\prec'}$, which is possible only if there exists a state $s'' = res(a_{i+1}^n, res(a_i^m, s'))$ with $s' \in R_s^{\prec_{i-1}}$, $m, n \geq 0$, $s''$ in $R_s^{\prec_{i+1}}$ but not in $R_s^{\prec'_{i+1}}$. However, this is not possible since also $res(a_i^m, res(a_{i+1}^n, s'))$ is defined and equal to $s''$. □

Given the theorem, the problem of determining whether $\prec$ dominates and/or is dominated by $\prec'$ arises when $a_i$ and $a_{i+1}$ mutually interfere. Clearly, if $a_i$ and $a_{i+1}$ interfere in their effects, for some state $s$ and $m, n \geq 1$, we may have that executing $a_i^m; a_{i+1}^n$ or $a_{i+1}^n; a_i^m$ in $s$ leads to a different state and thus in the general case, $\prec'$ does not strongly dominate $\prec$ and $\prec$ does not strongly dominate $\prec'$. However, when either $(i)$ $a_i$ blocks $a_{i+1}$ or $(ii)$ $a_{i+1}$ supports $a_i$ and $a_i$ does not interfere with the executability of $a_{i+1}$ then $a_i^m; a_{i+1}^n$ is executable in a subset of the states in which $a_{i+1}^n; a_i^m$ is executable. An action $a$

1. *blocks* an action $a'$ if $a'$ contains a precondition which becomes contradictory once the variables $v$ are substituted with $e$ whenever $v := e$ is a simple assignment in `eff`$(a)$, and
2. *supports* $a'$ if $a$ interferes with the executability of $a'$ and for each precondition $p$ of $a'$ containing a variable $v$ assigned by $a$, $(i)$ $v$ is assigned by $a$ with a simple assignment, and $(ii)$ $p$ becomes valid once each variable $v$ is substituted with $e$ whenever $v := e \in$ `eff`$(a)$.

The above definitions of an action blocking/supporting another action are similar to the notion of disabling/enabling action in [17] in the classical setting. If $a_i^m; a_{i+1}^n$ is executable in a subset of the states in which $a_{i+1}^n; a_i^m$ is executable, and $a_i$ does not interfere with the effects of $a_{i+1}$ – and vice versa –, then $\prec'$ strongly dominates $\prec$.

**Theorem 7.** *Let $\Pi$ be a numeric planning problem. Let $\prec = a_1; \dots; a_k$ be a pattern, $k \geq 2$. Let $i \in [1, k)$. Let $\prec'$ be the pattern differing from $\prec$ only because $a_{i+1}$ precedes $a_i$ in $\prec'$. Assume that $a_i$ does not interfere with the effects of $a_{i+1}$ and vice versa. Assume that either $(i)$ $a_i$ blocks $a_{i+1}$, or $(ii)$ $a_{i+1}$ supports $a_i$ and $a_i$ does not interfere with the executability of $a_{i+1}$. Then, $\prec'$ strongly dominates $\prec$.*

**Proof.** We prove that for any state $s$, $R_s^{\prec_{i+1}} \subseteq R_s^{\prec'_{i+1}}$. Let $s$ be an arbitrary state.

For any $m, n \geq 0$, we prove that if $s'' \in R_s^{\prec_{i+1}}$, i.e., if $s'' = res(a_i^m; a_{i+1}^n, s')$ for some state $s' \in R_s^{\prec_{i-1}}$, then $s'' = res(a_{i+1}^n; a_i^m, s')$ and thus $s'' \in R_s^{\prec'_{i+1}}$.

For either $m = 0$ or $n = 0$, the sequence $a_i^m; a_{i+1}^n$ is equal to the sequence $a_{i+1}^n; a_i^m$ and hence $res(a_i^m; a_{i+1}^n, s') = res(a_{i+1}^n; a_i^m, s')$ and the thesis trivially follows.

Assume $m \geq 1$. If $a_i$ blocks $a_{i+1}$ then $a_{i+1}$ is not executable in $res(a_i^m, s)$ and thus $n = 0$, and this case is already covered by the previous one.

Assume also $n \geq 1$. If $a_i$ does not interfere with the executability and the effects of $a_{i+1}$ then since $a_{i+1}^n$ is executable in $res(a_i^m, s')$, $a_{i+1}^n$ is also executable in $s$. Further, since $a_{i+1}$ supports $a_i$, $a_i$ is executable in $res(a_{i+1}^n, s')$. Given that $res(a_{i+1}^n; a_i^m, s')$ is defined, $res(a_i^m; a_{i+1}^n, s') = res(a_{i+1}^n; a_i^m, s')$ follows from the hypothesis that $a_i$ and $a_{i+1}$ do not mutually interfere in their effects. □

**Example 7.** According to our definitions and considering the sets of actions at each level of the ARPG as in Eq. (7)

1. for level 0, each pair of distinct actions at this level do not mutually interfere except for the pairs {`lftl`, `lftr`}, {`rgtl`, `rgtr`}, {`lre`, `rle`}. Further, no action at this level blocks or support another action at the same level.
2. level 1 consists of the single action `conn`, and
3. for level 2, the action `disc` blocks the action `exch`.

Given the above, given two patterns $\prec$ and $\prec'$ extending the partial order induced by the ARPG, if `disc` follows `exch` in $\prec$, $\prec$ strongly dominates $\prec'$. Notice that the last three actions in $\prec$ are as in `conn`; `exch`; `disc`, where `conn` precedes `exch` because of the ARPG, and `exch` precedes `disc` because `disc` blocks `exch`. The fact that `conn`; `exch` is better than `exch`; `conn` is also a consequence of Theorem 7: `conn` supports `exch`, the two actions do not interfere in their effects and `exch` does not interfere with the executability of `conn`. Indeed, the order induced by the ARPG construction may correspond to the supporting relation (as in this case), but this is not always the case. Consider for example the modification of the example in which the two robots start in the same position and are already paired. In such a case, $x_l = x_r$ and $p = \top$ holds at level 0 and `conn`, `exch` and `disc` will all be at the same ARPG level. According to the ARPG partial ordering, the three actions can be put in any ordering, while Theorem 7 allows us to conclude that the pattern `conn`; `exch`; `disc` strongly dominates the other 5 orderings.

### 3.4. Plan quality

Thanks to Theorem 3, we know that any model $\mu$ of $\Pi^{\prec}$ corresponds to the valid plan $\pi = a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \dots; a_k^{\mu(a_k)}$. However, the discovered plan may include redundant actions.

**Example 8.** Assume, as in Example 1, that the initial state is $I = \{p = \bot, x_l = -X_I, x_r = X_I, q_l = Q, q_r = 0, q = 1\}$, where $X_I, Q$ are positive integers, and that $G = \{q_l = 0, q_r = Q, x_l = -X_I, x_r = X_I\}$.

If the pattern is computed using the ARPG construction outlined in the previous subsection, extended to order two actions at the same level using the results of Theorem 7, the pattern $\prec_I$ returned by COMPUTEPATTERN($\Pi$) in Algorithm 1 is, e.g.,

$$\prec_I = \texttt{lftl}; \texttt{rgtl}; \texttt{lftr}; \texttt{rgtr}; \texttt{rle}; \texttt{lre}; \texttt{conn}; \texttt{exch}; \texttt{disc},$$

and the procedure SPP($\Pi$) will determine the existence of a plan after two concatenations of the above pattern, i.e., with

$$\begin{aligned} \prec = \; & \texttt{lftl1}; \texttt{rgtl1}; \texttt{lftr1}; \texttt{rgtr1}; \texttt{rle1}; \texttt{lre1}; \texttt{conn1}; \texttt{exch1}; \texttt{disc1}; \\ & \texttt{lftl2}; \texttt{rgtl2}; \texttt{lftr2}; \texttt{rgtr2}; \texttt{rle2}; \texttt{lre2}; \texttt{conn2}; \texttt{exch2}; \texttt{disc2}. \end{aligned}$$

The model $\mu$ returned by SOLVE($\Pi^{\prec}$) will be such that

$$\begin{gathered} \mu(\texttt{lftl1}) = k, \quad \mu(\texttt{rgtl1}) = k + X_I, \\ \mu(\texttt{lftr1}) = X_I, \quad \mu(\texttt{rgtr1}) = 0, \\ \mu(\texttt{rle1}) = m, \quad \mu(\texttt{lre1}) = n, \\ \mu(\texttt{conn1}) = 1, \quad \mu(\texttt{exch1}) = Q, \quad \mu(\texttt{disc1}) = 1, \\ \mu(\texttt{lftl2}) = p + X_I, \quad \mu(\texttt{rgtl2}) = p, \\ \mu(\texttt{lftr2}) = 0, \quad \mu(\texttt{rgtr2}) = X, \\ \mu(\texttt{rle2}) = q, \quad \mu(\texttt{lre2}) = r, \\ \mu(\texttt{conn2}) = \mu(\texttt{exch2}) = \mu(\texttt{disc2}) = 0, \end{gathered}$$

for some $k, m, n, p, q, r \geq 0$ with $m, n, q, r \leq 1$ and $n = 1$ when $m = 1$. Any such plan corresponds to

1. having the left robot going to the left for $k$ times ($\mu(\texttt{lftl1}) = k$) and then to the right for $k + X$ times ($\mu(\texttt{rgtl1}) = k + X_I$) to reach the origin,
2. having the right robot going directly to the origin ($\mu(\texttt{lftr1}) = X_I$, $\mu(\texttt{rgtr1}) = 0$),
3. possibly enabling the right-to-left exchange ($\mu(\texttt{rle1}) = m \in [0, 1]$) and then surely enabling the left-to-right exchange ($\mu(\texttt{lre1}) = 1$) when $\mu(\texttt{rle1}) = 1$,
4. connecting, exchanging $Q$ objects and disconnecting ($\mu(\texttt{conn1}) = 1$, $\mu(\texttt{exch1}) = Q$, $\mu(\texttt{disc1}) = 1$),
5. having the left robot going to the left for $p + X_I$ times ($\mu(\texttt{lftl2}) = p + X_I$) and then to the right for $p$ times ($\mu(\texttt{rgtl2}) = p$) to reach the position it originally had,
6. having the right robot going directly to its original position ($\mu(\texttt{lftr2}) = 0$, $\mu(\texttt{rgtr2}) = X_I$),
7. (possibly) enabling the left-to-right and/or the right-to-left exchange ($\mu(\texttt{rle2}) = q$, $\mu(\texttt{lre2}) = r$).

In such plans, some actions can be executed even if unnecessary (e.g., `lftl1`, `lre`) or can be executed more times than necessary (e.g., `lftl1`). This does not happen when $k = m = n = p = q = r = 0$. In particular, $k = 0$ (resp. $p = 0$) corresponds to preventing the left robot from going unnecessarily to the left before (resp. after) connecting.

Notice that if rolling is disabled (i.e., if for every action $a$, $(a = 0 \vee a = 1)$ is imposed),

1. $\prec_I$ needs to be concatenated at least $2X_I + Q$ times in $\prec$ before SOLVE($\Pi^{\prec}$) becomes satisfiable, but
2. when $\prec$ is $\prec_I$ concatenated $2X_I + Q$ times, in any plan corresponding to a model of $\Pi^{\prec}$, $(i)$ no `lftl` useless action occurs before the `exch` action, and $(ii)$ no useless `rgtl` action occurs after the `exch` action.

Analogously, if in SPP($\Pi$) we do not allow executing two actions which are part of a same $\prec_I$ unless they do not mutually interfere (i.e., if for every pair of distinct mutually interfering actions $a$ and $a'$ in $\prec_I$, $(a = 0 \vee a' = 0)$ is imposed),

1. $\prec_I$ needs to be concatenated at least 5 times in $\prec$ before SOLVE($\Pi^{\prec}$) becomes satisfiable[7], but
2. when $\prec$ is $\prec_I$ concatenated 5 times, in any plan corresponding to a model of $\Pi^{\prec}$, $(i)$ no `lftl` useless action occurs before the `exch` action, and $(ii)$ no useless `rgtl` action occurs after the `exch` action.

As the example shows, the plan returned by SPP($\Pi$) may include unnecessary actions, especially when allowing for action rolling and/or the execution of mutually interfering actions which are part of the same initially computed pattern $\prec_I$. This fact is not surprising if SOLVE($\Pi^{\prec}$) is only required to compute one of the possibly infinitely many models of $\Pi^{\prec}$. Indeed, in some applications, it may be useful to look for a model $\mu$ whose corresponding plan $\pi$ is optimal according to some criteria. In our setting, we say that a plan $\pi$ corresponding to a model $\mu$ of $\Pi^{\prec}$ is

[7] In step 1 the robots moves until $x_l = x_r$, in step 2 they connect, in step 3 they exchange the objects, in step 4 they disconnect and in step 5 they return to the original positions.

1. *optimal* if, for any pattern $\prec'$, there does not exist a plan $\pi'$ corresponding to a model $\mu'$ of $\Pi^{\prec'}$, with fewer actions than $\pi$,
2. $\prec$*-optimal* if there does not exist another model $\mu'$ of $\Pi^{\prec}$ whose corresponding plan has fewer actions than $\mu$,
3. $\pi$*-optimal* if there does not exist another model $\mu'$ whose corresponding plan $\pi'$ is a subsequence of $\pi$.

Clearly, if a plan $\pi$ is optimal, then, for any pattern $\prec$, $\pi$ is also $\prec$-optimal, and, if $\pi$ is $\prec$-optimal, then it is also $\pi$-optimal. Any plan $\pi$ satisfying one of the above two last conditions is irredundant: removing some actions in $\pi$ leads to an invalid plan. Though returning an irredundant plan may be a desirable property, it comes with an extra price, since it is well known that checking whether a plan is irredundant is already co-NP-hard in the classical setting with no numeric variables (see, e.g., [18]).

While extending our work for computing optimal plans is not easy (see [19] for the most related recent work on the topic), a $\prec$-optimal and/or $\pi$-optimal plan correspond to a model of $\Pi^{\prec}$, which can be computed as follows. A model $\mu$ with corresponding plan $\pi$

1. is $\prec$-optimal if it minimizes $\sum_{i=1}^{k} a_i$, and
2. is $\pi$-optimal if it minimizes $\sum_{i=1}^{k} a_i$ subject to $a \leq \mu(a)$ for each $a \in A$, i.e., it is a subsequence of $\pi$.

Thus, it is relatively easy to find a $\prec$-optimal and/or $\pi$-optimal plan if the solver SMT also supports the minimization of $\sum_{i=1}^{k} a_i$, as, e.g., Z3 v4.12.2 [20], does. Other solutions are possible to improve the quality of the returned plan. Bofill et al., (2016) propose $(i)$ to call a standard SMT solver to find an initial model $\mu$ of $\Pi^{\prec}$, and then $(ii)$ call a MaxSMT solver on the problem $\Pi^{\prec} \cup \{a_i = 0 : \mu(a_i) = 0, i \in [1, k]\}$ together with $\{a_i = 0 : \mu(a_i) > 0, i \in [1, k]\}$, the latter treated as soft clauses (see the paper for more details). Building on the concepts introduced in classical planning by Giunchiglia and Maratea, (2007), another possibility for effectively computing models $\mu$ with a maximal set of actions $a$ such that $\mu(a) = 0$ is to prioritize the search for these solutions in the solver's heuristic, and some SMT solvers, such as MathSAT5 [22], offer native support for specifying the order in which the heuristic should operate (see, e.g., [21] for more details). Both these methods allow us to compute an irredundant plan assuming rolling actions is not possible. Other methods have been proposed, especially in the classical setting, some of which working in polynomial time, see, e.g., [18,23,24]

In any case, while these methods may reduce the number of executed actions, they do not guarantee to return an optimal plan. Indeed, an optimal plan may not correspond to a model of $\Pi^{\prec}$.

## 4. Relation to planning as satisfiability encodings

Let $\Pi = \langle V_B, V_N, A, I, G\rangle$ be a numeric planning problem. As briefly outlined in the introduction, in the standard planning as satisfiability framework the problem of finding a solution is solved by $(i)$ considering $n$ copies of a logical model of how actions cause transitions from one state to another, and $(ii)$ checking the existence of a solution starting with $n = 0$ transitions, and incrementing $n$ upon failure, see, e.g., [5]. Different approaches have been proposed, each characterized by how the transitions from one state to another are encoded as a logical formula.

In this section, we first formally define what is an encoding in the planning as satisfiability framework (Section 4.1), then we present the rolled-up and standard encodings (Section 4.2), the $R^2\exists$ encoding (Section 4.3), and how they are related to our pattern encoding when used in the planning as satisfiability framework (Section 4.4).

### 4.1. Planning as satisfiability

Formally, a *(planning as satisfiability) encoding* $E$ *of* $\Pi$ is a tuple

$$\Pi^E = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}(\mathcal{X}), \mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}'), \mathcal{G}(\mathcal{X})\rangle, \tag{8}$$

where $\mathcal{X}$ is a finite set of propositional and numeric *state variables* including $V_B \cup V_N$. $\mathcal{A}$ is a finite set of *action variables*, each one equipped with a domain representing the values it can take. $\mathcal{I}(\mathcal{X})$ and $\mathcal{G}(\mathcal{X})$ are the *initial state formula* and the *goal formula*, respectively, defined as in the previous section, with the difference that now the goal formula is simply the conjunction of the formulas in $G$, once each $v = \top$ and $v = \bot$ are substituted with $v$ and $\neg v$, respectively.

Each planning as satisfiability encoding is characterized by the definition of the *symbolic transition relation* $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$, a formula on the variables $\mathcal{X} \cup \mathcal{A} \cup \mathcal{X}'$, such that $\mathcal{X}' = \{v' \mid v \in \mathcal{X}\}$ is a copy of $\mathcal{X}$ and

1. *correctness*: each model $\mu$ of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ has to correspond to one sequence of actions $\alpha$ such that $(i)$ $\alpha$ is executable in the state $s$ such that, for each variable $v \in V_B \cup V_N$, $s(v) = \mu(v)$; and $(ii)$ the last state induced by $\alpha$ executed in $s$ is the state $s'$ such that, for each variable $v \in V_B \cup V_N$, $s'(v) = \mu(v')$, and
2. *completeness*: for each state $s$ and action $a$ in $A$, if $s'$ is the state resulting from the execution of $a$ in $s$, then there must be a model $\mu$ of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ such that for each variable $v \in V_B \cup V_N$, $s(v) = \mu(v)$ and $s'(v) = \mu(v')$.

Let $\Pi^E = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}(\mathcal{X}), \mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}'), \mathcal{G}(\mathcal{X})\rangle$ be an encoding of $\Pi$. In the planning as satisfiability approach [5], an integer $n \geq 0$ called *bound* or *number of steps* is fixed, $n + 1$ disjoint copies $\mathcal{X}_0, \dots, \mathcal{X}_n$ of the set $\mathcal{X}$ of state variables, and $n$ disjoint copies $\mathcal{A}_0, \dots, \mathcal{A}_{n-1}$ of the set $\mathcal{A}$ of action variables are made, and then

1. $\mathcal{I}(\mathcal{X}_0)$ is the formula in the variables $\mathcal{X}_0$ obtained by substituting each variable $x \in \mathcal{X}$ with $x_0 \in \mathcal{X}_0$ in $\mathcal{I}(\mathcal{X})$;
2. for each step $i = 0, \dots, n-1$, $\mathcal{T}(\mathcal{X}_i, \mathcal{A}_i, \mathcal{X}_{i+1})$ is the formula in the variables $\mathcal{X}_i \cup \mathcal{A}_i \cup \mathcal{X}_{i+1}$ obtained by substituting each variable $x \in \mathcal{X}$ (resp. $a \in \mathcal{A}$, $x' \in \mathcal{X}'$) with $x_i \in \mathcal{X}_i$ (resp. $a_i \in \mathcal{A}_i$, $x_{i+1} \in \mathcal{X}_{i+1}$) in $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$;

3. $\mathcal{G}(\mathcal{X}_n)$ is the formula in the variables $\mathcal{X}_n$ obtained by substituting each variable $x \in \mathcal{X}$ with $x_n \in \mathcal{X}_n$ in $\mathcal{G}(\mathcal{X})$.

Then, the *(planning as satisfiability) encoding* $\Pi^E$ *of* $\Pi$ *with bound* $n$ is the formula

$$\Pi_n^E = \mathcal{I}(\mathcal{X}_0) \wedge \bigwedge_{i=0}^{n-1} \mathcal{T}(\mathcal{X}_i, \mathcal{A}_i, \mathcal{X}_{i+1}) \wedge \mathcal{G}(\mathcal{X}_n). \tag{9}$$

The plan corresponding to a model $\mu$ of $\Pi_n^E$ is the sequence of actions $\alpha_0; \dots; \alpha_{n-1}$, where each $\alpha_i$ is the sequence of actions corresponding to the model of $\mathcal{T}(\mathcal{X}_i, \mathcal{A}_i, \mathcal{X}_{i+1})$ obtained by restricting $\mu$ to $\mathcal{X}_i \cup \mathcal{A}_i \cup \mathcal{X}_{i+1}$, $i \in [0, n)$. The standard procedure for computing a plan for $\Pi$ is to

1. start fixing the bound $n$ to 0,
2. check the existence of a model $\mu$ for $\Pi_n^E$, returning the plan corresponding to $\mu$ if such $\mu$ is determined, and
3. increment $n$ and repeat the previous step, otherwise.

The correctness of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ ensures the *correctness of* $\Pi^E$: each sequence of actions returned by the standard procedure using $\Pi^E$ is a plan. The completeness of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ ensures also the *completeness of* $\Pi^E$: if there exists a plan for $\Pi$ of length $n$, $\Pi_n^E$ is satisfiable and the standard procedure using $\Pi^E$ will return a plan (notice that, depending on the encoding, $\Pi_k^E$ can become satisfiable even for some $k < n$).

It is clear that the number of variables and size of (9) increase with the bound $n$, explaining why much of the research in planning as satisfiability has concentrated on how to produce encodings allowing to find plans with the lowest possible bound $n$.

## 4.2. Rolled-up and standard encodings

In the state-of-the-art *rolled-up encoding* $\Pi^R$ *of* $\Pi$ proposed in [6], each action $a \in A$ is defined as an action variable which can get an arbitrary value $k \in \mathbb{N}$, corresponding to have $k$ (consecutive) occurrences of $a$. [8] Then, the symbolic transition relation $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ of $\Pi^R$ is the conjunction of the formulas in the following sets:

1. $\mathrm{pre}^R(A)$, consisting of, for each $a \in A$, $v = \bot$ and $w = \top$ in $\mathrm{pre}(a)$,

   $$a > 0 \rightarrow (\neg v \wedge w),$$

   and, for each $a \in A$ and $\psi \trianglerighteq 0$ in $\mathrm{pre}(a)$,

   $$a > 0 \rightarrow \psi \trianglerighteq 0, \quad a > 1 \rightarrow \psi[a] \trianglerighteq 0,$$

   where $\psi[a]$ is the linear expression obtained from $\psi$ by substituting each variable $x$ with
   (a) $x + (a-1) \times \psi_1$, whenever $x \mathrel{+}= \psi_1 \in \mathtt{eff}(a)$ is a linear increment,
   (b) $\psi_1$, if $x := \psi_1 \in \mathrm{eff}(a)$ is a simple assignment.
   The last two formulas ensure that $\psi \trianglerighteq 0$ holds in the states in which the first and the last execution of $a$ happens (See [6]).
2. $\mathrm{eff}^R(A)$, consisting of, for each $a \in A$, $v := \bot$, $w := \top$, linear increment $x \mathrel{+}= \psi$ and general assignment $y := \psi_1$ in $\mathrm{eff}(a)$,

   $$a > 0 \rightarrow (\neg v' \wedge w' \wedge x' = x + a \times \psi \wedge y' = \psi_1).$$

3. $\mathrm{frame}^R(V_B \cup V_N)$, consisting of, for each variable $v \in V_B$ and $w \in V_N$,

   $$\bigwedge_{a:\ v:=\top \in \mathrm{eff}(a)} a = 0 \wedge \bigwedge_{a:\ v:=\bot \in \mathrm{eff}(a)} a = 0 \rightarrow v' \equiv v,$$

   $$\bigwedge_{a:\ w:=\psi \in \mathrm{eff}(a)} a = 0 \rightarrow w' = w.$$

4. $\mathrm{mutex}^R(A)$, consisting of $(a_1 = 0 \vee a_2 = 0)$, for each pair of distinct actions $a_1$ and $a_2$ which are in mutex. Two distinct actions, $a_1$ and $a_2$, are in *mutex* whenever there exists a variable assigned by $a_1$ which occurs either in $\mathrm{pre}(a_2)$ or in the right-hand side of an assignment in $\mathrm{eff}(a_2)$. [9]
5. $\mathrm{amo}^R(A)$, consisting of, for each action $a$ not eligible for rolling,

   $$(a = 0 \vee a = 1).$$

Notice that if for action $a$ the formula $(a = 0 \vee a = 1)$ belongs to $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$, we can equivalently $(i)$ define $a$ as a Boolean variable, and then $(ii)$ replace $a = 0$, $a > 0$, $a = 1$ and $a > 1$ with $\neg a$, $a$, $a$ and $\bot$, respectively, in $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$. It is clear that if $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ contains $(a = 0 \vee a = 1)$ for any action $a$, then the rolled-up encoding $\Pi^R$ reduces to the standard encoding as defined, e.g., in [14]. Equivalently, in the *standard encoding* $\Pi^S$ *of* $\Pi$, the symbolic transition relation $\mathcal{T}^S(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ is obtained by adding, for each action $a$, $(a = 0 \vee a = 1)$ to $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$. The decoding function of the rolled-up (resp. standard) encoding associates to each model $\mu$ of $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ (resp. $\mathcal{T}^S(\mathcal{X}, \mathcal{A}, \mathcal{X}')$) the sequences of actions in which each action $a$ occurs $\mu(a)$ times. The rolled-up and standard encoding are correct and complete [6].

[8] To ease the presentation, our definition of $\Pi^R$ considers just the cases $\alpha = 0$ and $\alpha = 1$ of Theorem 1 in [6], as we did in the previous section.

[9] Notice that if two actions mutually interfere then they are also in mutex, while the vice versa does not necessarily hold. For instance, two actions $a_1$ and $a_2$ with $x := x + 1$ in their effects are in mutex but do not mutually interfere, and allowing for both $a_1 > 0$ and $a_2 > 0$ in the $R$ encoding leads to models not corresponding to valid plans.

**Theorem 8** (Scala et al. (2016)). *Let $\Pi$ be a numeric planning problem. The planning as satisfiability rolled-up encoding $\Pi^R$ and the standard encoding $\Pi^S$ are both correct and complete.*

### *4.3. Relaxed-relaxed ∃ ($R^2\exists$) encoding*

A problem with the rolled-up and standard encodings is the presence of the axioms in $\mathrm{mutex}(A)$, which forces some actions to be set to 0 even when there exists an ordering allowing to execute them sequentially starting from a state $s$, see, e.g., [17]. Indeed, allowing to set more actions to a value $> 0$ while maintaining correctness and completeness of the encoding, allows finding solutions to (9) with a lower value for the bound. Several proposals along these lines have been made. Here we present the $R^2\exists$ encoding firstly proposed by [7] for classical planning and then extended for numeric planning by [25], which is arguably the state-of-the-art encoding in which actions are encoded as Boolean variables.

In the $R^2\exists$ encoding, action variables are Boolean and assumed to be ordered according to a given total order. Different orderings lead to different $R^2\exists$ encodings. In the following, we represent the total ordering as an elementary and complete[10] pattern.

Consider an elementary and complete pattern $\prec = a_1; a_2; \dots; a_k$, $k \geq 0$. We denote the $R^2\exists$ *$\prec$-encoding of* $\Pi$ as $\Pi^{R^2\exists,\prec}$. In $\Pi^{R^2\exists,\prec}$, for each action $a$ and variable $v$ assigned by $a$, a newly introduced variable $v^a$ with the same domain of $v$ is added to the set $\mathcal{X}$ of state variables. Intuitively, each new variable $v^a$ represents the value of $v$ after the sequential execution of some actions in the initial sequence of $\prec$ ending with $a$. The symbolic transition relation $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ of $\Pi^{R^2\exists,\prec}$ is the conjunction of the formulas in the following sets:

1. $\mathrm{pre}^{R^2\exists,\prec}(A)$, consisting of, for each $a \in A$, $v = \bot$, $w = \top$ and $\psi \trianglerighteq 0$ in $\mathrm{pre}(a)$,

   $$a \rightarrow (\neg v^{\ll,a} \wedge w^{\ll,a} \wedge \psi^{\ll,a} \trianglerighteq 0),$$

   where, for each variable $x \in V_B \cup V_N$, $x^{\ll,a}$ stands for the variable $(i)$ $x$, if there is no action preceding $a$ in $\prec$ assigning $x$; and $(ii)$ $x^b$, if $b$ is the last action assigning $x$ preceding $a$ in $\prec$. Analogously, $\psi^{\ll,a}$ is the linear expression obtained from $\psi$ by substituting each variable $x \in V_N$ with $x^{\ll,a}$.
2. $\mathrm{eff}^{R^2\exists,\prec}(A)$, consisting of, for each $a \in A$, $v := \bot$, $w := \top$ and general assignment $x := \psi$ in $\mathrm{eff}(a)$,

   $$a \rightarrow (\neg v^a \wedge w^a \wedge x^a = \psi^{\ll,a}),$$
   $$\neg a \rightarrow (v^a \leftrightarrow v^{\ll,a} \wedge w^a \leftrightarrow w^{\ll,a} \wedge x^a = x^{\ll,a}).$$

3. $\mathrm{frame}^{R^2\exists,\prec}(V_B \cup V_N)$, consisting of, for each variable $v \in V_B$ and $w \in V_N$,

   $$v' \leftrightarrow v^{\ll,g}, \quad w' = w^{\ll,g},$$

   where $g$ is a dummy action following all the other actions in $\prec$.

The decoding function of the $R^2\exists$ $\prec$-encoding associates to each model $\mu$ of $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ the sequence of actions obtained from $\prec$ by deleting the actions $a$ with $\mu(a) = \bot$. In the $R^2\exists$ $\prec$-encoding, there are no mutex axioms and the size of $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ is linear in the size of $\Pi$. However, it introduces many new state variables (in the worst case, $|V_B \cup V_N| \times |A|$). The $R^2\exists$ $\prec$-encoding of $\Pi$ is correct and complete.

**Theorem 9** (Bofill et al., (2017)). *Let $\Pi$ be a numeric planning problem. Let $\prec$ be an elementary and complete pattern. The planning as satisfiability $R^2\exists$ $\prec$-encoding $\Pi^{R^2\exists,\prec}$ is correct and complete.*

### *4.4. Relationships among the standard, rolled-up, relaxed-relaxed exists and pattern encodings*

Consider an elementary and complete pattern $\prec$. Since $\prec$ is elementary and complete, the $\prec$-symbolic transition relation $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ can be used in the planning as satisfiability framework, allowing for a direct comparison between the so far proposed planning as satisfiability encoding and the $\prec$-encoding in the planning as satisfiability framework. Given this, we write

1. $\Pi^{S,\prec}$ for the planning as satisfiability encoding (8) in which the symbolic transition relation $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ is $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ as defined in Section 3.2, and
2. $\Pi_n^{S,\prec}$ for the corresponding planning as satisfiability encoding with bound $n$.

Of course, the planning as satisfiability pattern $\prec$-encoding $\Pi^{S,\prec}$ is correct and complete.

**Theorem 10.** *Let $\Pi$ be a numeric planning problem. Let $\prec$ be an elementary and complete pattern. The planning as satisfiability pattern $\prec$-encoding $\Pi^{S,\prec}$ is correct and complete.*

**Proof.** If $\Pi^{S,\prec}$ is either incorrect or incomplete, Theorem 3 does not hold for any problem $\Pi$. □

Comparing the planning as satisfiability pattern $\prec$-encoding $\Pi^{S,\prec}$ with the rolled-up $\Pi^R$ and the $\Pi^{R^2\exists,\prec}$ encoding, $\Pi^{S,\prec}$ allows in a single state transition

[10] We recall the definition of Section 3.2: a pattern is elementary if the same action doesn't appear multiple times in it, and complete if all the actions in $A$ appear in it.

1. the multiple consecutive execution of the same action as in the $\Pi^R$ encoding, and
2. the combination of multiple, even contradictory effects on a same variable by different actions, as in the $R^2\exists$ encoding.

Because of this, $\Pi^{S,\prec}$ dominates both $\Pi^R$ and $\Pi^{R^2\exists,\prec}$, and the latter two dominate the standard encoding $\Pi^S$. Given two planning as satisfiability encodings $E_1$ and $E_2$ we say that $E_1$ *dominates* $E_2$ if, for each bound $n$, $\Pi_n^{E_2}$ satisfiability implies that also $\Pi_n^{E_1}$ is satisfiable. Thus, if $E_1$ dominates $E_2$, assuming the correctness of the two encodings and that a plan will be searched by incrementally increasing the bound starting from 0, $E_2$ will never find a plan with a bound lower than the one needed by $E_1$.

**Theorem 11.** *Let $\Pi$ be a numeric planning problem. Let $\prec$ be an elementary and complete pattern. The planning as satisfiability SPP $\prec$-encoding $\Pi^{S,\prec}$ dominates the rolled-up encoding $\Pi^R$ and the $R^2\exists$ $\prec$-encoding $\Pi^{R^2\exists,\prec}$. Both $\Pi^R$ and $\Pi^{R^2\exists,\prec}$ dominate the standard encoding $\Pi^S$.*

**Proof.** We prove the various statements one by one. Since $\prec$ is elementary and complete, we can write $\mathcal{A}$ instead of $\mathcal{A}^\prec$.

1. $\Pi^{S,\prec}$ dominates $\Pi^R$. We have to prove that, for any bound $n$, if $\Pi_n^R$ is satisfiable then also $\Pi_n^{S,\prec}$ is satisfiable, which follows from the fact that any model $\mu$ of $\mathcal{T}^R(\mathcal{X},\mathcal{A},\mathcal{X}')$ is also a model of $\mathcal{T}^\prec(\mathcal{X},\mathcal{A},\mathcal{X}')$. Let $\mu$ be a model of $\mathcal{T}^R(\mathcal{X},\mathcal{A},\mathcal{X}')$ and $\alpha$ be the sequence of actions corresponding to the model $\mu$. Clearly, $\alpha$ is a valid plan for the planning problem $\Pi_\mu = \langle V_B, V_N, A, I_\mu, G_\mu\rangle$ in which $I_\mu$ is the restriction of $\mu$ to $V_B \cup V_N$ and $G_\mu = \bigwedge_{v\in V_B:\mu(v')=\top} v \wedge \bigwedge_{v\in V_B:\mu(v')=\bot} \neg v \wedge \bigwedge_{v\in V_N} v = \mu(v')$, i.e., the planning problem in which the initial state and the goal formula corresponds to the values assigned by $\mu$ to the variables in $\mathcal{X} = V_B \cup V_N$ and $\mathcal{X}'$. From the completeness of the pattern encoding, the pattern $\alpha$-encoding of $\Pi_\mu$ is satisfiable. Then, also the SPP $\prec$-encoding of $\Pi_\mu$ is satisfiable since:
   (a) any two actions in $\alpha$ do not mutually interfere, and thus we can reorder the actions in $\alpha$ as to respect the ordering in $\prec$ (Theorem 7), and
   (b) for each action $a \notin \alpha$, $\mu(a) = 0$.
2. $\Pi^{S,\prec}$ dominates $\Pi^{R^2\exists,\prec}$. As in the previous case, we prove that any model $\mu$ of $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X},\mathcal{A},\mathcal{X}')$ is also a model of $\mathcal{T}^\prec(\mathcal{X},\mathcal{A},\mathcal{X}')$. Let $\mu$ be a model of $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X},\mathcal{A},\mathcal{X}')$ and $\alpha$ be the sequence of actions corresponding to the model $\mu$. The sequence $\alpha$ is a subsequence of $\prec$ and thus, by the completeness of the SPP $\prec$-encoding, is also a model of $\mathcal{T}^\prec(\mathcal{X},\mathcal{A},\mathcal{X}')$.
3. $\Pi^R$ dominates $\Pi^S$. The fact that $\Pi^R$ dominates $\Pi^S$ follows from the monotonicity of first order logic: the formulas in $\Pi^S$ are a subset of the formulas in $\Pi^R$, and thus if $\Pi^S$ is satisfiable, so $\Pi^R$ is.
4. $\Pi^{R^2\exists,\prec}$ dominates $\Pi^S$. For simplicity, we assume action variables in $\mathcal{T}^S(\mathcal{X},\mathcal{A},\mathcal{X}')$ are Boolean, i.e., that $a = 0$ corresponds to $\neg a$ and $a = 1$ to $a$. Let $\mu$ be a model of $\mathcal{T}^S(\mathcal{X},\mathcal{A},\mathcal{X}')$. Because of the effect and mutex axioms in $\Pi^S$, for each variable $v$ and action $a$ such that $\mu(a) = 1$, $v := e \in \mathrm{eff}(a)$, $\mu(v') = \mu(e)$, and we can extend $\mu$ to be a model of $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X},\mathcal{A},\mathcal{X}')$ by assigning $\mu(v^a) = \mu(e)$.

□

In the example below, we show that for any two distinct encodings in $\{\Pi^S, \Pi^R, \Pi^{R^2\exists,\prec}, \Pi^{S,\prec}\}$, the only dominance relations that hold are the ones established in the theorem.

**Example 9.** The rolled-up (resp. standard) encoding of the two robots problem admits a model with bound $n_R = 5$ (resp. $n_S = 2X_I + Q + 2$, and thus $n_S = n_R$ only when $X_I = Q = 1$). Assuming that in $\prec$ actions are ordered as in the plan (2), $\Pi_n^{S,\prec}$ is satisfiable when $n = n_{S,\prec} = 1 < n_R$, while $\Pi_n^{R^2\exists,\prec}$ is satisfiable when $n = n_{R^2\exists,\prec} = 2(X_I - 1) + Q$, and thus $n_{R^2\exists,\prec} = n_{S,\prec}$ if and only if $X_I = Q = 1$, and $n_{R^2\exists,\prec} \le n_R$ if and only if $2(X_I - 1) + Q \le 5$. If actions in $\prec$ are not ordered as in the plan (2), the bound needed by $\Pi^{S,\prec}$ and $\Pi^{R^2\exists,\prec}$ increase. In the worst case, $\Pi^{S,\prec}$ (resp. $\Pi^{R^2\exists,\prec}$) admits a solution with a bound equal to the one needed by $\Pi^R$ (resp. $\Pi^S$), and this happens when actions in $\prec$ are in reverse order wrt the plan (2).

## 5. Implementation and experimental analysis

In this section, we first experimentally analyse the performance of the basic procedure in Algorithm 1 when

1. exploiting the pattern selection procedure used in [2] enhanced with the results presented in Section 3.3 (Section 5.1), and
2. implementing the strategies presented in Section 3.4 in order to return plans with higher quality (Section 5.2).

Then, we perform a comparative analysis with all the publicly available state-of-the-art symbolic (Section 5.3) and search-based (Section 5.4) numeric planners, summarizing the results with all the considered planners in the final Section 5.5.

For the experiments, we adopted the same settings used in the Agile Track of the 2023 Numeric IPC [13]. In particular, we considered all its 20 domains and 20 problems per domain, to which we added 20 problems of the LineExchange domain. The added domain generalizes Example 1 by having $N = 4$ robots on a line which can exchange items while staying in their adjacent segments of length $D = 2$. In particular, in the initial state, the first robot has $Q \in \mathbb{N}_0$ items and the goal is to transfer all the items to the last robot in the line. For every problem, we set the time limit to 5 min, on an Intel Xeon Platinum 8000 3.1GHz with 8 GB of RAM. We performed some experiments with a time limit of 30 min and obtained the same qualitative results.

We summarize the results in every domain via tables, in which

1. Both the names of the solvers and of the domains are abbreviated to save space. Further, each domain is labelled with “S” (for simple) if every numeric effect of each action either increases or decreases the assigned variable by a constant, and is labelled with “L” (for linear), otherwise.

2. For each domain in the table, we always show the number of problems solved (Sub-table Solved), and dummyTXdummy– the average time needed to find a solution, counting the time limit when a solution could not be found (Sub-table Time). A "-" indicates that no problem in the domain was solved with the given resources.
3. The best results are in bold. We also include a final line, labelled *Best*, reporting on how many of the considered 420 problems each planner obtained the best result. [11]

All the considered symbolic planners have been run using Z3 v4.12.2 [20] for checking the satisfiability of the formula (9), represented as a set of assertions in the SMTLIB format [16]. All our systems have been implemented as part of the PATTY system, and are publicly available[12], together with the LINEEXCHANGE domain and the problems used in this paper. In our systems, each lifted input domain specified in PDDL2.1 is first grounded by instantiating its variables over the objects defined in the corresponding problem, considering all possible combinations. The resulting numeric planning problem is then simplified by eliminating actions that can never be executable, since they contain a precondition that is falsified in the initial state by variables that are never modified by any action.

## 5.1. Impact of the computing pattern procedure

As already discussed in the previous sections, how the pattern is selected can have a dramatic impact on the performance of the SPP procedure. Assuming the existence of a plan of length $n$, the SPP procedure in Algorithm 1 needs from 1 to $n$ iterations before finding it, how many depending on the characteristics of the planning problem and of the selected pattern.

In our previous paper [2], the pattern was selected by exploiting the ARPG construction informally presented in Section 3.3. Here we extend the system $\text{PATTY}_\text{A}$ implementing such a strategy, by ensuring that action $a_1$ precedes action $a_2$ in the pattern when both are at the same ARPG level, and either $a_2$ blocks $a_1$, or $a_1$ supports $a_2$ without $a_2$ interfering with the executability of $a_1$. We refer to the resulting system as $\text{PATTY}_\text{E}$. Both $\text{PATTY}_\text{A}$ and $\text{PATTY}_\text{E}$ lexicographically order any two actions $a_1$ and $a_2$ whenever they are not ordered according to the previous criteria. [13] Finally, for each problem, we evaluated five different versions of PATTY, each using a different randomly generated pattern. To summarize PATTY's performance across all problems and domains with these random patterns, we followed these steps:

1. for each problem, we sorted the five obtained results by solving time, and
2. selected the first, third, and fifth results to represent the performance of the three virtual planners $\text{PATTY}_\text{R}^{min}$, $\text{PATTY}_\text{R}^{med}$, and $\text{PATTY}_\text{R}^{max}$, respectively. $\text{PATTY}_\text{R}^{min}$, $\text{PATTY}_\text{R}^{med}$, and $\text{PATTY}_\text{R}^{max}$ results indicate the performance that can be expected in the best, median, and worst case, when using a randomly generated pattern.

Table 1 summarizes the results. In the sub-tables/columns, beside the information on the number of solved problems and on the average time, we report the average number of calls to the SMT solver (Sub-table SMT calls). To enable meaningful comparisons, the number of SMT solver calls was calculated considering only the problems solved by all the planners able to solve at least one problem in the domain. We remind that the number of SMT calls is equal to both the number $n$ of iterations and the number of times the initially computed pattern $\prec_I$ needs to be concatenated to find a valid plan.

As it can be seen, the results align with the theoretical finding that $\text{PATTY}_\text{E}$ dominates $\text{PATTY}_\text{A}$, as the latter never exhibits a lower number of calls to the SMT solver than the former. Further, the enhanced pattern computation of $\text{PATTY}_\text{E}$ produces some effects on 6 out of the 12 domains, with problems requiring more than one call to the SMT solver. Still, although $\text{PATTY}_\text{E}$ dominates $\text{PATTY}_\text{A}$, the latter solves more problems in two domains (balanced by $\text{PATTY}_\text{E}$'s superior results in three other domains). Indeed, for a problem in each of these two domains, the SMT solver manages to find a solution on $\text{PATTY}_\text{A}$ encoding while it fails on $\text{PATTY}_\text{E}$ encoding.

Considering also the performance of $\text{PATTY}_\text{R}^{min}$, $\text{PATTY}_\text{R}^{med}$ and $\text{PATTY}_\text{R}^{max}$ the following observations are in order:

1. on some domains (Like BLGRP (S) and CNT (S)) the pattern selection does not have an impact: all the problems in these domains are solved by concatenating the pattern just once, even when the pattern is randomly generated,
2. on some other domains (significantly, HPWR (S)) the pattern selection does have an impact: the ARPG based pattern construction is very productive, while the random generation of patterns is not,
3. yet on some other domains (and in particular, DRN (S)) the random generation of pattern seems to be better: indeed, on average DRN (S) problems require more than 4 iterations to be solved, and exploiting a different pattern (even a randomly generated one), likely from the second iteration on, leads to a lower number of calls to the SMT solver (though not necessarily to best performance).

Overall, the pattern computation procedure used by $\text{PATTY}_\text{E}$ (resp. $\text{PATTY}_\text{A}$) increments the number of solved problems by the 8.5%/12.9%/23.8% (resp. 6.6%/10.8%/21.6%) wrt $\text{PATTY}_\text{R}^{min}$/$\text{PATTY}_\text{R}^{med}$/ $\text{PATTY}_\text{R}^{max}$. Not surprisingly, $\text{PATTY}_\text{R}^{min}$ has the best time performance on most problems: indeed, on the 259 problems that $\text{PATTY}_\text{R}^{min}$ solves, it gets by construction the best time result out of 5 different runs.

[11] For the number of solved problems, the last line is just the sum of the previous ones. Notice that the sum of the numbers in the last line do not sum up to the total number of problems –as it could be expected– since some problems ($i$) are not solved by any planner in the table, and/or ($ii$) are solved with the same best result by more than one planner.

[12] http://pattyplan.com

[13] Here, differently from [2], we also introduced the lexicographic ordering to uniquely characterize the used pattern.

**Table 1**
Comparative analysis among Patty$_E$, Patty$_A$, Patty$_R^{min}$, Patty$_R^{med}$ and Patty$_R^{max}$.

| | Solved (out of 20) | | | | | Time (s) | | | | | SMT calls | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | $P_E$ | $P_A$ | $P_R^{min}$ | $P_R^{med}$ | $P_R^{max}$ | $P_E$ | $P_A$ | $P_R^{min}$ | $P_R^{med}$ | $P_R^{max}$ | $P_E$ | $P_A$ | $P_R^{min}$ | $P_R^{med}$ | $P_R^{max}$ |
| BlGrp (S) | **20** | **20** | **20** | **20** | **20** | 1.8 | **1.6** | **1.6** | 1.7 | 1.8 | **1.0** | **1.0** | **1.0** | **1.0** | **1.0** |
| Cnt (S) | **20** | **20** | **20** | **20** | **20** | 0.9 | 0.9 | **0.8** | 0.9 | 1.0 | **1.0** | **1.0** | **1.0** | **1.0** | **1.0** |
| Cnt (L) | **20** | **20** | **20** | **20** | 17 | 1.1 | **0.9** | 11.2 | 34.4 | 105.2 | **1.0** | **1.0** | 1.9 | 1.9 | 1.9 |
| Del (S) | 5 | 3 | **6** | **6** | 4 | 226.4 | 256.0 | **208.3** | 212.9 | 236.0 | **1.7** | 3.3 | 2.3 | 2.3 | 3.0 |
| Drn (S) | 3 | 3 | **5** | 3 | 3 | 255.3 | 255.2 | **246.5** | 255.2 | 255.4 | 5.7 | 5.7 | **4.3** | 5.0 | 5.3 |
| Exp (S) | 2 | 2 | **3** | **3** | 2 | 270.2 | 273.9 | **257.9** | 261.0 | 275.9 | **3.0** | 6.0 | 5.0 | 5.5 | 7.5 |
| Farm (S) | **20** | **20** | **20** | **20** | **20** | 2.4 | 2.8 | **0.9** | 2.1 | 7.3 | **1.0** | **1.0** | **1.0** | 1.1 | 1.1 |
| Farm (L) | **20** | **20** | **20** | **20** | 19 | 2.7 | 2.7 | **1.1** | 2.9 | 27.4 | **1.0** | **1.0** | 1.1 | 1.1 | 1.4 |
| HPwr (S) | **20** | **20** | 1 | – | – | **9.4** | 22.9 | 295.2 | – | – | **1.0** | **1.0** | 7.0 | – | – |
| Mrkt (L) | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| MPrime (S) | **12** | 10 | 11 | 9 | 8 | **137.7** | 166.2 | 165.2 | 185.1 | 207.8 | **1.1** | 2.0 | 2.0 | 2.6 | 3.1 |
| PathM (S) | 18 | **19** | 13 | 12 | 6 | 42.3 | **37.2** | 117.9 | 147.7 | 232.7 | **1.0** | **1.0** | 2.8 | 3.7 | 3.8 |
| PlWat (S) | 6 | 6 | **7** | 6 | 6 | 215.3 | 217.8 | **198.0** | 212.8 | 214.4 | 7.6 | 7.6 | **6.8** | 7.8 | 8.8 |
| Rvr (S) | 15 | 11 | **16** | **16** | 11 | 101.4 | 149.4 | **96.7** | 124.9 | 166.7 | **1.7** | 2.4 | 2.8 | 3.0 | 3.9 |
| Sail (S) | **20** | **20** | **20** | **20** | **20** | 3.6 | 1.1 | **0.9** | 1.2 | 24.4 | 3.3 | 3.3 | **2.3** | 2.8 | 3.0 |
| Sail (L) | 19 | **20** | **20** | **20** | 19 | 16.3 | 8.2 | **0.9** | 1.0 | 16.2 | 1.5 | 1.5 | **1.2** | 1.6 | 1.8 |
| Stlrs (S) | 8 | **9** | 3 | – | – | 210.5 | **193.1** | 265.8 | – | – | **1.0** | **1.0** | 2.3 | – | – |
| Sgr (S) | **20** | **20** | **20** | **20** | **20** | 10.3 | 14.6 | **6.2** | 14.4 | 28.8 | **2.5** | 3.1 | 2.8 | 3.3 | 3.5 |
| Tpp (L) | 2 | 2 | **3** | **3** | 2 | 270.2 | 270.2 | **259.3** | 260.4 | 270.6 | **2.5** | **2.5** | **2.5** | **2.5** | 3.5 |
| Zeno (S) | **11** | **11** | **11** | **11** | **11** | **136.4** | **136.4** | 137.7 | 138.8 | 140.7 | **1.6** | **1.6** | 2.7 | 3.0 | 3.5 |
| Line (L) | **20** | **20** | **20** | **20** | 19 | **1.2** | 8.3 | 2.1 | 3.4 | 50.4 | **2.8** | 4.7 | 4.4 | 5.0 | 5.7 |
| *Best* | **281** | **276** | **259** | **249** | **227** | **85** | **86** | **134** | **2** | **0** | **256** | **197** | **156** | **115** | **92** |

**Table 2**
Comparative analysis between Patty$_E$, Patty$_M$, Patty$_I$ and Patty$_C$. Each domain is labeled with S (for simple) if every numeric effect of each action either increases or decreases by a constant the assigned variable, and with L (for linear), otherwise. In the table, names have been abbreviated to save space. See [13] for more details. Best results are in bold.

| | Solved (out of 20) | | | | Time (s) | | | | Plan length | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | $P_E$ | $P_M$ | $P_I$ | $P_C$ | $P_E$ | $P_M$ | $P_I$ | $P_C$ | $P_E$ | $P_M$ | $P_I$ | $P_C$ |
| BlGrp (S) | **20** | **20** | 19 | **20** | **1.8** | 32.1 | 2.5 | 8.8 | 626 | **226** | 279 | 609 |
| Cnt (S) | **20** | 19 | **20** | **20** | **0.9** | 34.4 | 1.3 | 5.1 | 533 | **352** | 408 | 496 |
| Cnt (L) | **20** | 11 | 14 | **20** | **1.1** | 144.6 | 140.1 | **1.1** | 52 | **26** | 31 | 34 |
| Del (S) | **5** | 3 | **5** | **5** | 226.4 | 255.4 | **214.4** | 226.5 | 19 | **16** | 17 | **16** |
| Drn (S) | **3** | **3** | **3** | **3** | **255.3** | 256.5 | **255.3** | **255.3** | 25 | **12** | **12** | 14 |
| Exp (S) | 2 | **3** | 2 | 2 | 270.2 | **253.4** | 270.2 | 270.3 | 38 | **28** | **28** | **28** |
| Farm (S) | **20** | 12 | 19 | **20** | **2.4** | 154.8 | 49.5 | 7.7 | 740 | **275** | 339 | 382 |
| Farm (L) | **20** | 4 | 15 | **20** | **2.7** | 265.0 | 108.0 | 15.8 | 412 | **16** | 178 | 386 |
| HPwr (S) | **20** | 4 | 18 | **20** | **9.4** | 269.8 | 54.5 | 15.5 | 56 | **32** | 38 | 40 |
| Mrkt (L) | – | – | – | – | – | – | – | – | – | – | – | – |
| MPrime (S) | **12** | 8 | 10 | **12** | 137.7 | 194.5 | 151.7 | **136.9** | 53 | **8** | **8** | 12 |
| PathM (S) | **18** | 6 | 6 | 17 | **42.3** | 237.8 | 211.0 | 92.9 | 684 | **124** | 166 | 245 |
| PlWat (S) | **6** | 4 | **6** | **6** | 215.3 | 255.2 | 217.4 | **214.8** | 301 | **151** | 196 | 291 |
| Rvr (S) | **15** | 6 | **15** | **15** | 101.4 | 207.5 | **95.6** | 98.0 | 65 | **16** | **16** | 18 |
| Sail (S) | **20** | 8 | 13 | **20** | **3.6** | 203.1 | 149.3 | 29.9 | 932 | **435** | 760 | 842 |
| Sail (L) | **19** | 8 | **19** | **19** | **16.3** | 221.7 | 44.5 | 22.4 | 355 | **59** | 194 | 208 |
| Stlrs (S) | **8** | 1 | – | 4 | **210.5** | 295.5 | – | 247.0 | 7.0k | **26** | – | 135 |
| Sgr (S) | **20** | 10 | 19 | **20** | 10.3 | 170.5 | **10.2** | 10.6 | 47 | **20** | 26 | 27 |
| Tpp (L) | **2** | **2** | **2** | **2** | 270.2 | 272.3 | 270.2 | **270.1** | 13 | **8** | 10 | 12 |
| Zeno (S) | **11** | 10 | 10 | **11** | **136.4** | 147.7 | 143.7 | 136.5 | 22 | **15** | 18 | 18 |
| Line (L) | **20** | 19 | **20** | **20** | **1.2** | 6.3 | 1.7 | 3.0 | 399 | **329** | **329** | 337 |
| *Best* | **281** | **161** | **235** | **275** | **180** | **9** | **17** | **83** | **24** | **161** | **124** | **69** |

## *5.2. Quality of the computed plan*

As discussed in Section 3.4, it is indeed possible for the returned plan to contain redundant actions. To assess the extent of this issue, we will compare Patty$_E$ with:

1. Patty$_M$, i.e., Patty$_E$ where the solver is instructed to return a solution that minimizes the length of the returned plan, i.e., the quantity $\sum_{i=1}^{k} a_i$,

2. $\text{PATTY}_{\text{I}}$, i.e., $\text{PATTY}_{\text{E}}$ where the first computed model $\mu$ is used to find an irredundant plan, i.e., a plan corresponding to a model that minimizes $\sum_{i=1}^{k} a_i$ while also satisfying $\wedge_{i=1}^{k} a_i \leq \mu(a_i)$,
3. $\text{PATTY}_{\text{C}}$, i.e., $\text{PATTY}_{\text{E}}$ where, from the first computed plan, redundant actions are removed employing the Action Elimination (AE) Algorithm, quadratic in the length of the plan, presented in Algorithm 1 of [24] – originally introduced by [26] and later rediscovered and formulated by [27] – cast for numeric planning.

$\text{PATTY}_{\text{M}}$ and $\text{PATTY}_{\text{I}}$ are thus guaranteed to return a $\prec$-optimal and a $\pi$-optimal plan, respectively, as discussed in Section 3.4. The results are in Table 2.

The table shows the number of solved problems and average time, and the average length of the returned plan (Sub-table Plan length), the latter computed considering only the problems solved by all the considered planners. Notice that when a plan is returned, $\text{PATTY}_{\text{E}}$, $\text{PATTY}_{\text{M}}$, $\text{PATTY}_{\text{I}}$ and $\text{PATTY}_{\text{C}}$ use the same pattern.

As it can be seen from the table, for every domain

1. the average length of the computed plan is the smallest for $\text{PATTY}_{\text{M}}$ and the highest for $\text{PATTY}_{\text{E}}$,
2. vice versa, the average number of solved problems is the highest for $\text{PATTY}_{\text{E}}$ and the lowest for $\text{PATTY}_{\text{M}}$ for all domains except Exp (S), where $\text{PATTY}_{\text{M}}$ is able to solve one more problem than the others, (which is also $\prec$-optimal),
3. the coverage of $\text{PATTY}_{\text{C}}$ is almost identical to the coverage of $\text{PATTY}_{\text{E}}$, except for the STLRS (S) domain, where the plan firstly returned by $\text{PATTY}_{\text{E}}$ is very long (7k actions) causing $\text{PATTY}_{\text{C}}$ to reach the 5 min timeout, due to the AE algorithm, quadratic in the length of the plan,
4. the difference between the time needed by $\text{PATTY}_{\text{E}}$ vs $\text{PATTY}_{\text{I}}$ varies between being (almost) null (for DRN (S)) and very significant (e.g., for CNT (L)). This is similar for $\text{PATTY}_{\text{E}}$ and $\text{PATTY}_{\text{C}}$, where the difference in time is proportional to the plan length. Occasionally, a shorter time than $\text{PATTY}_{\text{E}}$ for $\text{PATTY}_{\text{I}}$, $\text{PATTY}_{\text{M}}$, $\text{PATTY}_{\text{C}}$ is reported, which can be attributed to the varying performance of the SMT solver, even when considering the same problem.

Depending on the domain, the reduction in the length of the returned plan varies between being marginal (see, e.g., DEL (S)) and very significant (see, e.g., SAIL (L), STLRS (S)).

We recall that both $\text{PATTY}_{\text{M}}$ and $\text{PATTY}_{\text{I}}$ are guaranteed to return an irredundant plan. This necessitates proving the non-existence of any other plan which is a subsequence of the given one. $\text{PATTY}_{\text{C}}$ instead, is not guaranteed to return an irredundant plan. However, in some domains – like RVR (S) or MPRIME (S) –, the plans produced are still of good quality, while maintaining the coverage and time.

### *5.3. Comparative analysis to other SOTA symbolic planners*

We compared our planner $\text{PATTY}_{\text{E}}$ to the three planning as satisfiability planners SPRINGROLL (based on the rolled-up $\Pi^R$ encoding [6]), the version $R^2\exists$ of PATTY computing the planning as satisfiability $R^2\exists$ $\prec$-encoding $\Pi^{R^2\exists,\prec}$, and OMTPLAN, based on the $\Pi^S$ standard encoding, optimized to prune useless variables [19]. OMTPLAN participated and ranked second in the 2023 IPC. As an ablation study, we also implemented a version of $R^2\exists$ inside PATTY where action variables are represented as non-negative integers, and thus can be rolled: we dubbed this technique *Rolled Relaxed-Relaxed* $\exists$ *step* ($R^3\exists$). The results are in Table 3. In the table, besides the number of solved problems and average time, we show: the number of calls to the SMT solver, the number of variables (Sub-table Variables) and clauses (Sub-table Clauses) of the encoding when a solution is found. The last three numbers have been computed considering the problems solved by all the symbolic planners able to solve at least one problem in the domain.

Considering the table, three main observations are in order. First, $\text{PATTY}_{\text{E}}$ always finds a solution with a number of calls to the SMT solver which is never higher than the ones needed by the other considered symbolic planners (as theoretically established by Theorem 11). Consequently, $\text{PATTY}_{\text{E}}$ produces formulas with (far) fewer variables and clauses than $R^2\exists$, $R^3\exists$, OMTPLAN and SPRINGROLL, when the plan is found. The lower number of variables and clauses of $\text{PATTY}_{\text{E}}$ is also due to the particular encoding in which no variables representing the intermediate states are used.

Second, by looking at the performances of $\text{PATTY}_{\text{E}}$ vs $R^3\exists$, we show that our approach is not simply an agglomerate of the $R^2\exists$ [8] and of the rolling [6] approaches. In fact, our approach indeed incorporates the two ideas but actually improves on them. In the pattern approach, for each action $a_i$ in the pattern $\prec = a_1; \dots; a_k$ the expression $\sigma_i(v)$ denotes the value of the variable $v$ as a function of the state and action variables in $\mathcal{X} \cup \{a_1, a_2, \dots, a_i\}$ without adding extra variables. In the $R^2\exists$ approach, with the same order $\prec$, instead, as shown in Section 4.3, an additional variable $v^{a_i}$ would have been added to the encoding. Avoiding using these additional variables is very beneficial for $\text{PATTY}_{\text{E}}$, that uses far fewer variables than $R^2\exists$, $R^3\exists$, and SPRINGROLL, as shown by the subtables concerning the variables and the clauses and reflected in the smaller coverage and the increased planning time.

Third, considering the sub-tables of solved problems and average time, $\text{PATTY}_{\text{E}}$ outperforms all the other planners in almost every domain: $\text{PATTY}_{\text{E}}$ always solves more problems and in only two domains it exhibits a longer average solving time. Interestingly,

1. on some domains action rolling is important (as witnessed by the performance of SPRINGROLL on BLGRP (S), FARM (S) and SAIL (S)),
2. on some other domains it is important to allow for sequences of mutually interfering actions in a single step (as witnessed by $R^2\exists$ planner on FARM (L), TPP (L), ZENO (S)),
3. in SGR (S), OMTPLAN has remarkably good performance compared to SPRINGROLL and $R^2\exists$ planner, likely thanks to its variable pruning techniques, playing a role also in ZENO (S) (by comparison to SPRINGROLL).

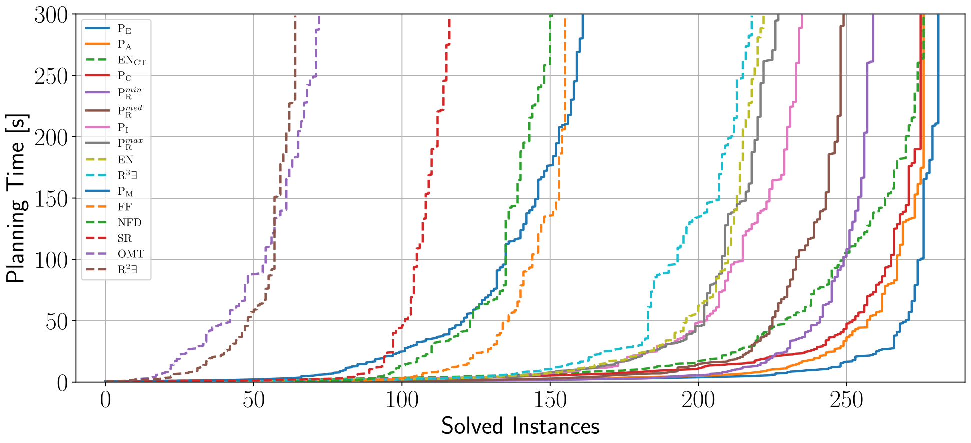

**Fig. 2.** Number of problems solved ($x$-axis) in a given time ($y$-axis), by all the presented systems. $\mathrm{P_E}$ stands for PATTY$_\mathrm{E}$, and similarly for $\mathrm{P_A}/\mathrm{P_R^{min}}/\mathrm{P}_\mathrm{R}^{med}/\mathrm{P_R^{max}}/\mathrm{P_I}/\mathrm{P_M}/\mathrm{P_C}$. The different versions of PATTY are represented with solid lines. EN$_\mathrm{CT}$/EN/METRICFF/NFD/SR/OMT/$R^2\exists$/$R^3\exists$ stand for the ENHSP$_\mathrm{CT}$/ENHSP/METRICFF/NFD/SPRINGROLL/OMTPLAN/$R^2\exists$/$R^3\exists$ planners and are represented with dashed lines. In the legend, the planners are listed in reverse order of when their curve intersects the timeout line and are in lexicographic order when they intersect at the same point.

Finally, considering the plan length, PATTY$_\mathrm{E}$ has the shortest average plan length in five domains, three of which thanks to the fact that it is the only system able to solve some of its problems. On the other domains, the difference in the plan length can be marginal (as for ZENO (S)) or very significant (e.g., for FARM (L) and SAIL (S)).

As discussed in Section 3.4, if rolling is not allowed (as it is for $R^2\exists$ and OMTPLAN), the set of models of the respective encoding is guaranteed to be finite, while this is not necessarily the case for PATTY$_\mathrm{E}$ and SPRINGROLL. When an encoding has infinitely many models, PATTY$_\mathrm{E}$ and SPRINGROLL may return an arbitrarily long plan, as each model corresponds to a different plan. Further, as the example in Section 3.4 shows, there can be cases in which our encoding can have infinitely many models while SPRINGROLL may not, given that the latter does not allow executing mutually conflicting actions in a single step. Indeed, by disabling rolling and ensuring that exactly one action is executed at each step, all symbolic planners are guaranteed to return an optimal plan with the minimum possible number of actions (though this might be impractical since it will require making a number of SMT calls equal to the plan length).

## *5.4. Comparative analysis to other SOTA search-based planners*

We compared our planner PATTY$_\mathrm{E}$ with the four search-based planners ENHSP [4], ENHSP$_\mathrm{CT}$ [28], METRICFF [29] and NUMERICFASTDOWNWARD (NFD) [30]. NFD competed in the 2023 IPC, ranking first. The results are reported in Table 4. Here, besides the number of solved problems and time, we also report the average length of the computed plans, as usual computed considering the problems solved by all the planners able to solve at least one problem in the domain.

Considering the sub-tables with the performance data, PATTY$_\mathrm{E}$ solves the most problems in 13 domains, followed by ENHSP$_\mathrm{CT}$, ENHSP, METRICFF, NFD in 10, 7, 4, 2, 0 domains, respectively. Overall, PATTY$_\mathrm{E}$, ENHSP$_\mathrm{CT}$, ENHSP, NFD, METRICFF solve 281, 276, 222, 151 and 155 problems, respectively. As pointed out in [28], the problems in which PATTY$_\mathrm{E}$ performs better are those with infinitely large state spaces (in which search-based methods are likely to get lost) and in which its pattern computation allows finding relatively short rolled up plans. If the domain has mostly finite domain variables and requires long plans with multiple non-consecutive executions of the same actions, then PATTY$_\mathrm{E}$ concatenates the initially computed pattern $\prec_I$ for $n > 1$ times. As $n$ increases, the number of decision variables in the encoding increases. Further, $\prec_I$ is likely to provide limited guidance when concatenated for $n > 1$ times. By contrast, ENHSP$_\mathrm{CT}$ exploitation of multi-queue search and also a portfolio of different heuristics allow it to adapt its search mechanism to the specific domain and also the state currently at hand. The differences in the average plan length between PATTY$_\mathrm{E}$ and the other considered planners is even more evident here: for average plan length, PATTY$_\mathrm{E}$ generates shorter plans than the other systems in 3 domains, while ENHSP$_\mathrm{CT}$, ENHSP, NFD, and METRICFF lead in 10, 4, 9 and 7 domains, respectively.

The fact that depending on the domain there can be significant differences between PATTY$_\mathrm{E}$ and the search-based planners, highlights the complementary nature of these planners. For this reason, a very different picture can be obtained by considering a different set of benchmarks.

## *5.5. Overall comparative analysis*

The cactus plot in Fig. 2 summarizes the performance of all the systems we presented. The graph plots how many problems can be solved in a given time. As it can be seen, all the different versions of PATTY have better performance than the other symbolic planning

**Table 3**
Comparative analysis between Patty$_E$ and the symbolic planners $R^2\exists$, $R^3\exists$, OMTPlan and SpringRoll.

| | Solved (out of 20) | | | | | Time (s) | | | | | SMT calls | | | | | Plan length | | | | | Variables | | | | | Clauses | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | P$_E$ | R$^2\exists$ | R$^3\exists$ | OMT | SR | P$_E$ | R$^2\exists$ | R$^3\exists$ | OMT | SR | P$_E$ | R$^2\exists$ | R$^3\exists$ | OMT | SR | P$_E$ | R$^2\exists$ | R$^3\exists$ | OMT | SR | P$_E$ | R$^2\exists$ | R$^3\exists$ | OMT | SR | P$_E$ | R$^2\exists$ | R$^3\exists$ | OMT | SR |
| BlGrp (S) | **20** | 17 | **20** | 2 | **20** | 1.8 | 83.4 | 8.3 | 270.2 | **1.6** | **1.0** | 6.0 | **1.0** | 8.5 | **1.0** | 124 | **22** | 172 | 60 | 74 | **40** | 1.4k | 250 | 265 | **40** | **101** | 1.7k | 331 | 776 | 122 |
| Cnt (S) | **20** | 11 | **20** | 18 | **20** | **0.9** | 169.2 | 1.4 | 92.7 | **0.9** | **1.0** | 13.3 | **1.0** | 13.3 | **1.0** | 164 | 125 | 258 | 163 | **112** | **45** | 7.6k | 391 | 619 | **45** | **114** | 8.9k | 482 | 1.8k | 137 |
| Cnt (L) | **20** | 4 | **20** | 3 | 6 | **1.1** | 240.3 | 2.5 | 255.3 | 227.6 | **1.0** | 1.7 | **1.0** | 5.3 | 2.7 | 10 | 8 | 1.4k | **5** | 7 | **26** | 208 | 122 | 122 | 60 | **58** | 276 | 166 | 1.5k | 297 |
| Del (S) | **5** | 1 | 3 | 1 | – | **226.4** | 285.7 | 272.2 | 295.6 | – | **1.0** | 2.0 | **1.0** | 10.0 | – | **10** | **10** | 11 | **10** | – | **250** | 15.9k | 6.9k | 1.9k | – | **662** | 16.7k | 7.4k | 246.0k | – |
| Drn (S) | **3** | **3** | **3** | **3** | **3** | **255.3** | 259.0 | 255.5 | 256.3 | 256.3 | **5.7** | 8.3 | **5.7** | 12.3 | 9.7 | 25 | 14 | 26 | **12** | 15 | **142** | 1.3k | 938 | 299 | 232 | **344** | 1.6k | 1.2k | 5.1k | 2.4k |
| Exp (S) | **2** | – | **2** | **2** | – | **270.2** | – | 271.5 | 276.2 | – | **3.0** | – | **3.0** | 16.0 | – | 38 | - | 37 | **28** | – | **254** | – | 6.7k | 1.1k | – | **612** | – | 7.4k | 35.2k | – |
| Farm (S) | **20** | – | **20** | – | **20** | 2.4 | – | 19.5 | – | **1.3** | **1.0** | – | **1.0** | – | 2.2 | 786 | – | 894 | – | **334** | **63** | – | 240 | - | 134 | **120** | – | 323 | - | 1.3k |
| Farm (L) | **20** | 2 | **20** | 1 | – | **2.7** | 270.2 | 25.5 | 286.2 | – | **1.0** | 8.0 | **1.0** | 12.0 | – | 145 | **14** | 99 | 19 | – | **19** | 338 | 37 | 112 | – | **32** | 460 | 55 | 545 | – |
| HPwr (S) | **20** | – | 10 | – | – | **9.4** | – | 234.3 | – | – | **1.0** | – | **1.0** | – | – | **97** | – | 116 | – | – | **444** | – | 7.9k | – | – | **788** | – | 8.3k | – | – |
| Mrkt (L) | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| MPrime (S) | **12** | 5 | 5 | 6 | 10 | **137.7** | 233.2 | 238.9 | 229.6 | 174.2 | **1.2** | 2.2 | **1.2** | 4.2 | 5.2 | 54 | **7** | 63 | 8 | 34 | **364** | 36.3k | 15.3k | 1.1k | 1.4k | **918** | 37.4k | 15.9k | 86.0k | 59.5k |
| PathM (S) | **18** | 1 | 5 | 3 | 1 | **42.3** | 286.1 | 243.6 | 262.4 | 286.0 | **1.0** | 6.0 | **1.0** | 9.0 | 3.0 | 57 | **12** | 1.8k | 28 | 57 | **186** | 19.2k | 2.8k | 986 | 416 | **318** | 20.0k | 3.0k | 5.2k | 1.4k |
| PlWat (S) | **6** | - | 5 | – | – | **215.3** | – | 242.0 | – | – | **7.6** | – | **7.6** | – | – | **318** | – | 385 | – | – | **363** | – | 4.0k | – | – | **997** | – | 4.9k | – | – |
| Rvr (S) | **15** | 8 | 6 | 7 | 11 | **101.4** | 202.6 | 218.9 | 232.6 | 181.8 | **1.3** | 2.0 | **1.3** | 7.8 | 7.8 | 58 | **14** | 55 | **14** | 17 | **339** | 21.2k | 12.1k | 1.2k | 1.6k | **697** | 21.7k | 12.6k | 100.6k | 53.1k |
| Sail (S) | **20** | – | **20** | – | **20** | **3.6** | – | 26.1 | – | 5.5 | **3.3** | – | **3.3** | – | 7.3 | 6.1k | – | 17.6k | – | **1.2k** | **135** | – | 1.3k | – | 286 | **266** | – | 1.4k | – | 2.1k |
| Sail (L) | 19 | – | **20** | 1 | – | 16.3 | – | **1.6** | 285.8 | – | **1.0** | – | **1.0** | 13.0 | – | 161 | – | 4.1k | **59** | – | **84** | – | 898 | 874 | – | **200** | – | 1.1k | 5.8k | – |
| Stlrs (S) | **8** | – | – | – | – | **210.5** | – | – | – | – | **1.0** | – | – | – | – | **38.3k** | – | – | – | – | **1.4k** | – | – | – | – | **2.9k** | – | – | – | – |
| Sgr (S) | **20** | 1 | 9 | 18 | – | **10.3** | 288.2 | 215.3 | 113.7 | – | **2.0** | **2.0** | **2.0** | 5.0 | – | 32 | 29 | 43 | **18** | – | **814** | 55.5k | 40.9k | 1.7k | – | **2.0k** | 56.9k | 42.4k | 92.4k | – |
| Tpp (L) | **2** | **2** | **2** | – | – | **270.2** | 271.9 | 271.6 | – | – | **2.5** | **2.5** | **2.5** | – | – | 13 | **10** | **10** | – | – | **237** | 2.6k | 3.0k | – | – | **604** | 3.0k | 3.5k | – | – |
| Zeno (S) | **11** | 9 | 8 | 7 | – | **136.4** | 174.6 | 195.1 | 209.6 | – | **1.6** | **1.6** | **1.6** | 5.3 | – | 17 | 16 | 16 | **13** | - | **241** | 7.0k | 7.9k | 931 | – | **700** | 7.4k | 8.4k | 74.8k | – |
| Line (L) | **20** | – | **20** | – | 5 | **1.2** | – | 2.0 | – | 262.4 | **3.0** | – | **3.0** | – | 26.0 | **110** | – | 112 | – | 158 | **161** | – | 1.4k | – | 1.1k | **381** | – | 1.6k | – | 4.2k |
| *Best* | **281** | **64** | **218** | **72** | **116** | **224** | **0** | **14** | **5** | **39** | **281** | **17** | **218** | **0** | **43** | **130** | **42** | **23** | **36** | **63** | **281** | **0** | **1** | **0** | **43** | **281** | **0** | **1** | **0** | **0** |

**Table 4**
Comparative analysis between PATTY$_E$ and the search-based planners ENHSP, ENHSP$_{CT}$, METRICFF and NUMERICFASTDOWNWARD(NFD). A "*" indicates that no problem was solved by all the planners capable of solving at least one problem in the domain.

| | Solved (out of 20) | | | | | Time (s) | | | | | Plan Length | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | P$_E$ | EN$_{CT}$ | EN | NFD | FF | P$_E$ | EN$_{CT}$ | EN | NFD | FF | P$_E$ | EN$_{CT}$ | EN | NFD | FF |
| BlGrp (S) | **20** | 14 | 16 | – | 2 | **1.8** | 117.2 | 81.5 | – | 270.2 | 124 | **22** | **22** | – | 24 |
| Cnt (S) | **20** | 10 | 12 | 11 | 15 | **0.9** | 163.8 | 133.8 | 149.8 | 95.7 | 128 | 85 | 85 | **84** | **84** |
| Cnt (L) | **20** | 12 | 10 | 6 | 8 | **1.1** | 142.3 | 170.9 | 214.0 | 180.0 | 30 | 16 | 29 | 16 | **13** |
| Del (S) | 5 | 14 | 13 | 9 | **18** | 226.4 | 117.1 | 121.7 | 165.2 | **41.2** | **25** | 28 | 31 | 35 | **25** |
| Drn (S) | 3 | **18** | 16 | 16 | 2 | 255.3 | **55.4** | 62.9 | 66.0 | 268.4 | 16 | **7** | 8 | **7** | **7** |
| Exp (S) | 2 | **6** | **6** | 3 | – | 270.2 | 224.0 | **212.3** | 253.7 | – | **36** | 48 | 72 | 54 | – |
| Farm (S) | **20** | **20** | **20** | 15 | 9 | 2.4 | 1.8 | **0.9** | 85.3 | 188.1 | 701 | **292** | **292** | **292** | 341 |
| Farm (L) | **20** | **20** | 18 | 11 | 15 | 2.7 | **2.5** | 48.6 | 151.2 | 80.5 | 864 | 254 | 34 | **21** | 34 |
| HPwr (S) | **20** | **20** | 2 | 1 | 1 | 9.4 | **4.6** | 270.3 | 285.1 | 285.0 | 64 | **16** | 20 | 35 | **16** |
| Mrkt (L) | - | **20** | 4 | – | – | – | **35.0** | 259.3 | – | – | – | **424** | 594 | – | – |
| MPrime (S) | 12 | **17** | **17** | 14 | **17** | 137.7 | 74.6 | 68.1 | 127.2 | **45.1** | 63 | **7** | 9 | **7** | 8 |
| PathM (S) | **18** | 3 | 2 | 1 | 10 | **42.3** | 262.8 | 272.2 | 284.2 | 154.9 | 57 | **12** | 18 | **12** | 14 |
| PlWat (S) | 6 | **20** | 16 | 14 | 3 | 215.3 | **41.1** | 101.3 | 167.2 | 268.3 | 285 | **235** | 429 | 393 | 455 |
| Rvr (S) | **15** | 12 | 8 | 4 | 10 | **101.4** | 143.7 | 197.4 | 240.8 | 133.3 | 34 | 13 | 33 | **9** | **9** |
| Sail (S) | **20** | **20** | **20** | 10 | 1 | 3.6 | 5.0 | **2.0** | 150.3 | 285.0 | **174** | **174** | **174** | **174** | 179 |
| Sail (L) | **19** | 2 | 2 | 15 | 8 | **16.3** | 270.8 | 270.6 | 96.8 | 182.8 | * | * | * | * | * |
| Stlrs (S) | **8** | 2 | 1 | - | 4 | **210.5** | 279.0 | 288.6 | - | 243.8 | * | * | * | - | * |
| Sgr (S) | **20** | 11 | 8 | 4 | 13 | **10.3** | 144.5 | 182.5 | 245.7 | 122.5 | * | * | * | * | * |
| Tpp (L) | 2 | **7** | 3 | 2 | 2 | 270.2 | **212.3** | 255.2 | 270.0 | 266.7 | 13 | 11 | 11 | **5** | 9 |
| Zeno (S) | 11 | 17 | **19** | 9 | 11 | 136.4 | 89.5 | **28.1** | 172.5 | 135.0 | 20 | 15 | **14** | 21 | **14** |
| Line (L) | **20** | 11 | 9 | 6 | 6 | **1.2** | 149.5 | 175.4 | 235.0 | 211.6 | 211 | **171** | 276 | 234 | 187 |
| *Best* | **281** | **276** | **222** | **151** | **155** | **136** | **62** | **41** | **37** | **94** | **65** | **162** | **96** | **85** | **92** |

systems. The better results achieved even by PATTY$_M$/PATTY$_I$–guaranteed to return irredundant plans– and PATTY$_R^{max}$–representing PATTY's lowest performance across 5 runs per problem with a randomly generated pattern– demonstrate the robustness of our approach. All our systems except for PATTY$_M$ have also better performance than all the search-based planners, except for ENHSP$_{CT}$, which solves roughly the same number of problems as PATTY$_E$ and PATTY$_A$.

Overall, of the 420 problems we considered, PATTY$_E$ successfully solved 281, while ENHSP$_{CT}$, ENHSP and SPRINGROLL solved 276, 221 and 116, respectively. ENHSP$_{CT}$ and SPRINGROLL are the top performers among the search-based and the other symbolic planners, respectively.

## 6. Conclusions and future work

We proposed Symbolic Pattern Planning (SPP), a novel approach for solving automated planning problems in deterministic domains. A pattern is a sequence of actions, each of which can be executed for 0 or more times. The core idea of SPP is to encode as a formula the state that results from executing the actions in the pattern zero or more times, and then impose the conditions of the initial and goal states. Assuming the correctness of the encoding, by iteratively extending the pattern by adding a complete sequence of actions, we obtain a correct and complete procedure for planning.

On the theoretical side, we proved that when SPP is cast in the planning as satisfiability framework, our encoding generalizes both the $R^2\exists$ encoding by allowing for action rolling (as in the $R$ encoding), and the rolled-up $R$ encoding by allowing for actions with interfering preconditions and effects (as in the $R^2\exists$ encoding). This generalization leads the pattern encoding to often find plans with a lower number of calls to the SMT solver.

Experimentally, we considered the basic SPP procedure in which an initial elementary and complete pattern is computed at the beginning and then iteratively used to extend the current pattern until a valid plan is found. We considered the benchmarks in the 2023 IPC, numeric track and showed that the resulting planner PATTY performs better than all the currently available symbolic planners, and comparatively well also when considering search-based planners.

This work can be extended along several lines. First, as outlined in the introduction, it is possible to apply the SPP idea to any deterministic planning problem for which it is possible to define a formula encoding the state resulting from the execution of each action in the pattern for 0 or more times. One possibility is thus to consider more expressive languages for planning domain specification, such as PDDL level 3 or level 4 [1]. Yet another possibility is to specialize the encoding that we propose to deal with classical planning problems, e.g., formalized in PDDL level 1. Such a specialization is needed to compete with the highly tuned current classical planners. Finally, it is clear that the simple SPP procedure that we defined and experimentally tested in this paper, can be improved in many ways. The possibility to define and use a different pattern at each iteration seems the first natural step. Here, the problem is to define when and how to recompute the pattern, given the previously defined and used patterns. We are currently working on all these three lines of research.

## CRediT authorship contribution statement

**Matteo Cardellini:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Formal analysis, Data curation, Conceptualization; **Enrico Giunchiglia:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization; **Marco Maratea:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology.

## Data availability

The link to github is available in the paper.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:
Enrico Giunchiglia reports financial support was provided by European Union. Marco Maratea reports financial support was provided by European Union. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

Enrico Giunchiglia acknowledges the financial support from PNRR MUR Project PE0000013 FAIR "Future Artificial Intelligence Research", funded by the European Union - NextGenerationEU, CUP J33C24000420007, and from Project PE00000014 "SEcurity and RIghts in the CyberSpace", CUP D33C22001300002. Marco Maratea has been partially supported by the MUR under PNRR project FAIR "Future AI Research", CUP H23C22000860006.

## References

[1] M. Fox, D. Long, PDDL2.1: an extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (2003) 61-124. https://doi.org/10.1613/jair.1129
[2] M. Cardellini, E. Giunchiglia, M. Maratea, Symbolic numeric planning with patterns, in: M.J. Wooldridge, J.G. Dy, S. Natarajan (Eds.), Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada, AAAI Press, 2024, pp. 20070–20077. https://doi.org/10.1609/AAAI.V38I18.29985.
[3] C. Barrett, C. Tinelli, Satisfiability Modulo Theories, Handbook of Model Checking (2018) 305–343.
[4] E. Scala, P. Haslum, S. Thiebaux, M. Ramirez, Interval-based relaxation for general numeric planning, in: Proceedings of the Twenty-second European Conference on Artificial Intelligence, 2016, p. 655-663. https://doi.org/10.3233/978-1-61499-672-9-655.
[5] H.A. Kautz, B. Selman, Planning as satisfiability, in: B. Neumann (Ed.), 10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings, John Wiley and Sons, 1992, pp. 359–363.
[6] E. Scala, M. Ramirez, P. Haslum, S. Thiebaux, Numeric planning with disjunctive global constraints via SMT, Proceedings of the International Conference on Automated Planning and Scheduling 26 (2016) 276-284. https://doi.org/10.1609/icaps.v26i1.13766
[7] T. Balyo, Relaxing the relaxed exist-step parallel planning semantics, in: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, IEEE, Herndon, VA, USA, 2013, pp. 865–871. http://ieeexplore.ieee.org/document/6735342/. https://doi.org/10.1109/ICTAI.2013.131
[8] M. Bofill, J. Espasa, M. Villaret, The RANTANPLAN planner: system description, Knowl. Eng. Rev. 31 (5) (2016) 452-464. https://doi.org/10.1017/S0269888916000229
[9] M. Cardellini, E. Giunchiglia, Temporal numeric planning with patterns, Proc. AAAI Conf. Artif. Intell. 39 (25) (2025) 26481–26489. https://doi.org/10.1609/aaai.v39i25.34848.
[10] V. Vidal, The YAHSP planning system: forward heuristic search with lookahead plans analysis, in: International Planning Competition, 2004, p. 56.
[11] D. Alarnaouti, F. Percassi, M. Vallati, An extensive empirical analysis of macro-actions for numeric planning, in: International Conference of the Italian Association for Artificial Intelligence, Springer, 2024, pp. 23–36.
[12] L. Bonassi, A.E. Gerevini, E. Scala, Planning with qualitative action-trajectory constraints in PDDL, in: L. De Raedt (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, ijcai.org, 2022, pp. 4606–4613. https://doi.org/10.24963/IJCAI.2022/639.
[13] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, J. Seipp, The 2023 International Planning Competition, AI Mag. n/a (n/a) (n.d.). https://onlinelibrary.wiley.com/doi/pdf/10.1002/aaai.12169https://doi.org/10.1002/aaai.12169
[14] F. Leofante, E. Giunchiglia, E. Ábráham, A. Tacchella, Optimal planning modulo theories, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan, 2020, p. 4128-4134. https://www.ijcai.org/proceedings/2020/571. https://doi.org/10.24963/ijcai.2020/571.
[15] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Pearson, 3 edition, 2010.
[16] C. Barrett, P. Fontaine, C. Tinelli, The satisfiability modulo theories library (SMT-LIB), 2016, (www.SMT-LIB.org). Accessed: 2024-01-06.
[17] J. Rintanen, K. Heljanko, I. Niemelä, Planning as satisfiability: parallel plans and algorithms for plan search, Artif. Intell. 170 (12-13) (2006) 1031–1080. https://doi.org/10.1016/j.artint.2006.08.002.
[18] P. Bercher, P. Haslum, C. Muise, A survey on plan optimization, in: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024, ijcai.org, 2024, pp. 7941–7950. https://www.ijcai.org/proceedings/2024/879.
[19] F. Leofante, OMTPlan: a tool for optimal planning modulo theories, J. Satisf. Boolean Model. Comput. 14 (1) (2023) 17–23. https://doi.org/10.3233/SAT-220001
[20] L. De Moura, N. Bjørner, Z3: an efficient SMT solver, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.
[21] E. Giunchiglia, M. Maratea, Planning as satisfiability with preferences, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, AAAI Press, 2007, pp. 987–992. http://www.aaai.org/Library/AAAI/2007/aaai07-157.php.

[22] A. Cimatti, A. Griggio, B.J. Schaafsma, R. Sebastiani, The MathSAT5 SMT solver, in: N. Piterman, S.A. Smolka (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings, 7795 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 93–107. https://doi.org/10.1007/978-3-642-36742-7_7.

[23] T. Balyo, L. Chrpa, A. Kilani, On different strategies for eliminating redundant actions from plans, in: Proceedings of the International Symposium on Combinatorial Search, 5, 2014, pp. 10–18.

[24] L. Chrpa, On speeding up methods for identifying redundant actions in plans, in: Proceedings of the International Conference on Automated Planning and Scheduling, 32, 2022, pp. 252–260.

[25] M. Bofill, J. Espasa, M. Villaret, Relaxed exists-step plans in planning as SMT, in: C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 563–570. https://doi.org/10.24963/ijcai.2017/79.

[26] E. Fink, Q. Yang, Formalizing plan justifications, in: Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence, 1992, pp. 1–9.

[27] H. Nakhost, M. Müller, Action elimination and plan neighborhood graph search: two algorithms for plan improvement, in: Proceedings of the International Conference on Automated Planning and Scheduling, 20, 2010, pp. 121–128.

[28] D.Z. Chen, S. Thiébaux, Novelty heuristics, multi-queue search, and portfolios for numeric planning, in: A. Felner, J. Li (Eds.), Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024, AAAI Press, 2024, pp. 203–207. https://doi.org/10.1609/SOCS.V17I1.31559.

[29] J. Hoffmann, The metric-FF planning system: translating “ignoring delete Lists” to numeric state variables, J. Artif. Intell. Res. 20 (2003) 291–341.

[30] R. Kuroiwa, A. Shleyfman, J.C. Beck, LM-Cut heuristics for optimal linear numeric planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, 32, 2022, pp. 203–212.