Temporal Numeric Planning with Patterns

Matteo Cardellini, Enrico Giunchiglia

DIBRIS, Università degli Studi di Genova, Italy
matteo.cardellini@edu.unige.it, enrico.giunchiglia@unige.it

Abstract

We consider temporal numeric planning problems Π expressed in PDDL 2.1 level 3, and show how to produce SMT formulas (i) whose models correspond to valid plans of Π, and (ii) that extend the recently proposed planning with patterns approach from the numeric to the temporal case. We prove the correctness and completeness of the approach and show that it performs very well on 10 domains with required concurrency.

Introduction

We consider temporal numeric planning problems expressed in PDDL 2.1 level 3 (Fox and Long 2003). Differently from the classical case, where plans are sequences of instantaneous actions and variables are Boolean, in these problems 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. These two extensions make the problem of finding a valid plan much more difficult –even undecidable in the general case (Helmert 2002; Gigante et al. 2022)– and extending state-of-the-art solving techniques from the classical/numeric to the temporal numeric setting is far from easy.

In this paper, we extend the recently proposed Symbolic Pattern Planning (SPP) approach (Cardellini, Giunchiglia, and Maratea 2024) to handle temporal numeric problems. Specifically, given one such problem Π and a bound $n \in \mathbb{N}^{\geq 0}$, we show how to produce a Satisfiability Modulo Theory (SMT) formula (Barrett et al. 2021) (i) whose models correspond to valid plans of Π (correctness), (ii) which is ensured to be satisfiable for some value of the bound n when Π has a valid plan (completeness), and (iii) which is equivalent to the pattern encoding proposed by Cardellini, Giunchiglia, and Maratea when the problem is numeric, i.e., when all the actions are instantaneous. These results significantly advance the state-of-the-art, as all symbolic temporal numeric planners are based on the standard encoding with effect and explanatory frame axioms. Given this, we expect to obtain also in the temporal setting the substantial improvements achieved in the numeric case, where it was shown (i) that the pattern encoding *dominates* (i.e., is able to produce valid plans with a bound n possibly lower and never higher than) both the relaxed-relaxed $R^2\exists$ encoding (Balyo 2013; Bofill, Espasa, and Villaret 2017), and the action rolling R encoding (Scala et al. 2016b); (ii) that both the $R^2\exists$ and R encodings dominate the standard one, and (iii) that theoretical dominance leads to improved experimental performance, as shown by the analysis in (Cardellini, Giunchiglia, and Maratea 2024) on the numeric benchmarks of the 2023 International Planning Competition.

To test the effectiveness of our approach, we compare our planner with all publicly available temporal planners (both symbolic and based on search) on 10 temporal domains with required concurrency (Cushing et al. 2007). The results highlight the strong performances of our planner, which achieved the highest coverage (i.e., number of solved problems) in 9 out of 10 domains, while the second-best planner had the highest coverage in 4 domains. Additionally, compared to the other symbolic planners, our system is able to find a valid plan with a lower bound on all the problems.

The main contributions of this paper are thus:

1. We extend the SPP approach to handle temporal numeric problems specified in PDDL 2.1.
2. We prove the correctness and completeness of our SMT encoding.
3. We conduct an extensive comparative analysis with all available temporal numeric planners, both search-based and symbolic-based, and show that our approach achieves the highest coverage in 9 out of 10 domains.

After the preliminary definitions, we present the basic ideas underlying current standard encodings in SMT, followed by the presentation of our approach, the experimental comparative analysis and the conclusions. A running example is used to illustrate the features of our encoding.

Preliminaries

In PDDL2.1 (Fox and Long 2003) a *temporal numeric planning problem* is a tuple $\Pi = \langle V_B, V_N, A, I, G\rangle$, where

1. V_B and V_N are finite sets of *Boolean* and *numeric variables*, ranging over $\{\top, \bot\}$ and $\mathbb{Q}$ respectively,
2. I is a selected *initial state*, and a *state* is a function mapping each variable to an element in its domain,

3. G is a finite set of conditions, called *goals*. A *condition* is either $v = \top$ or $v = \bot$ or $\psi \trianglerighteq 0$, with $v \in V_B$, ψ a linear expression in V_N and $\trianglerighteq \in \{<, \leq, =, \geq, >\}$.
4. A is a finite set of *(instantaneous/snap) actions* and *durative actions*. An *action* a is a pair $\langle \mathrm{pre}(a), \mathrm{eff}(a)\rangle$ in which (i) $\mathrm{pre}(a)$ are the *(pre)conditions* of a, and (ii) $\mathrm{eff}(a)$ are the *effects* of a of the form $v := \top$, $v := \bot$, $x := \psi$, with $v \in V_B$, $x \in V_N$ and ψ a linear expression in V_N. For each action a, every variable $v \in V_B \cup V_N$ must occur in $\mathrm{eff}(a)$ at most once to the left of the assignment operator ":=", and when this happens v is said to be *assigned* by a. A durative action b is a tuple $\langle b^\vdash, b^\vdash, b^\dashv, [L, U]\rangle$, where $b^\vdash, b^\vdash, b^\dashv$ are the actions *starting*, *lasting* and *ending* b, respectively, and $L, U \in \mathbb{Q}^{>0}$ are bounds on the duration d of b, $L \leq U$. The action $b^\vdash$ has no effects, and its preconditions $\mathrm{pre}(b^\vdash)$ must hold throughout the execution of b. From here on, for simplicity, we consider only durative actions, as snap actions can be treated as durative actions without lasting and ending actions, as in (Panjkovic and Micheli 2023).

Let $\Pi = \langle V_B, V_N, A, I, G\rangle$ be a temporal numeric planning problem. A *timed durative action* is a pair $\langle t, b\rangle$ with $t \in \mathbb{Q}^{\geq 0}$ and b a durative action $\langle b^\vdash, b^\dashv, b^\vdash, [L, U]\rangle$ in which $[L, U]$ is replaced with a single *duration* value $d \in [L, U]$: t (resp. $t + d$) is the time in which $b^\vdash$ (resp. $b^\dashv$) is executed. A *temporal (numeric) plan* π for Π is a finite set of timed durative actions. Thus, in π, multiple snap actions can be executed at the same time, but any two such actions a and a' must be non *mutex*, i.e., a must not interfere with a', and vice versa. An action a *does not interfere* with an action a' if for every variable v assigned by a

1. v does not occur in the preconditions of a', and
2. if $v \in V_B$, either v is not assigned by a' or $v := \top \in \mathrm{eff}(a)$ if and only if $v := \top \in \mathrm{eff}(a')$, and
3. if $v \in V_N$ then either v does not occur in the effects of a' or the only occurrences of v in both a and a' are within linear increments of v. An expression $v := v + \psi$ is a *linear increment of* v if v does not occur in ψ.

If a and a' are not in mutex, the order in which they are executed in any state s is not relevant, i.e., $res(a, res(a', s)) = res(a', res(a, s))$. The expression $res(a, s)$ is the *result of executing a in state s*, which (i) is defined when s satisfies the preconditions of a, and (ii) is the state $s' = res(a, s)$ such that for each $v \in V_B \cup V_N$, $s'(v) = s(e)$ if $v := e \in \mathrm{eff}(a)$, and $s'(v) = s(v)$ otherwise. Given a set $A = \{a_1, \ldots, a_n\}$ of pairwise non mutex actions, we write $res(A, s)$ as an abbreviation for $res(a_1, \ldots, res(a_n, s)\ldots)$, order not relevant.

Consider a temporal plan π. The execution of π induces a sequence of states $s_0, s_1; \ldots; s_m$, each state s_i with an associated time $t_i > t_{i-1}$ at which a non empty set A_i of actions, each starting/ending a durative action in π, is executed. The temporal plan π is *valid* if:

1. s_0 is the initial state, $s_{i+1} = res(A_{i+1}, s_i)$ and s_m satisfies the goal formulas, with $i \in [0, m)$;
2. ϵ-separation: for any pair of mutex actions $a \in A_i$ and $a' \in A_j$, $|t_i - t_j| \geq \epsilon > 0$ (and thus $i \neq j$);
3. no self-overlapping: for any two distinct timed durative actions $\langle t, b\rangle$ and $\langle t', b\rangle$ with durations d and d' respectively, if $t' \geq t$ in π, then $t' \geq t + d$;
4. lasting-action: for each timed durative action $\langle t, \langle b^\vdash, b^\dashv, b^\vdash, d\rangle\rangle$ in π, if $b^\vdash$ and $b^\dashv$ are executed at $t_i = t$ and $t_j = t_i + d$ respectively, the preconditions of $b^\vdash$ are satisfied in each state $s_i, \ldots, s_{j-1}$.

We thus considered the standard notion of validity used, e.g., in (Fox and Long 2003; Rankooh and Ghassem-Sani 2015; Haslum et al. 2019; Panjkovic and Micheli 2024), in which, assuming $V_N = \emptyset$, the problem of deciding the existence of a valid temporal plan is in PSPACE. Other, more general definitions of plan validity can be given, relaxing the second condition to allow for $|t_i - t_j| > 0$ and/or removing the third condition. With such generalizations, the complexity of deciding the existence of a valid temporal plan, still with $V_N = \emptyset$, can become EXPSPACE-complete (Rintanen 2007) and can even become undecidable (Gigante et al. 2022).

Standard Encodings in SMT

Several approaches for computing a valid plan of Π have been proposed, either based on search (see, e.g., (Benton, Coles, and Coles 2012; Gerevini, Saetti, and Serina 2010; Eyerich, Mattmüller, and Röger 2012)) or on planning as satisfiability (see, e.g., (Shin and Davis 2004, 2005; Rankooh and Ghassem-Sani 2015; Rintanen 2015; Cashmore et al. 2016; Rintanen 2017; Cashmore, Magazzeni, and Zehtabi 2020; Panjkovic and Micheli 2023, 2024)). We follow the second approach, in which (i) a *bound* or *number of steps* n (initially set to 0) is fixed, (ii) a corresponding SMT formula is produced, and (iii) a valid plan is returned if the formula is satisfiable, while n is increased and the previous step iterated, otherwise. In more detail, given a temporal numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$ and a value for the bound $n \geq 0$, in the second step, these works:

1. Make $n + 1$ copies of a set $\mathcal{X}$ of *state variables* which includes $V_B \cup V_N$, each copy $\mathcal{X}_i$ meant to represent the state at the i-th step; make n copies of a set $\mathcal{A}$ of *(Boolean) durative action variables* which includes A, each copy $\mathcal{A}_i$ meant to represent the durative actions executed at the i-th step; and introduce a set $\{t_0, \ldots, t_n\}$ of *time variables*, each t_i being the time associated to the i-th state $\mathcal{X}_i$.
2. Impose proper axioms defining the value of the variables in $\mathcal{X}_{i+1}$ based on the values of the variables in $\mathcal{X}_i$, and of the snap actions which are executed in the state $\mathcal{X}_i$. In particular, these axioms enforce in the state $\mathcal{X}_{i+1}$ the effects of the actions executed in the state $\mathcal{X}_i$, and also that no two mutex actions are executed in $\mathcal{X}_i$.

A similar construction underpins also the standard encoding used for classical and numeric planning problems. However, in these contexts, the standard encoding is known to underperform compared to the R encoding by Scala et al. (2016b), the $R^2\exists$ encoding by Bofill, Espasa, and Villaret (2016), and the pattern $\prec$-encoding by Cardellini, Giunchiglia, and Maratea (2024). Indeed, at each step $i \in [0, n)$,

1. in the R encoding, each action variable can be "rolled-up" taking a value in $\mathbb{N}^{\geq 0}$ representing how many times

the action is consecutively executed,

2. the $R^2\exists$ encoding allows for the execution of actions in mutex and/or with contradictory effects, and
3. the $\prec$-encoding allows for the consecutive execution of actions, even if in mutex and with contradictory effects.

As a consequence, the $\prec$-encoding dominates the $R^2\exists$ and R encodings, which in turn dominate the standard encoding. This dominance usually leads to better performance, as the number of solver calls, along with the number of variables and the encoding size, all increase linearly with the bound n.

To highlight the potential benefits of moving from the standard encoding to the $\prec$-encoding also in the temporal numeric setting, consider the following simplified version of the bottle example from (Shin and Davis 2005).

Example. *There is a set $\{1, \ldots, q\}$ of bottles, the first p of which containing l_i litres of liquid ($i \in [1, p]$), and the action $\mathtt{pr}_{i,j}$ of pouring from the i-th bottle (with effects at start) in $[1, p]$ into the j-th bottle in $(p, q]$ (with effects at end), one litre every $d_{i,j}$ seconds. In the current encodings, each $\mathtt{pr}_{i,j}$ is Boolean and thus can be executed at most once in between two consecutive states. Further, time variables are associated to the states. For these reasons, with a current encoding S, the goal of emptying the bottles in $[1, p]$ needs a number of steps $n \geq \max_{i=1}^{p} l_i$, how many depending also on the specific $d_{i,j}$ values since each executed $\mathtt{pr}_{i,j}$ can start/end at a different time from the others. Further, S needs at least $n = \sum_{i=1}^{p} l_i$ steps when $q = p + 1$, due to the conflicting effects of pouring to a single bottle.*

Despite the apparent complexity introduced by the temporal aspects, (Cushing et al. 2007) demonstrated that these problems are no more difficult than their numeric counterparts without the temporal requirements. Indeed, in the above domain each problem admits a solution in which all the durative actions are sequentially executed, one after the other. For this reason, such problems are said to be *without required concurrency* (Cushing et al. 2007), and they can be (more easily) solved by non-temporal planners by (i) replacing each durative action b with a snap action combining the preconditions and effects of $b^{\vdash}, b^{\mapsto}, b^{\dashv}$, (ii) finding a sequential solution to the resulting non-temporal problem, and (iii) post-process the found solution to introduce execution times. We thus consider the following example, whose problems require concurrency.

Example (cont'd). *Consider the previous example extended with $\mathtt{nc}_k$ which at start uncaps the bottle $k \in [1, q]$ and then caps it back after d_k seconds. Any problem in which all the bottles are initially capped requires concurrency, since pouring from i to j is possible only if both bottles i and j are uncapped. This scenario can be modelled in* PDDL *2.1 with $V_B = \{c_k \mid k \in [1, q]\}$, $V_N = \{l_k \mid k \in [1, q]\}$ and the set of durative actions $A = \{\mathtt{nc}_k \mid k \in [1, q]\} \cup \{\mathtt{pr}_{i,j} \mid i \in [1, p], j \in (p, q]\}$ whose actions are:*

$$\mathtt{pr}_{i,j}^{\vdash} : \langle \{c_i = \bot, l_i > 0, c_j = \bot\}, \{l_i \mathrel{-}= 1\} \rangle,$$
$$\mathtt{pr}_{i,j}^{\mapsto} : \langle \{c_i = \bot, c_j = \bot\}, \emptyset \rangle, \mathtt{pr}_{i,j}^{\dashv} : \langle \emptyset, \{l_j \mathrel{+}= 1\} \rangle,$$
$$\mathtt{nc}_k^{\vdash} : \langle \{c_k = \top\}, \{c_k := \bot\} \rangle, \mathtt{nc}_k^{\dashv} : \langle \{c_k = \bot\}, \{c_k := \top\} \rangle.$$

As customary, $v \mathrel{+}= \psi$ (resp. $v \mathrel{-}= \psi$) is an abbreviation for $v := v + \psi$ (resp. $v := v - \psi$). With $q = 2$ and $p = 1$, there are three durative actions $\mathtt{pr}_{1,2}$, $\mathtt{nc}_1$ and $\mathtt{nc}_2$. Considering the starting/ending actions, $\mathtt{pr}_{1,2}^{\vdash}$ is mutex with $\mathtt{nc}_1^{\vdash}$, $\mathtt{nc}_1^{\dashv}$, $\mathtt{nc}_2^{\vdash}$, $\mathtt{nc}_2^{\dashv}$. If the bottles are initially capped and the durations allow to pour all the litres with just one execution of $\mathtt{nc}_1$ and $\mathtt{nc}_2$), we need a bound

1. *$n = l_1 + 3$ with the standard encoding (one step for uncapping the bottles, 1 step for starting the first pour action after ϵ time, l_1 steps for pouring the litres and the final step for executing the capping of the bottles),*
2. *$n = 4$ if we generalize the R encoding since we can roll-up the $\mathtt{pr}_{1,2}$ action and collapse the l_1 steps into 1,*
3. *$n = l_1$ if we generalize the $R^2\exists$ encoding since we can execute all the actions (even the mutex ones) in one step except for the repeated execution of $\mathtt{pr}_{1,2}$ (action variables are still Boolean),*
4. *$n = 1$ if we generalize the $\prec$-encoding since we can execute all the actions in one step.*

If, e.g., $q = 4$ and $p = 2$ in the standard encoding we need $n = l_1 + l_2 + 3$ steps if the durations of the pour actions forces them to start/end at different times, while we can maintain $n = 1$ generalizing the $\prec$-encoding.

Temporal Numeric Planning with Patterns

Let $\Pi = \langle V_B, V_N, A, I, G \rangle$ be a temporal numeric planning problem. Here, we extend the SPP approach to the temporal setting by (i) formally defining the notion of pattern $\prec$ and defining the sets $\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{T}^{\prec}, \mathcal{X}'$ of variables used in our encoding; (ii) extending the definition of rolling to durative actions; (iii) defining the pattern state encoding formula, $T_s^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$, setting the value of each variable in $\mathcal{X}'$ as a function of $\mathcal{X}$ and $\mathcal{A}^{\prec}$; (iv) defining the pattern time encoding formula, $T_t^{\prec}(\mathcal{A}^{\prec}, \mathcal{T}^{\prec})$, enforcing the desired temporal properties of the actions; and (v) proving the correctness and completeness of the presented encoding. Each point is treated in a separate subsection. Intuitively, a pattern is a sequence of starting/ending actions. For each of these actions, the encoding sets (i) an integer specifying how many times the corresponding durative action is executed in sequence, (ii) the conditions for its executability and effects, and (iii) the time at which each durative action sequence has to be started/ended. While durative action sequences that do not interfere might swap order, those that interfere need to maintain the ordering given in the pattern for their starting/ending actions, to ensure executability.

Pattern and Language Definition

A *pattern* is a finite sequence $\prec = a_1; a_2; \ldots; a_k$ of actions, each starting/ending a durative action in A. A pattern is arbitrary, allowing for multiple occurrences of the same action, even consecutively. Each action occurrence in the pattern corresponds to a distinct variable in the encoding, and, given the variable name, we have to be able to uniquely identify

1. which durative action it is starting/ending, and
2. which of the possible multiple occurrences of the action in $\prec$ we are considering.

For these reasons, we perform the following two initial steps which do not affect the generality of our approach:

1. Whenever in A there are two distinct durative actions b_1 and b_2 with $b_1^\vdash = b_2^\vdash$ or $b_1^\vdash = b_2^\dashv$ or $b_1^\dashv = b_2^\dashv$, we break the identity by adding to the preconditions of one of the two actions an always satisfied condition like $0 = 0$, and
2. In a pattern $\prec$, repeated occurrence of an action a are replaced with distinct copies a'. Both a and a' are assumed to be starting/ending the same durative action b, and, abusing notation, we write, e.g., $a = b^\vdash$ and $a' = b^\vdash$.

We can therefore take the action in the pattern to be the action variables in our encoding, and we can assume that each action starts/ends exactly one durative action.

Consider a pattern $\prec = a_1; a_2; \ldots; a_k$, $k \geq 0$. Our encoding is based on the following sets of variables:

1. $\mathcal{X} = V_B \cup V_N$ to represent the initial state;
2. $\mathcal{X}'$ containing a *next state variable* x' for each state variable $x \in \mathcal{X}$, used to represent the goal state;
3. $\mathcal{A}^\prec$ consisting of the set of actions in the pattern $\prec$, each variable a_i ranging over $\mathbb{N}^{\geq 0}$ and whose value represents the number of times the durative action started/ended by a_i is consecutively executed/rolled up, with $i \in [1, k]$;
4. $\mathcal{T}^\prec$, with (i) a variable $t_i \in \mathbb{Q}^{\geq 0}$ representing the time in which the i-th action a_i in $\prec$ is executed; (ii) if a_i is starting b, a variable $d_i \in \mathbb{Q}^{\geq 0}$ representing the time taken by the consecutive execution of b for p times, where $p \geq 0$ is the value assumed by the variable $a_i \in \mathcal{A}^\prec$, and (iii) for convenience, a variable $t_0 = 0$ as the initial time.

In the following we keep using v, w, x for state variables, ψ for a linear expression, a for a (snap) action, b for a durative action, t for a time variable and d for a duration, each symbol possibly decorated with subscripts/superscripts.

Rolling Durative Actions

We start by defining when a durative action b can be rolled up. Intuitively, b can be consecutively executed more than once when (i) the Boolean effects of its starting/ending actions do not disable the repetition of b given the preconditions of its starting/lasting/ending actions, (ii) the numeric effects of $b^\vdash$ and $b^\dashv$ do not interfere between themselves, and (iii) it might be useful to execute b more than once. Formally, we say that b is *eligible for rolling* if the following three conditions are satisfied:

1. if $V, V' \in \{\bot, \top\}$, $V \neq V'$, then (i) $v = V \in \mathrm{pre}(b^\vdash)$ iff $v := V \in \mathrm{eff}(b^\dashv)$ or $v := V' \notin \mathrm{eff}(b^\vdash) \cup \mathrm{eff}(b^\dashv)$, and (ii) $v = V \in \mathrm{pre}(b^\vdash) \cup \mathrm{pre}(b^\dashv)$ iff $v := V \in \mathrm{eff}(b^\vdash)$ or $v := V' \notin \mathrm{eff}(b^\vdash) \cup \mathrm{eff}(b^\dashv)$;
2. if $v := \psi$ is a numeric effect of $b^\vdash$ or $b^\dashv$, then (i) v does not occur in any other effect of $b^\vdash$ or $b^\dashv$, and (ii) either v does not occur in ψ or $v := \psi$ is a linear increment;
3. $b^\vdash$ or $b^\dashv$ include a linear increment in their effects (this last condition imposed for the usefulness of rolling).

If b has a duration in $[L, U]$ and is eligible for rolling, consecutively executing b for $p \geq 1$ times

1. has a duration in $[p \times L + (p-1) \times \epsilon_b, p \times U + (p-1) \times \epsilon_b]$, where $\epsilon_b = \epsilon$ if $b^\vdash$ and $b^\dashv$ are mutex, and $\epsilon_b = 0$ otherwise. Such interval allows for ϵ-separation if $b^\vdash$ and $b^\dashv$ are mutex;
2. causes v to get value $(p \times \psi)$ if $v \mathrel{+}= \psi$ is a linear increment of $b^\vdash$ or $b^\dashv$, while all the other variables keep the value they get after the first execution of b.

Notice that it is assumed that all the consecutive executions of b have the same duration. Indeed, according to the semantics, the duration of b can be arbitrarily fixed as long as each single execution respects the duration constraints, which are part of the domain specification. This assumption does not affect the completeness of our encoding. Should every valid plan require two consecutive executions of b with different durations, we will find a plan when considering a pattern with two or more occurrences of the starting/ending actions of b. Indeed, rolling is an optimization, and our procedure is complete even if we rule out rolling by adding the constraint $a \leq 1$ for each action a.

Then, for each $i \in [0, k]$, the value of a variable $v \in V_B \cup V_N$ after the sequential execution of $a_1; \ldots; a_i$, each action possibly repeated multiple times, is given by $\sigma_i(v)$, inductively defined as $\sigma_0(v) = v$, and, for $i > 0$,

1. if v is not assigned by a_i, $\sigma_i(v) = \sigma_{i-1}(v)$;
2. if $v := \top \in \mathrm{eff}(a_i)$, $\sigma_i(v) = (\sigma_{i-1}(v) \vee a_i > 0)$;
3. if $v := \bot \in \mathrm{eff}(a_i)$, $\sigma_i(v) = (\sigma_{i-1}(v) \wedge a_i = 0)$;
4. if $v \mathrel{+}= \psi \in \mathrm{eff}(a_i)$ is a linear increment,
$$\sigma_i(v) = \sigma_{i-1}(v) + a_i \times \sigma_{i-1}(\psi),$$
i.e., the value of v is incremented by $\sigma_{i-1}(\psi)$ a number of times equal to the value assumed by the variable a_i;
5. if $v := \psi \in \mathrm{eff}(a_i)$ is not a linear increment,
$$\sigma_i(v) = \mathrm{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v)),$$
where $\mathrm{ITE}(c, t, e)$ (for "*If c Then t Else e*" returns t or e depending on whether c is true or not, and is a standard function in SMTLIB (Barrett, Fontaine, and Tinelli 2016).

Above and in the following, for any linear expression ψ and $i \in [0, k]$, $\sigma_i(\psi)$ is the expression obtained by substituting each variable $v \in V_N$ in ψ with $\sigma_i(v)$.

Given a durative action b eligible for rolling and a state s, to determine the maximum number of times that b can be executed consecutively in s, we rely on the following Theorem, in which $\psi[p, b^\vdash, q, b^\dashv]$ represents the value of ψ after p and q repetitions of the actions $b^\vdash$ and $b^\dashv$, respectively. Formally, $\psi[p, b^\vdash, q, b^\dashv]$ is the expression obtained from ψ by substituting each variable x with

1. $x + p \times \psi'$ (resp. $x + q \times \psi'$), when $x \mathrel{+}= \psi' \in \mathrm{eff}(b^\vdash)$ (resp. $x \mathrel{+}= \psi' \in \mathrm{eff}(b^\dashv)$) is a linear increment, and
2. ψ'', when $x := \psi'' \in \mathrm{eff}(b^\vdash) \cup \mathrm{eff}(b^\dashv)$ is not a linear increment.

Theorem 1. *Let b be a durative action eligible for rolling. Let s be a state. The result of executing $b^\vdash; b^\vdash; b^\dashv$ consecutively for $p \geq 1$ times in s is defined if and only if for each numeric condition $\psi \trianglerighteq 0$,*

1. *if $\psi \trianglerighteq 0 \in \mathrm{pre}(b^\vdash)$, s satisfies $\psi[0, b^\vdash, 0, b^\dashv] \trianglerighteq 0$ (i.e., $\psi \trianglerighteq 0$) and $\psi[p-1, b^\vdash, p-1, b^\dashv] \trianglerighteq 0$;*
2. *if $\psi \trianglerighteq 0 \in \mathrm{pre}(b^\vdash) \cup \mathrm{pre}(b^\dashv)$, s satisfies $\psi[1, b^\vdash, 0, b^\dashv] \trianglerighteq 0$ and $\psi[p, b^\vdash, p-1, b^\dashv] \trianglerighteq 0$.*

Proof. The thesis follows from the monotonicity in p of the functions $\psi[p-1, b^{\vdash}, p-1, b^{\dashv}]$ and $\psi[p, b^{\vdash}, p-1, b^{\dashv}]$ (see (Scala et al. 2016b)). □

Example (cont'd). *For $i \in [1,p]$, $j \in (p,q]$, the pouring action $pr_{i,j}$ is eligible for rolling while both nc_i and nc_j are not. Action $pr_{i,j}$ can be consecutively executed for l_i times in the states in which bottles i and j are uncapped and at least l_i litres are in the i-th bottle.*

The Pattern State Encoding

Let $\prec = a_1; a_2; \ldots; a_k$, $k \geq 0$, be a pattern. The pattern state encoding defines the executability conditions of each action and how to compute the value of each variable in $\mathcal{X}'$ based on the values of the variable in $\mathcal{X}$ and in $\mathcal{A}^{\prec}$. Formally, the *pattern state $\prec$-encoding* $T_s^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ *of* Π is the conjunction of the formulas in the following sets:

1. $\mathrm{pre}^{\prec}(A)$: for each $i \in [1,k]$ and for each $v = \bot, w = \top$, $\psi \trianglerighteq 0$ in $\mathrm{pre}(a_i)$:
   (a) $\neg v$ and w must hold to execute a_i:
   $$a_i > 0 \rightarrow (\neg\sigma_{i-1}(v) \wedge \sigma_{i-1}(w)),$$
   (b) and, if a_i is starting b, (i.e., if $a_i = b^{\vdash}$) (Theorem 1):
   $$a_i > 0 \rightarrow \sigma_{i-1}(\psi[0, b^{\vdash}, 0, b^{\dashv}]) \trianglerighteq 0,$$
   $$a_i > 1 \rightarrow \sigma_{i-1}(\psi[a_i - 1, b^{\vdash}, a_i - 1, b^{\dashv}]) \trianglerighteq 0,$$
   (c) if a_i is ending b, (i.e., if $a_i = b^{\dashv}$) (Theorem 1, noting that in σ_{i-1}, $b^{\vdash}$ has been executed a_i times):
   $$a_i > 0 \rightarrow \sigma_{i-1}(\psi[-a_i + 1, b^{\vdash}, 0, b^{\dashv}]) \trianglerighteq 0,$$
   $$a_i > 1 \rightarrow \sigma_{i-1}(\psi[0, b^{\vdash}, a_i - 1, b^{\dashv}]) \trianglerighteq 0.$$
2. $\mathrm{amo}^{\prec}(A)$: for each $i \in [1,k]$, if a_i is starting a durative action which is not eligible for rolling:
   $$a_i \leq 1.$$
3. $\mathrm{frame}^{\prec}(V_B \cup V_N)$: for each variable $v \in V_B$ and $w \in V_N$:
   $$v' \leftrightarrow \sigma_k(v), \qquad w' = \sigma_k(w).$$

Example (cont'd). *Assume $p = 2$ and $q = 4$. Let*

$$\begin{array}{c} nc_1^{\vdash}; nc_2^{\vdash}; nc_3^{\vdash}; nc_4^{\vdash}; pr_{1,3}^{\vdash}; pr_{1,4}^{\vdash}; pr_{2,3}^{\vdash}; pr_{2,4}^{\vdash}; \\ nc_1^{\dashv}; nc_2^{\dashv}; nc_3^{\dashv}; nc_4^{\dashv}; pr_{1,3}^{\dashv}; pr_{1,4}^{\dashv}; pr_{2,3}^{\dashv}; pr_{2,4}^{\dashv}. \end{array} \quad (1)$$

be the fixed pattern $\prec$. Assume $i \in [1,2]$, $j \in [3,4]$, $k \in [1,4]$. The pattern state encoding entails $(nc_k^{\vdash} \leq 1)$ since the durative action nc is not eligible for rolling, and

$$nc_i^{\vdash} > 0 \rightarrow c_i, \;\; nc_i^{\dashv} > 0 \rightarrow \neg(c_i \wedge nc_i^{\vdash} = 0),$$
$$pr_{i,j}^{\vdash} > 0 \rightarrow (\neg(c_i \wedge nc_i^{\vdash} = 0) \wedge \neg(c_j \wedge nc_j^{\vdash} = 0)),$$
$$pr_{i,3}^{\vdash} > 0 \rightarrow l_i > 0, \qquad pr_{i,4}^{\vdash} > 0 \rightarrow l_i - pr_{i,3}^{\vdash} > 0,$$
$$pr_{i,3}^{\vdash} > 1 \rightarrow pr_{i,3}^{\vdash} < l_i, \qquad pr_{i,4}^{\vdash} > 1 \rightarrow pr_{i,4}^{\vdash} < l_i - pr_{i,3}^{\vdash},$$
$$c'_k \equiv (c_k \wedge nc_k^{\vdash} = 0) \vee nc_k^{\dashv} > 0,$$
$$l'_i = l_i - pr_{i,3}^{\vdash} - pr_{i,4}^{\vdash}, \qquad l'_j = l_j + pr_{1,j}^{\dashv} + pr_{2,j}^{\dashv}.$$

The first four lines define the preconditions for executing each action, and the last two specify the frame axioms.

As the frame axioms in the example make clear, the $\prec$-encoding allows in the single state transition from $\mathcal{X}$ to $\mathcal{X}'$ (i) the multiple consecutive execution of the same action, as in the rolled-up R encoding (Scala et al. 2016b), and (ii) the combination of multiple even contradictory effects on a same variable by different actions, as in the $R^2\exists$ encoding (Bofill, Espasa, and Villaret 2016).

The Pattern Time Encoding

Let $\prec = a_1; a_2; \ldots; a_k$, $k \geq 0$, be a pattern. The pattern time $\prec$-encoding associates to each action a_i in $\prec$ a starting time t_i and duration d_i, which are both set to 0 when a_i is not executed, i.e., when $a_i = 0$. In defining the constraints for t_i and d_i they have to respect the semantics of temporal planning problems and also the causal relations between the actions in the pattern and exploited in the pattern state $\prec$-encoding. Consider for instance two actions a_i and a_j in $\prec$ with $i < j$, $a_i > 0$ and $a_j > 0$. We surely have to guarantee that $t_i < t_j$ if a_i and a_j are in mutex: the formulas checking that the preconditions of a_j (resp. a_i) are satisfied, take into account that a_i (resp. a_j) has been (resp. has not been) executed *before* a_j (resp. a_i). Even further, we have to impose that $t_i + \epsilon \leq t_j$ for the ϵ-separation rule. If, on the other hand, a_i and a_j are not in mutex, then it is not necessary to guarantee $t_i < t_j$ unless a_j is ending the durative action started by a_i or because of the lasting action of the durative action started by a_j. As an example of the impact of the lasting action on the encoding, assume a_j is starting action b. Then, it may be the case a_i is not in mutex with a_j but it is in mutex with the lasting action $b^{\vdash}$ of b. Hence, the formulas checking the executability of $b^{\vdash}$ encode that a_i precedes a_j in the pattern, and consequently we will have to guarantee $t_i < t_j$.

Given the above, the *pattern time $\prec$-encoding* $T_t^{\prec}(\mathcal{A}^{\prec}, \mathcal{T}^{\prec})$ *of* $\prec$ is the conjunction of $(t_0 = 0)$ and the following formulas:

1. $\mathrm{dur}^{\prec}(A)$: for each durative action $\langle b^{\vdash}, b^{\dashv}, b^{\vdash\dashv}, [L,U]\rangle \in A$ and for each action $a_i = b^{\vdash}$ and $a_j = b^{\dashv}$ in $\prec$:
   $$a_i > 0 \rightarrow t_i \geq t_0 + \epsilon,$$
   $$a_i = 0 \rightarrow t_i = t_0 \wedge d_i = 0, \;\; a_j = 0 \rightarrow t_j = t_0,$$
   $$a_i > 0 \rightarrow a_i \times (L + \epsilon_b) \leq d_i + \epsilon_b \leq a_i \times (U + \epsilon_b).$$
   The last formula guarantees also ϵ-separation when b is consecutively executed, and $b^{\vdash}$ and $b^{\dashv}$ are in mutex.
2. $\text{start-end}^{\prec}(A)$: for each durative action b, each starting action $a_i = b^{\vdash}$ (resp. ending action $a_j = b^{\dashv}$) in $\prec$ must have a matching ending (resp. starting) action:
   $$a_i > 0 \rightarrow \bigvee_{j \in E_i}(a_i = a_j \wedge t_j = t_i + d_i),$$
   $$a_j > 0 \rightarrow \bigvee_{i \in S_j}(a_i = a_j \wedge t_j = t_i + d_i),$$
   where $E_i = \{j \in (i,k] \mid a_i = b^{\vdash}, a_j = b^{\dashv}\}$, and $S_j = \{i \in [1,j) \mid a_j = b^{\dashv}, a_i = b^{\vdash}\}$.
3. $\mathrm{epsilon}^{\prec}(A)$: every two actions a_i and a_j in $\prec$ with $j < i$ are ϵ-separated if they are mutex or different copies of the same action:
   $$a_i > 0 \rightarrow (t_i \geq t_j + \epsilon).$$
   Further, for every two actions a_i and a_j starting respectively b and b', if the starting or ending action of b is mutex with the starting or ending action of b':
   $$a_i > 1 \rightarrow (t_i \geq t_j + d_j \vee t_j \geq t_i + d_i \vee a_j = 1 \wedge t_i \geq t_j \wedge t_i + d_i \leq t_j + d_j).$$
   This formula ensures that the start/end actions of b' are not executed during the multiple consecutive executions of b, thereby guaranteeing ϵ-separation.

4. $\text{noOverlap}^{\prec}(A)$: for each durative action b, each starting action $a_i = b^{\vdash}$ in $\prec$ can be executed only after the previous executions of b ended:

$$a_i > 0 \rightarrow \bigwedge_{j \in B_i}(t_i \geq t_j + d_j),$$

where $B_i = \{j \in [1, i) \mid a_i = b^{\vdash}, a_j = b^{\vdash}\}$.

5. $\text{lasting}^{\prec}(A)$: for each durative action b with $\text{pre}(b^{\vdash}) \neq \emptyset$, and for each action $a_i = b^{\vdash}$ in $\prec$:

(a) The preconditions of $b^{\vdash}$ must be satisfied in each (consecutive) execution of b, i.e., for each $v = \bot$, $w = \top$, $\psi \trianglerighteq 0$ in $\text{pre}(b^{\vdash})$ (Theorem 1):

$$a_i > 0 \rightarrow \neg\sigma_i(v) \wedge \sigma_i(w) \wedge \sigma_{i-1}(\psi[1, b^{\vdash}, 0, b^{\dashv}]) \trianglerighteq 0,$$
$$a_i > 1 \rightarrow \sigma_{i-1}(\psi[a_i, b^{\vdash}, a_i - 1, b^{\dashv}]) \trianglerighteq 0.$$

(b) For each action a_j in $\prec$ mutex with $b^{\vdash}$,

i. if $j < i$, then a_j cannot be executed after a_i:

$$a_i > 0 \rightarrow t_i \geq t_j,$$

and, when a_j is a starting action, also:

$$a_i > 0 \wedge a_j > 1 \rightarrow t_i \geq t_j + d_j.$$

These formulas ensure that b does not start until all executions of a_j happened.

ii. if $j > i$ and a_j is executed before b ends, then (i) no rolling takes place:

$$t_0 + \epsilon \leq t_j < t_i + d_i \rightarrow a_i \leq 1 \wedge a_j \leq 1,$$

and (ii) a_j has to maintain the preconditions of $b^{\vdash}$, i.e., for each $v = \bot$, $w = \top$, $\psi \trianglerighteq 0$ in $\text{pre}(b^{\vdash})$:

$$t_0 + \epsilon \leq t_j < t_i + d_i \rightarrow \neg\sigma_j(v) \wedge \sigma_j(w) \wedge \sigma_j(\psi) \trianglerighteq 0.$$

Example (cont'd). *For $\texttt{pr}^{\vdash}_{i,j}$ (resp. $\texttt{nc}^{\vdash}_k$), let $t^{\vdash}_{i,j}$ (resp. $t^{\vdash}_k$) be the associated time variable, and analogously for the ending actions. If we further assume that when executed, the durations $d_{i,j}$ and d_k of $\texttt{pr}_{i,j}$ and $\texttt{nc}_k$ are 1 and 5 respectively, the temporal pattern encoding entails:*

$$\texttt{nc}^{\vdash}_k = 0 \rightarrow d_k = 0, \texttt{nc}^{\vdash}_k = 1 \rightarrow d_k = 5, \texttt{pr}^{\vdash}_{i,j} = d_{i,j},$$
$$\texttt{nc}^{\vdash}_k = \texttt{nc}^{\dashv}_k,\ \ \texttt{pr}^{\vdash}_{i,j} = \texttt{pr}^{\dashv}_{i,j},\quad \neg(t^{\vdash}_{i,j} \leq t^{\dashv}_i < t^{\vdash}_{i,j} + d_{i,j}),$$
$$\texttt{pr}^{\vdash}_{i,j} > 0 \wedge \texttt{nc}^{\vdash}_i = 1 \rightarrow t^{\vdash}_{i,j} \geq t^{\vdash}_i + \epsilon.$$

The formulas in the 3 lines respectively say that (i) uncapping a bottle takes 5s and pouring p litres takes p seconds, (ii) any started durative action has to be ended and it is not possible to cap a bottle while pouring from it, and (iii) we can start pouring from a bottle after ϵ time since we uncapped it. Similar facts hold for the destination bottles.

Correctness and Completeness Results

Let $\prec = a_1; a_2; \ldots; a_k$, $k \geq 0$, be a pattern. Though the pattern $\prec$ can correspond to any sequence of starting/ending actions of a durative action in A, it is clear that it is pointless to have (i) an ending action $b^{\dashv}$ without the starting action $b^{\vdash}$ before $b^{\dashv}$ in $\prec$; similarly (ii) a starting action $b^{\vdash}$ which is not followed by the ending action $b^{\dashv}$, and (iii) two consecutive occurrences of the same starting (ending) action in the pattern. In such cases, the pattern can be safely simplified by eliminating such actions. On the other hand, it makes sense to consider patterns with non consecutive occurrences of the same starting/ending action. Assuming b_1 and b_2 are two durative actions with $b_1^{\vdash}/b_1^{\dashv}$ mutex with $b_2^{\vdash}$, it might be useful to have a pattern including $b_1^{\vdash}; b_1^{\dashv}; b_2^{\vdash}; b_1^{\vdash}; b_1^{\dashv}$ to allow two executions of b_1, or b_1 to start/end before/after b_2 starts. No matter how $\prec$ is defined, the *$\prec$-encoding $\Pi^{\prec}$ of Π (with bound 1)* is correct, where

$$\Pi^{\prec} = I(\mathcal{X}) \wedge T_s^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}') \wedge T_t^{\prec}(\mathcal{A}^{\prec}, \mathcal{T}^{\prec}) \wedge G(\mathcal{X}'), \quad (2)$$

in which $I(\mathcal{X})$ and $G(\mathcal{X}')$ are formulas encoding the initial state and the goal conditions. To any model μ of $\Pi^{\prec}$ we associate the valid temporal plan π whose durative actions are started by the actions a_i in $\prec$ with $\mu(a_i) > 0$. Specifically, if $a_i = b^{\vdash}$, in π we have $\mu(a_i)$ consecutive executions of b, i.e., one timed durative actions $\langle t, \langle b^{\vdash}, b^{\vdash\dashv}, b^{\dashv}, d\rangle\rangle$ for each value of $p \in [0, \mu(a_i))$. The $(p+1)$-th execution of b happens at the time t and has duration d such that

$$t = \mu(t_i) + p \times (d + \epsilon_b),\ (d + \epsilon_b) \times \mu(a_i) = \mu(d_i) + \epsilon_b.$$

Completeness is guaranteed once we ensure that the sequence $\pi^{\downarrow}$ of the starting/ending actions of a valid temporal plan π, listed according to their execution times, is a subsequence of the pattern used in the encoding. This can be achieved by starting with a complete pattern, and then repeatedly chaining it till $\Pi^{\prec}$ becomes satisfiable. Formally, a pattern $\prec$ is *complete* if for each durative action $b \in A$, $b^{\vdash}$ and $b^{\dashv}$ occur in $\prec$. Then, we define $\prec^n$ to be the sequence of actions obtained concatenating $\prec$ for $n \geq 1$ times. Finally, $\Pi_n^{\prec}$ is the *pattern $\prec$-encoding of Π with bound n*, obtained from (2) by considering $\prec^n$ as the pattern $\prec$.

Theorem 2. *Let Π be a temporal numeric planning problem. Let $\prec$ be a pattern. Any model of $\Pi^{\prec}$ corresponds to a valid temporal plan of Π (correctness). If Π admits a valid temporal plan and $\prec$ is complete, then for some $n \geq 0$, $\Pi_n^{\prec}$ is satisfiable (completeness).*

Proof (hint). Correctness: Let μ be a model of $\Pi^{\prec}$ and π its associated plan. The ϵ-separation axioms ensure that the relative order between mutex actions in π and in $\prec$ is the same. The pattern state encoding ensures that executing sequentially the actions in π starting from I leads to a goal state. The axioms in the pattern time encoding are a logical formulation of the corresponding properties for the validity of π. Completeness: Let π be a valid temporal plan with n durative actions. Let $\prec_\pi$ be the pattern consisting of the starting and ending actions in π listed according to their execution times. The formula $\Pi^{\prec_\pi}$ is satisfied by the model μ whose associated plan is π. For any complete pattern $\prec$, $\prec_\pi$ is a subsequence of $\prec^{2\times n}$ and $\Pi_{2\times n}^{\prec}$ can be satisfied by extending μ to assign 0 to all the action variables not in $\prec_\pi$. □

Notice that when two actions a and a' are not in mutex and one is not the starting/ending action of the other, the pattern does not lead to an ordering on their execution times. For this reason, we may find a valid plan π for Π even before $\prec^n$ becomes a supersequence of $\pi^{\downarrow}$, $\pi^{\downarrow}$ defined as above.

Domain	Coverage (%)						Time (s)						Bound (Common)		
	PATTY$_T$	ANMLSMT	ITSAT	LPG	OPTIC	TFD	PATTY$_T$	ANMLSMT	ITSAT	LPG	OPTIC	TFD	PATTY$_T$	ANMLSMT	ITSAT
Temporal	**9**	**4**	**2**	**1**	**4**	**0**	**6**	**1**	**0**	**0**	**3**	**0**	**10**	**0**	**0**
CUSHING	**100.0**	30.0	-	-	**100.0**	10.0	**1.70**	235.35	-	-	3.12	270.02	**3.00**	11.33	-
POUR	**95.0**	5.0	-	-	-	-	**46.51**	285.96	-	-	-	-	**2.00**	15.00	-
SHAKE	**100.0**	50.0	-	-	-	-	**1.11**	155.15	-	-	-	-	**2.00**	9.50	-
PACK	**60.0**	5.0	-	-	-	-	**154.72**	285.00	-	-	-	-	**1.00**	6.00	-
BOTTLES	**10.0**	5.0	-	-	-	-	**284.28**	286.36	-	-	-	-	**7.00**	18.00	-
MAJSP	**85.0**	50.0	-	-	-	-	**90.54**	154.02	-	-	-	-	**8.40**	15.00	-
MATCHAC	**100.0**	**100.0**	**100.0**	-	**100.0**	-	2.20	0.46	0.71	-	**0.01**	-	**3.85**	10.00	4.00
MATCHMS	**100.0**	**100.0**	**100.0**	-	**100.0**	-	1.22	0.43	0.68	-	**0.01**	-	**3.60**	10.00	4.00
OVERSUB	**100.0**	**100.0**	-	**100.0**	**100.0**	-	1.02	0.05	-	0.08	**0.01**	-	**1.00**	4.00	-
PAINTER	35.0	**45.0**	-	-	10.0	-	211.69	**194.67**	-	-	270.03	-	**2.40**	16.80	-

Table 1: Comparative analysis. A "-" indicates that no result was obtained in our 300s time limit, either due to a timeout or an issue with the planner. The best results are in bold.

Example (cont'd). *Assume all $q \geq 2 \times p$ bottles are initially capped and that the bottles in $[1, p]$ contain $< d_k = 5$ litres. Then, $\Pi^{\prec}$ is satisfiable and a valid plan is found with one call to the* SMT *solver. Notice that in the pattern (1), the ending action $\mathtt{pr}^{\dashv}_{i,j}$ of the pouring actions are after the ending action $\mathtt{nc}^{\dashv}_k$ that caps the bottle. However, such two actions are not in mutex and our pattern time encoding does not enforce $t^{\dashv}_{i,j} > t^{\dashv}_k$, making it possible to solve the problem with a bound $n = 1$. On the other hand, if one bottle contains 5 litres, $\Pi^{\prec}$ is unsatisfiable because of ϵ-separation between the actions of uncapping and pouring from it, making it impossible to pour 5 times before the bottle is capped again. This problem is solved having $\prec^n$ with $n = 2$. More complex scenarios may require $\prec^n$ with higher values for n.*

Experimental Results

Table 1 presents the experimental analysis on the CUSHING domain (the only domain with required concurrency in the last International Planning Competition (IPC) with a temporal track (Coles et al. 2018)), all the domains and problems presented in (Panjkovic and Micheli 2023) (last five), and four new domains covering different types of required concurrency specified in (Cushing et al. 2007). The first new domain, POUR, is similar to the motivating example of this paper. SHAKE allows emptying a bottle by shaking it while uncapped. PACK calls for concurrently pairing two bottles together to be packed. The domain BOTTLES puts together all the actions and characteristics of the three aforementioned domains. Of these 10 domains, only POUR and BOTTLES, contain actions eligible for rolling.

The analysis compares our system PATTY$_T$ implemented by modifying the planner PATTY (Cardellini, Giunchiglia, and Maratea 2024) and using the SMT-solver Z3 v4.8.7 (De Moura and Bjørner 2008); the symbolic planners ANMLSMT (which corresponds to ANML$^{\text{OMT}}_{\text{INC}}$(OMSAT) in (Panjkovic and Micheli 2023)) and ITSAT (Rankooh and Ghassem-Sani 2015); and the search-based planners OPTIC (Benton, Coles, and Coles 2012), LPG (Gerevini, Saetti, and Serina 2010) and TEMPORALFASTDOWNWARD (TFD) (Eyerich, Mattmüller, and Röger 2012). ANMLSMT and OPTIC have been set in order to return the first valid plan they find. To use ANMLSMT, we manually converted the domains in PDDL 2.1 to the ANML language (Smith, Frank, and Cushing 2008). The experiments have been run using the same settings used in the Numeric/Agile Track of the last IPC, with 20 problems per domain and a time limit of 5 minutes. Analyses have been run on an Intel Xeon Platinum 8000 3.1GHz with 8 GB of RAM. In the table we show: the percentage of solved instances (Coverage); the average time to find a solution, counting the time limit when the solution could not be found (Time); the average bound at which the solutions were found, computed on the problems solved by all the symbolic planners able to solve at least one problem in the domain (Bound). The value of the bound coincides with the number of calls to the SMT solver. Each pattern $\prec$ is computed only once using the Asymptotic Relaxed Planning Graph, introduced in (Scala et al. 2016a) and already used in (Cardellini, Giunchiglia, and Maratea 2024). [1]

From the table, as expected PATTY$_T$ finds a solution with a bound always lower than the ones needed by the other symbolic planners. This allows PATTY$_T$ to have the highest coverage in 9 out of 10 domains (compared to the value 4 for the second best). The Painter domain is the only one where PATTY$_T$ has a lower coverage than ANMLSMT. ANMLSMT is a symbolic planner exploiting the standard encoding. Although it requires a higher bound to find a valid plan also in Painter, ANMLSMT encoding has 2490 mostly Boolean variables (action and most state variables are Boolean), while our encoding has 2058 mostly numeric variables (the only Boolean variables are in $\mathcal{X}$ and $\mathcal{X}'$). In the other domains, the ratio between the number of variables used by ANMLSMT and PATTY$_T$ is 0.16 on average, which provides an explanation of PATTY$_T$'s highest coverage and better performance on 9/10 and 6/10 domains, respectively. Overall, PATTY$_T$ is able to solve 157 out of the 200 considered problems, compared to the 98 of the second best.

Conclusion

We extended the SPP approach proposed in (Cardellini, Giunchiglia, and Maratea 2024) to the temporal numeric setting. We proved its correctness and completeness, and showed its benefits on various domains with required concurrency. As expected, all the problems have been solved by PATTY$_T$ with a bound lower than the one needed by the other planners based on planning as satisfiability.

[1] PATTY$_T$ and the PDDL 2.1 and ANML encoding of the new domains are available at https://github.com/matteocarde/patty .

Acknowledgments

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.

References

Balyo, T. 2013. Relaxing the Relaxed Exist-Step Parallel Planning Semantics. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, 865–871. IEEE Computer Society.

Barrett, C.; Fontaine, P.; and Tinelli, C. 2016. The Satisfiability Modulo Theories Library (SMT-LIB). www.smt-lib.org.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2021. Satisfiability Modulo Theories. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 1267–1329. IOS Press.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Bofill, M.; Espasa, J.; and Villaret, M. 2016. The RANTANPLAN planner: system description. *The Knowledge Engineering Review*, 31(5): 452–464.

Bofill, M.; Espasa, J.; and Villaret, M. 2017. Relaxed Exists-Step Plans in Planning as SMT. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 563–570. ijcai.org.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2024. Symbolic Numeric Planning with Patterns. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Washington, DC, USA, February 7-14, 2023*. AAAI Press.

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In Coles, A. J.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, 79–87. AAAI Press.

Cashmore, M.; Magazzeni, D.; and Zehtabi, P. 2020. Planning for Hybrid Systems via Satisfiability Modulo Theories. *J. Artif. Intell. Res.*, 67: 235–283.

Coles, A.; Coles, A.; Martinez, M.; and Sidiropoulos, P. 2018. International Planning Competition 2018 - Temporal Track. https://bitbucket.org/ipc2018-temporal/domains/src/master/. Accessed: 2023-08-01.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is Temporal Planning Really Temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

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

Eyerich, P.; Mattmüller, R.; and Röger, G. 2012. Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In *Towards Service Robots for Everyday Environments - Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*, volume 76 of *Springer Tracts in Advanced Robotics*, 49–64. Springer.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2010. Temporal Planning with Problems Requiring Concurrency through Action Graphs and Local Search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 226–229. AAAI.

Gigante, N.; Micheli, A.; Montanari, A.; and Scala, E. 2022. Decidability and complexity of action-based temporal planning over dense time. *Artif. Intell.*, 307: 103686.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. ISBN 978-3-031-00456-8.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*.

Panjkovic, S.; and Micheli, A. 2023. Expressive Optimal Temporal Planning via Optimization Modulo Theory. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 12095–12102. AAAI Press.

Panjkovic, S.; and Micheli, A. 2024. Abstract Action Scheduling for Optimal Temporal Planning via OMT. In *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 2024*. AAAI Press.

Rankooh, M. F.; and Ghassem-Sani, G. 2015. ITSAT: An Efficient SAT-Based Temporal Planner. *J. Artif. Intell. Res.*, 53: 541–632.

Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 280–287. AAAI.

Rintanen, J. 2015. Models of Action Concurrency in Temporal Planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1659–1665. AAAI Press.

Rintanen, J. 2017. Temporal Planning with Clock-Based SMT Encodings. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 743–749. ijcai.org.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016a. Interval-Based Relaxation for General Numeric Planning. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 655–663.

Scala, E.; Ramirez, M.; Haslum, P.; and Thiebaux, S. 2016b. Numeric Planning with Disjunctive Global Constraints via SMT. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26: 276–284.

Shin, J.; and Davis, E. 2004. Continuous Time in a SAT-Based Planner. In McGuinness, D. L.; and Ferguson, G., eds., *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 531–536. AAAI Press / The MIT Press.

Shin, J.; and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artif. Intell.*, 166(1-2): 194–253.

Smith, D. E.; Frank, J.; and Cushing, W. 2008. The ANML language. In *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, volume 31.