Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
National Doctoral Program in Artificial Intelligence (37th cycle)

Symbolic Pattern Planning

By

Matteo Cardellini

******

Supervisors:
Prof. E. Giunchiglia
Prof. M. Maratea
Prof. M. Vallati

Doctoral Examination Committee:
Dr. Nicola Gigante, Referee, Free University of Bolzano
Dr. Andrea Micheli, Referee, Fondazione Bruno Kessler
Prof. Stefano Di Carlo, Polytechnic University of Turin
Prof. Simon Parkinson, University of Huddersfield
Prof. Marco Roveri, University of Trento

Politecnico di Torino
2025

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Matteo Cardellini
2025

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

What could the storyteller add to what the actor has already brought to life?

Abstract

Planning involves finding a sequence of actions – i.e., a plan – that allows agents, operating within an environment, to reach a goal from a specified initial condition. Different flavours of planning exist depending on how the environment is represented: *classical* if it is represented only by propositional variables, *numeric* if variables can also assume numeric values, *temporal* if there is a concept of time, with actions having durations, and *hybrid* if the environment can act on its own. A technique for planning is *Planning as Satisfiability*, where one bounds the length of the plan by an integer n, translates a planning problem into a propositional formula encoding all possible plans up to length n, and checks for the satisfiability of the formula, increasing n upon failure.

This thesis introduces a novel approach called *Symbolic Pattern Planning* (SPP), based on Planning as Satisfiability. Given a planning problem, we propose to construct a plan for it by first fixing a pattern –defined as an arbitrary finite sequence of actions, roughly sketching an intuitive idea on how the actions in the final plan should be ordered– and then encoding as a formula the state resulting from the sequential execution of a subsequence of the actions in the pattern, starting from an arbitrary initial state. By imposing the conditions on the initial and goal states, we can check whether the pattern allows determining a valid plan – i.e., the formula is satisfiable – or whether the pattern needs to be extended and the procedure iterated until we find a valid plan as a subsequence of the pattern. We ground our proposal in the classical, numeric and temporal flavours of planning and prove the correctness (any returned plan is valid) and completeness (if there exists a valid plan, one will be returned) of the procedure. Moreover, for the three flavours, we show that our encoding allows to determine a valid plan in a number of iterations n which is never higher than the one needed by the state-of-the-art

planners for that flavour exploiting the planning as satisfiability approach. On the experimental side, we ran an extensive analysis for the numerical and temporal flavours, showing that the results validate the theoretical findings and that our planner PATTY has excellent comparative performances.

Regarding the classical planning flavour, we consider it as a special case of numeric planning – where there are no numeric variables – and we concentrate our attention to the special case of *classical planning with conditional effects*, where effects, i.e., the way the environment changes after an action in applied, can be *conditional*, meaning that their application depends on the agent's state before applying the action. This seemingly harmless modification makes the problem much more difficult than classical planning, and requires some non-trivial adjustments to the approach.

After presenting the SPP approach for all the aforementioned flavours, we push the envelope further and overcome some of its limitation, i.e., how the selection of a static pattern computed on the initial state can be detrimental when searching for plans where the order between actions changes during the plan. We show how to symbolically search for a valid plan by iteratively extending (adding actions to) and simplifying (removing actions from) the initially computed pattern, symbolically mimicking standard search-based procedures for planning. Again, the proposed procedure is proven correct and complete and, on the experimental side, it outperforms the previous SPP approach.

Finally, we conclude the thesis with a discussion on how to apply SPP techniques on the hybrid planning flavour and on an application domain, the *In-Station Train Dispatching Problem*, i.e., the problem of planning the movements of trains inside a railway station.

Contents

List of Figures

List of Tables

List of Theorems

List of Algorithms

Chapter 1

Introduction on Planning

1.1 Planning

Planning involves finding a sequence of actions – i.e., a plan – that allows an agent, acting in an environment, to reach a goal from a specified initial condition. Making plans is often considered a marker of intelligence, which is why automated planning has been one of the oldest problems in Artificial Intelligence (AI) [McCarthy and Hayes, 1969]. A planning system, or *planner*, takes a formalized problem model as input and applies problem-solving techniques –such as *heuristic search* [Bonet and Geffner, 2001] or *satisfiability* [Kautz and Selman, 1992] – to find a plan. Transforming the model into a search space or logical reasoning task, and developing heuristics and strategies to solve it efficiently, are challenges handled by the system's designer. The planner itself does not need to understand the nature of the problem; it can work with any problem that can be represented in the modelling language. This characteristic of planners is referred to as *domain-independence*.

The planning phase primarily addresses state transformation problems involving a set of actions that can alter the state of the system (i.e., the environment and the agents together). A plan is a sequence of actions in that set, possibly repeated, that, when executed, transitions the system from an initial state to a desired goal state. Depending on the specific nature of the planning problem, the plan could either be a sequence of actions or a detailed schedule, where actions are carried out at specific times.

1.2 Flavours of Planning

As stated before, domain-independent planners can solve only problems that can be represented in the modelling language for which the planner is built for. In this thesis, we call these modelling languages *flavours*. To describe flavours, we will employ an alternative definition of planning, based on the concept of *autonomous agents*. An autonomous agent[1] is a system – virtual or physical – which has *sensors* and *actuators*. Sensors are used to inspect the current state of the environment in which the agent operates, and actuators are used to change the state of the environment or of the agent itself. For example, a *vacuum robot* [Russell and Norvig, 2020] has (i) sensors to understand if there is dirt underneath itself and a localization sensor to understand where it is placed in a room, and (ii) actuators to vacuum the dirt and to move inside a room. This agent-based model allows defining different *flavours* of planning:

1. *Deterministic Planning*: where the sensors can observe the whole state of the world and the actuators can change the state of the world in an exact and reliable way. For example, the vacuum robot can precisely determine its position in the room and whether there is dirt below itself, and can precisely move in the room.

2. *Nondeterministic Planning*: where the agent can observe the state of the environment through perfect sensors, but the actuators could lead the agent to one among possible known states, and thus the agent can know the resulting state only after the actuators are used. For example, the vacuum robot's actuators could fail to vacuum correctly the dirt or its movements could be imprecise. The Nondeterministic Planning flavour presents in two variants:

   (a) *Fully Observable Nondeterministic* (FOND) Planning [Daniele et al., 1999] deals primarily with finding a *contingent plan*, i.e., simulating every potential outcome of a nondeterministic action and finding a plan which is guaranteed to reach the desired goal, even if the actuators fail,

[1] In this section, we use the singular term *agent* for simplicity, but in reality, planning can also deal with multiple agents. To make things easier, we can consider as a single agent the centralized control system that controls and gives command to all the (sub-)agents.

(b) *Probabilistic Planning* [Yoon et al., 2007] focuses on maximizing the probability of achieving the goal, considering only one randomly chosen action outcome for each action in the plan.

The state of the art for both FOND planning and probabilistic planning involves making the actions deterministic by replacing every nondeterministic action with a set of deterministic ones and using a deterministic planner to find a solution [Muise et al., 2012]. In this thesis, we concentrate mainly on the deterministic flavour, since improving this flavour also improves the nondeterministic one.

A plan is said to be *valid* when, among other properties which are dependent on the flavour, it can bring the agent from an initial state to a goal state. Other flavours of planning exist depending on whether the initial condition is or is not completely defined [Nebel, 2000] and thus, in the latter case, a plan must be found for all possible initial conditions. In this thesis, we assume the initial condition to be fully specified.

In the case of deterministic planning, usually, actions have *preconditions* – associated with the sensors of the agent – which is a condition imposing when the action is applicable, and *effects* – associated with the actuators of the agent – which describe how the state of the environment change after the action is executed. Using preconditions and effects, we can again categorize planning into different flavours:

1. In *Classical Planning* [McDermott, 2000; Bacchus, 2001] the environment is represented only by propositional variables (i.e., variables which can be either true or false), preconditions are propositional formulas over these propositional variables and effects can either put a variable to true or false. Actions are *instantaneous* (or atomic), meaning that there cannot be any other action applied in between the time the preconditions of the action are checked, and the effects of the action are applied. A valid plan is a sequence of actions, which, executed one after the other, leads from an initial state to a goal state.

2. In *Classical Planning with Conditional Effects* (CEs) (see, e.g., Nebel [2000]), we are in the same conditions as Classical Planning, but effects are *conditional*, meaning that even if an action is applied, its

effects can be applied or not, depending on whether a condition (i.e., a propositional formula) holds in the state before applying the action.

3. In *Numeric Planning* (see e.g., Fox and Long [2003] or Scala and Gerevini [2020]), the environment can be represented by propositional and numerical variables (ranging over $\mathbb{Q}$). Actions are instantaneous.

4. In *Temporal Planning* (see, e.g., Gigante et al. [2022]), the environment can be represented by propositional variables or also include numeric variables – in which case it is referred to as *Temporal Numeric Planning*. Actions are not instantaneous and can have durations (fixed or flexible). The preconditions can be checked at the start, at the end or throughout the action and effects can be applied at the start or at the end of the action. A plan is now a set of *timestamped actions*, telling when each action in the plan should start and its duration. For this reason, actions can now be executed in parallel.

5. In *Hybrid Planning* (see, e.g., Fox and Long [2006]), actions can have durations and the environment can act on its own with *events* and *processes*, which are not under the control of the agent. Processes and events have preconditions and effects and, while processes' effects represent continuous change in numeric variables, events' effects are immediately applied as soon as their preconditions are met. While the agent can always decide not to perform an action, even if its conditions are respected, events must be triggered as soon as their conditions are respected. For example, if we model the task of cooking pasta, the agent actions could be: putting the pot with water on the stove, turning on the stove, adding salt to the water or dropping the pasta in to the water. A process could take care of increasing the temperature of the water when the stove is on and the pot is on top of the stove and an event could signal that the water has reached $100°$C, it is boiling, and the temperature should no longer be increased. The process and the event cannot be postponed. For these reasons, hybrid planning has already proved very effective in solving complex real-world problems such as Traffic Control [Vallati et al., 2016], Train Dispatching [Cardellini et al., 2021a,b,c] – which we will explore in Chapter 6 –, Unmanned Aerial

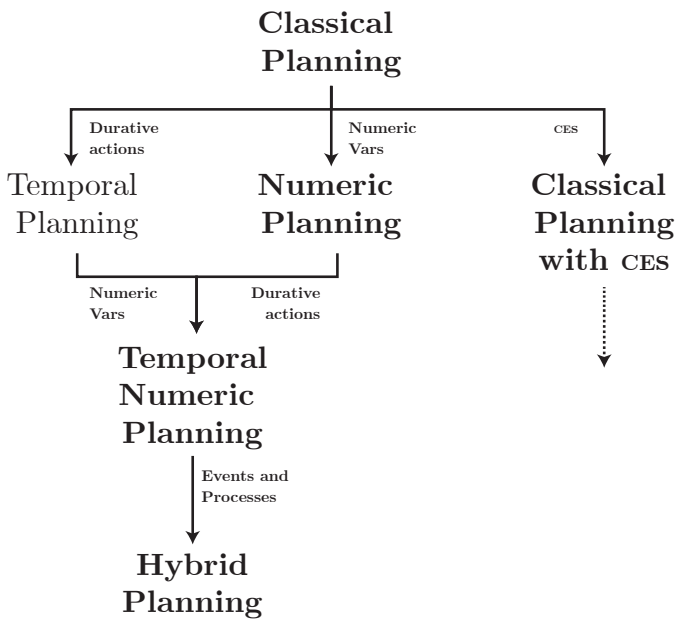

Fig. 1.1 Hierarchy of the different flavours of deterministic planning. Bold indicates flavours which are covered in this thesis.

Vehicle Control [Kiam et al., 2020] and Pharmacokinetic Optimization [Alaboud and Coles, 2019].

Several other flavours exist (see, e.g., Haslum et al. [2019]). In this thesis, we focus on these five flavours, given the massive literature focused on improving them, and the presence of several benchmark domains in the International Planning Competitions (IPC) [Taitler et al., 2024].

Moreover, one could notice how one flavour can be considered as a special case of another flavour. Fig. 1.1 shows the hierarchy of the flavours for deterministic planning. We can see that

1. Classical Planning is a special case of Classical Planning with CEs where all the CEs have effects where the condition is always respected (i.e., true)

2. Classical Planning is a special case of Numeric Planning where all the variables are propositional.

3. Numeric Planning (resp. Classical Planning) is a special case of Temporal Numeric Planning (resp. Temporal Planning) where all the actions have duration equal to zero.

4. Temporal Numeric Planning is a special case of Hybrid Planning where there are no events and processes.

The dashed arrow in Fig. 1.1 means that all the flavours above Classical Planning (i.e., Numeric, Temporal, Temporal Numeric and Hybrid) could, in theory, be defined also with CEs (e.g., Numeric Planning with CEs). We decided not to investigate these flavours further, since most of the literature with CEs is focused on the classical case.

1.3 Complexity of Planning

We now briefly review the literature on the study of the complexity of planning, depending on the flavour. Since we are interested in the decision problem, we define PLANEX to be the problem of deciding whether a valid plan exists for a planning task.

Theorem 1.1 (Bylander [1994])**.** PLANEX *for a classical planning task is* PSPACE*-complete.*

Theorem 1.2 (Nebel [2000])**.** PLANEX *for a classical planning task with* CE*s is* PSPACE*-complete.*

Theorem 1.3 (Helmert [2002])**.** PLANEX *for a numeric planning task is undecidable.*

Thm. 1.3 undecidability mainly comes from the fact that the state-space when considering also numeric variables in $\mathbb{Q}$ is infinite. For this reason, Gigante and Scala [2023] introduced the concept of *Bounded Numeric Planning* where each numeric variable (i) has an upper and lower bound, and (ii) has a discrete domain.

Theorem 1.4 (Gigante and Scala [2023])**.** PLANEX *for a bounded numeric planning task is* PSPACE*-complete.*

Regarding the temporal planning flavour (without numeric variables), Gigante et al. [2022] showed that the complexity depends on the particular semantics used. We will see in Chapter 4 that there exist different interpretations of plan validity depending on whether

1. *self-overlapping*: we allow self-overlapping of actions (i.e., the same action can be applied in parallel multiple times),
2. ϵ-*separation*: mutex actions (i.e., two actions that either modify each the same variable or where one action has in the preconditions a variable modified by the other action) are required to happen at times separated by a small value ϵ, or
3. >0-*separation*: mutex actions are required to not happen at the same time.

In Chapter 4, we will focus on the case where we disallow self-overlapping and guarantee ϵ-separation.

Theorem 1.5 (Gigante et al. [2022])**.** PLANEX *for temporal planning without self-overlapping and* ϵ*-separation is* PSPACE*-complete.* PLANEX *for temporal planning with self-overlapping and* ϵ*-separation is* EXPSPACE*-complete.* PLANEX *for temporal planning with self-overlapping and* >0*-separation is undecidable.*

In Temporal Numeric Planning, we directly inherit the properties of (unbounded) numeric planning, making it undecidable. For Hybrid Planning, in the literature there is no thorough analysis of its complexity, due also to a lack of formal description of its properties and characteristics. However, in the general case, it is easy to see that PLANEX for Hybrid Planning could be undecidable due to the presence of numeric continuous variables. Some investigation has yet to be made on how the complexity changes when we restrict the domain of numeric variables (like in Bounded Numeric Planning) and if we consider different semantics (like in Temporal Planning).

1.3.1 Compilation Schemes

As stated by Thms. 1.1, 1.2, 1.4 and 1.5, all the flavours we cover in this thesis are PSPACE-complete.[2] One could think that, since they all belong to the same complexity class, problems of the different flavours have the same "difficulty". This, however, is not supported by the literature, where approaches which

[2]We will introduce in Chapter 2 an approach for solving unbounded numeric planning, which is undecidable in general. However, many of the problems in the literature and in real-world instances present integer bounded variables.

have been proposed for a flavour fail to be applicable to other flavours, or in applications, where problems which can be swiftly solved by a solver of a flavour become much harder to solve when we introduce elements (e.g., numeric variables or durations) that bring the problem to another flavour.

For this reason, we need another concept to categorise how "difficult" it is to find a plan for a particular flavour. For this reason, *compilation schemes* have been introduced for classical planning and classical planning with CEs by Nebel [2000] and then expanded for numeric planning by Gigante and Scala [2023]. Intuitively, we say that a planning problem expressed in flavour A is *compilable into* another planning problem expressed in flavour B if there exists a function f mapping each planning problem Π_A in A into a planning problem $f(\Pi_A)$ in B such that (i) every plan for Π_A can be mapped into a plan for $f(\Pi_A)$ and (ii) the mapping f is polynomial-time and polynomial-space computable. Moreover, Nebel [2000] categorizes compilation schemes based on the plan's size of the compiled problem w.r.t. the original planning problem. Let π_A be a (valid) plan for Π_A and π_B be a (valid) plan for $f(\Pi_A)$ and let $|\pi_A|, |\pi_B|$ be their length. We say that

1. the compilation scheme *preserves plan size exactly* if $|\pi_B| \leq |\pi_A| + k$ for some $k \in \mathbb{N}^{\geq 0}$,
2. the compilation scheme *preserves plan size linearly* if $|\pi_B| \leq c \times |\pi_A| + k$ for some $c \in \mathbb{Q}^{>0}$ and $k \in \mathbb{N}^{\geq 0}$,
3. the compilation scheme *preserves plan size polynomially* if $|\pi_B| \leq p(|\pi_A|, |\Pi_A|)$ for some polynomial p depending on the length of π_A and on the original problem Π_A.

Intuitively, the desired property for compilation schemes is to preserve plan size exactly, since expanding the plan size usually denotes a longer time required to find a solution.

With this intuition of compilation schemes, we can recall some previous work from the literature

Theorem 1.6 (Nebel [2000])**.** *A classical planning problem with* CE*s cannot be compiled to a classical planning problem without* CE*s preserving plan size linearly.*

Theorem 1.7 (Gigante and Scala [2023])**.** *A (bounded) numeric planning problem can be compiled into a classical planning problem with* CE*s preserving plan size exactly, and vice versa.*

It is clear that the above theorem does not hold for unbounded numeric planning, due to its infinite state-space. While Thm. 1.7 shows that there could be some symmetries between (bounded) numeric planning and planning with CEs, the presented compilation by Gigante and Scala [2023] makes use of a translation from boolean formulas to non-linear numeric conditions (e.g., $p_a \wedge p_b$ with $p_a, p_b \in \{\top, \bot\}$ is translated into $n_a \times n_b = 1$ with $n_a, n_b \in \{0, 1\}$). This non-linearity of conditions – as we will explore in Chapter 2 – becomes very problematic for many solvers, and thus the compilation results are difficult to exploit.

Without explicitly employing compilation schemes, Cushing et al. [2007] shows "what makes temporal planning really temporal" and defines what are the characteristics that a temporal planning task must have to be *temporally expressive*. Most importantly, Cushing et al. introduce the property of *required concurrency* – i.e., where every plan must have at least two actions that are scheduled to happen in parallel. If a planning task doesn't require concurrency, then the problem can be translated into a classical planning problem, thus removing the temporal characteristics of the task. If, instead, the problem is temporally expressive, and thus it requires concurrency, then this translation cannot be performed without losing completeness (i.e., all the existing plans can be found).

For the Hybrid Planning flavour, Percassi et al. [2023a,b] show how it is possible to compile a *discrete hybrid planning problem* – i.e., where time is discretised by a discretisation step – directly into numeric planning while using two compilations that they name EXP and POLY. The first leads to a numeric planning task which is exponentially larger than the original hybrid planning task input but preserves the plan length. The second one keeps the resulting formulation polynomial but increases the plan length polynomially. Even if Percassi et al. do not prove that the compilation cannot be done without a polynomial increase in the plan length, it is believed that this is the case.

1.4 Satisfiability Modulo Theory

1.4.1 Satisfiability

The *satisfiability problem* (SAT) is formally defined as follows. Given a *propositional formula* $f(x_1, \ldots, x_n)$, composed of n *propositional variables*, *logical connectives* (e.g., $\wedge$ for *AND*, $\vee$ for *OR*, and $\neg$ for *NOT*), and parentheses for grouping, determine whether there exists a *truth assignment* or *model* to the variables $x_1, \ldots, x_n$ that satisfies f. A model is a mapping $\mu : \{x_1, x_2, \ldots, x_n\} \mapsto \{\top, \bot\}$, where $\top$ and $\bot$ are the symbols for *true* and *false*. The goal is to check whether there exists a model μ such that f evaluates to true under σ. Formally, this can be written as:

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

where $\equiv$ is the symbol for *logical equivalence* (e.g., $\top \vee \bot \equiv \top$). The formula μ is often expressed in *Conjunctive Normal Form* (CNF), where it is represented as a conjunction of *clauses*, each clause being a disjunction of *literals*. A literal is either a variable x_i or its negation $\neg x_i$. Thus, a CNF formula f can be expressed as:

$$f = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where each clause C_i is of the form:

$$C_i = (l_{i,1} \vee \cdots \vee l_{i,k_i}),$$

with $k_i \geq 1$, and $l_{i,j}$ being a literal. The formula f, expressed in CNF is satisfiable if and only if there exists a model μ such that at least one literal in every clause C_i is true.

1.4.2 Theories

Satisfiability Modulo Theories (SMT) extends the SAT problem by incorporating background theories, enabling reasoning over richer logical structures. Formally, SMT addresses the problem of determining the satisfiability of logical

formulas expressed in first-order logic regarding one or more background theories. These theories include linear arithmetic, bit-vectors, arrays, uninterpreted functions, and others.

Background theories enrich the expressiveness of SMT by providing specific semantic structures. Commonly used theories include:

- **Equality and Uninterpreted Functions (EUF):** Provides reasoning over uninterpreted function symbols, where no additional constraints on their behaviour are assumed except for equality.
- **Linear Arithmetic:** Supports reasoning over integers ($\mathbb{Z}$) and real numbers ($\mathbb{R}$) under linear constraints, e.g., $3x + 2y \geq 7$.
- **Bit-Vectors:** Enables reasoning about fixed-width binary representations, useful for hardware verification and cryptography.
- **Arrays:** Provides reasoning about indexed data structures with axioms such as *read-over-write*.
- **Quantifiers:** Allows universally or existentially quantified formulas, e.g., $\forall x\, P(x)$, to be incorporated into SMT queries.

In this thesis, we will employ SMT with the Linear Arithmetic theory.

1.5 Planning as Satisfiability

We now show the *Planning as Satisfiability* (PAS) technique [Kautz and Selman, 1992, 1996] to find valid plans for a planning task. We focus our attention to the case of deterministic planning problems with fully specified initial conditions, which covers all the flavours that we have presented in the previous section. Let Π be such a planning task. Independently of the flavour, we can represent the planning task through the following objects.

1. $\mathcal{X}$ is a set of *state variables*, each one equipped with a domain representing the values it can take. These variables represent the *current state*. Depending on the flavour, the variables can be propositional or numeric.

2. $\mathcal{A}$ is a set of *action variables*, each one again equipped with a domain. These variables model the execution of the actions of Π. Usually, there is at least a variable for every action. If the variable associated to an action is propositional, it models whether the action is applied or not, while if the variable is numeric, it models how many times the action is repeated (as we will see from Chapter 2, this is called *rolling*).

3. $\mathcal{X}'$ is a copy of $\mathcal{X}$ and represents the *next state*.

4. $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ is the *symbolic transition relation*, a formula in the variables $\mathcal{X} \cup \mathcal{A} \cup \mathcal{X}'$. Intuitively, the transition relation models how the current state $\mathcal{X}$ evolves into $\mathcal{X}'$, depending on the (possible) executions of the actions in $\mathcal{A}$. Together with $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$, a *decoding function* has to be defined enabling to associate to each model of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ at least one sequence of the actions of Π. Standard requirements for $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ are *correctness* and *completeness*, which depend on the flavour of Π. Correctness means that all sequences modelled by $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ are valid according to the particular semantic of the flavour and make $\mathcal{X}$ transition into $\mathcal{X}'$. Completeness means that $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ can capture all the valid sequences of actions which make $\mathcal{X}$ transition into $\mathcal{X}'$.

5. $\mathcal{I}(\mathcal{X})$ is the *initial state formula*, imposing the fully specified initial condition of Π on the state variables.

6. $\mathcal{G}(\mathcal{X})$ is the *goal formula*, imposing the goal condition of Π on the state variables.

Thus, a *(planning as satisfiability) encoding* E *of* Π is a tuple

$$\Pi^E = \langle \mathcal{X}, \mathcal{A}, \mathcal{I}(\mathcal{X}), \mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}'), \mathcal{G}(\mathcal{X}) \rangle \tag{1.1}$$

In the planning as satisfiability approach [Kautz and Selman, 1992], we fix an integer $n \geq 0$ called *bound* or *number of steps*, we make $n+1$ disjoint copies $\mathcal{X}_0, \ldots, \mathcal{X}_n$ of the set $\mathcal{X}$ of state variables, and n disjoint copies $\mathcal{A}_0, \ldots, \mathcal{A}_{n-1}$ of the set $\mathcal{A}$ of action variables, and define

1. $\mathcal{I}(\mathcal{X}_0)$ as 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 $i = 0, \ldots, n-1$, $\mathcal{T}(\mathcal{X}_i, \mathcal{A}_i, \mathcal{X}_{i+1})$ as 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. $\mathcal{G}(\mathcal{X}_n)$ as 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 *encoding* Π^E *of* Π *with bound* n is the formula

$$\Pi^E_n = \mathcal{I}(\mathcal{X}_0) \wedge \bigwedge_{i=0}^{n-1} \mathcal{T}(\mathcal{X}_i, \mathcal{A}_i, \mathcal{X}_{i+1}) \wedge \mathcal{G}(\mathcal{X}_n). \tag{1.2}$$

To each model μ of Π^E_n, we associate the set of sequences of actions $\alpha_0; \ldots; \alpha_{n-1}$, where each α_i is a sequence of actions corresponding to the model of the formula $\mathcal{T}(\mathcal{X}_i, \mathcal{A}_i, \mathcal{X}_{i+1})$ obtained by restricting μ to $\mathcal{X}_i \cup \mathcal{A}_i \cup \mathcal{X}_{i+1}$, $i \in [0, n)$. In the following, $(\Pi^E_n)^{-1}$ is the set of sequences of actions in A associated to a model of Π^E_n. The correctness of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ ensures the *correctness of* Π^E: for each bound n, each sequence in $(\Pi^E_n)^{-1}$ is a plan. The completeness of $\mathcal{T}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ ensures the *completeness of* Π^E: if there exists a plan for Π, it will be found by considering Π^E_0, Π^E_1, ... It is clear that the number of variables and size of Eq. 1.2 increase with the bound n, explaining why much of the research has concentrated on how to produce encodings allowing to find plans with the lowest possible bound n.

1.6 Thesis Contribution

The common thread of this thesis lies in the last sentence of the previous section. We introduce the Symbolic Pattern Planning (SPP) approach – based on the PAS approach– for different flavours of planning, and we show that our approach can *dominate* all the other approaches in the literature. Given two planning as satisfiability encodings E_1 and E_2 we say that E_1 *dominates* E_2 if, for each bound n, $\Pi^{E_2}_n$ satisfiability implies that also $\Pi^{E_1}_n$ 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_1 will find a plan with a lower or equal bound than E_2. We will present our

approach for Classical and Numeric Planning (Chapter 2), Classical Planning with Conditional Effects (Chapter 3) and Temporal Planning (Chapter 4). We will also perform experimental analysis on well-known domains, showing that our solver PATTY outperforms most of the state-of-the-art planners in the literature in the characterised flavours. Successively, in Chapter 5, we will present a flavour-independent approach that allows to boost even more the performances of PATTY, employing an approach mimicking standard search procedures. In Chapter 6, we will formalise hybrid planning and discuss how our approach can be extended also to this flavour. Moreover, we will present an application – the In-Station Train Dispatching Problem – and discuss how our approach can greatly benefit this application setting. Finally, in Chapter 7, we will recap the work done in the thesis, we will provide some pointers for future work and we will discuss on the ethical problems concerning planning and AI in general.

Chapter 2

SPP in Classical and Numeric Planning

The chapter[1] is structured as follows. After the preliminaries on how to define a numeric planning problem Π in PDDL2.1 (Section 2.1), we present our SPP encoding, proving how it allows defining correct and complete procedures for Π in Section 2.2. In the same section, we present the outlined pattern selection procedures and address the plan quality problem, respectively. In Section 2.3 we frame our encoding in the planning as satisfiability approach, and show that our encoding provably dominates the state-of-the-art rolled-up and $R^2\exists$ encodings. We end the chapter with the experimental analysis, the conclusions, and perspectives on future work. One running example is used throughout the chapter to illustrate the formal definitions and the theoretical results.

2.1 Numeric Planning in PDDL2.1

As briefly outlined in Chapter 1, there are many languages for specifying planning problems. Here, we specifically consider *numeric planning problems*

[1] Part of this chapter has been published in [Cardellini et al., 2024b]

specified in PDDL2.1, level 2 [Fox and Long, 2003], standardly defined as a tuple $\Pi = \langle V_B, V_N, A, I, G\rangle$ in which[2]

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 ψ a *linear expression* over V_N, i.e., with ψ 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 ψ 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.

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 $v \in V_B \cup V_N$,

[2]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.

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

Consider a finite sequence of actions $\alpha = a_1; \ldots; a_n$ of length $n \geq 0$ (for $n = 0$, α reduces to the empty sequence ϵ).

The *state sequence* $s_0; \ldots; s_n$ *induced by* α *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. α is *executable in* s_0,
2. s_n is the *result of executing* α *in* s_0, which will be denoted also with $res(\alpha, s_0)$, i.e.,

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

Notice that for any state s, the empty sequence ϵ 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 α is a *(valid) plan*.

Example 2.1. *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 ≤ 0 and ≥ 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 at rate q, the two robots must connect setting a Boolean variable p to $\top$, and this is possible only if they have the same position. Once connected, they must disconnect before moving again. The quantity q can be positive or negative, corresponding to l giving objects to r or vice versa. This scenario can be modelled in PDDL with $V_B = \{p\}$, $V_N = \{x_l, x_r, q_l, q_r, q\}$ and the*

following set of actions:

$$\begin{gathered}
\texttt{lftr} : \langle \{x_r > 0\}, \{x_r \mathrel{-}= 1\}\rangle,\ \texttt{rgtr} : \langle \{p = \bot\}, \{x_r \mathrel{+}= 1\}\rangle, \\
\texttt{lftl} : \langle \{p = \bot\}, \{x_l \mathrel{-}= 1\}\rangle,\ \texttt{rgtl} : \langle \{x_l < 0\}, \{x_l \mathrel{+}= 1\}\rangle, \\
\texttt{conn} : \langle \{x_l = x_r\}, \{p := \top\}\rangle,\ \texttt{disc} : \langle \{p = \top\}, \{p := \bot\}\rangle, \\
\texttt{exch} : \langle \{p = \top, q_l \geq q, q_r \geq -q\}, \{q_l \mathrel{-}= q, q_r \mathrel{+}= q\}\rangle, \\
\texttt{lre} : \langle \{\}, \{q := 1\}\rangle, \texttt{rle} : \langle \{\}, \{q := -1\}\rangle.
\end{gathered} \tag{2.1}$$

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

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*

$$\texttt{rgtl}^{X_I}; \texttt{lftr}^{X_I}; \texttt{conn}; \texttt{exch}^{Q}; \texttt{disc}; \texttt{lftl}^{X_I}; \texttt{rgtr}^{X_I} \tag{2.2}$$

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 in Eq. 2.2, 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 chapter, v, w, x, y denote variables, a, b denote actions and ψ 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 α, or
2. a plan, in which case we will use the letter π, or
3. a pattern (see later), 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 α and α', $\alpha; \alpha'$ denotes the sequence of actions obtained by concatenating α' to the end of α. Finally, we continue to use standard logical terminology using terms like satisfiable, contradictory and valid, taking them for granted.

2.2 Symbolic Pattern Planning

Consider a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G \rangle$, and a *pattern* $\prec = a_1; a_2; \ldots; a_k$, 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 ϵ), 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 $a1, a2, \ldots,$ 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 (subsection 2.2.1), proving its correctness and completeness assuming the corresponding correctness and completeness of our pattern $\prec$-encoding $\Pi^{\prec}$ of Π. The formal definition of $\Pi^{\prec}$, together with the proof of its correctness and completeness, is given in subsection 2.2.2. Different procedures to compute patterns and high-quality plans are presented in subsections 2.2.3 and 2.2.4, respectively.

2.2.1 A Simple SPP Procedure

The basic idea of SPP is to define the value of each state variable in the state resulting from the execution of a subsequence α of the pattern $\prec$ as a function of both the state in which α starts and of the pattern $\prec$. More in details, assume that to each action occurrence a_i in the pattern we associate a distinct action variable whose value denotes whether a_i has been executed or not after $a_1; \ldots; a_{i-1}$. Then, every subsequence α of $\prec$

1. corresponds to one assignment to the action variables in the encoding, and
2. assuming α is executable in a state s and that $s' = res(\alpha, s)$, since Π is deterministic it is possible to express 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 $\prec$.

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 natural numbers, the value of each variable modeling how many times the action is being consecutively executed, assuming it is possible (and useful) to execute an action consecutively more than once in Π.

Then, the *(SPP) $\prec$-encoding of* Π is a formula having form

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

in which

1. $\mathcal{I}(\mathcal{X})$ is the *initial state formula*, a formula in 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 in 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 $\mathcal{X} \cup \mathcal{A}^{\prec}$.

Indeed, each $\prec$-encoding of Π 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 Π.

Algorithm 2.1 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)      /* Π = ⟨V_B, V_N, A, I, G⟩ */
2:     n ← 0; ≺ ← ϵ;
3:     ≺_I ← ComputePattern(Π);
4:     while (True) do
5:         Π^≺ ← I(X) ∧ T^≺(X, A^≺, X') ∧ G(X');
6:         μ ← Solve(Π^≺);
7:         if (μ ≠ 0) then
8:             return GetPlan(μ, ≺);
9:         end if
10:        ≺ ← ≺; ≺_I;
11:        n ← n + 1;
12:    end while
13: end function
```

For the *completeness* of the $\prec$-encoding, we require that if there exists a subsequence α of $\prec$ which is a plan of Π, $\Pi^{\prec}$ is satisfiable.

Then, if for any pattern $\prec$ we are able to define a correct and complete $\prec$-encoding $\Pi^{\prec}$ of Π, the following simple SPP procedure is guaranteed to return a plan for Π 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,
3. extend $\prec$ by concatenating $\prec_I$ to it and iterate the second step upon its failure.

If π is a plan of length n, π 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 π is a subsequence of the pattern being tested, and even before if the plan π contains multiple consecutive occurrences of a same action and our encoding allows to model such consecutive executions with a single action variable in $\prec_I$.

Algorithm 2.1 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 in 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 μ of $\Pi^{\prec}$, i.e., the sequence of actions

$$a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \ldots; a_k^{\mu(a_k)}. \tag{2.3}$$

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).

Theorem 2.1. *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.*

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 π be a plan of length n. Then, after the n-th iteration of the loop in SPP(Π), π 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

$$\prec_I \leftarrow \text{COMPUTEPATTERNI}(\Pi, \prec);$$

in between lines 9 and 10, in which COMPUTEPATTERNI($\Pi, \prec$) is assumed to return a complete pattern.

2. compute an n-complete pattern to be used in the next iteration. The new n-complete pattern may be entirely different from the previously used pattern. This modification amounts to remove line 3 and replace line 10. with the line of code

$$\prec \leftarrow \textsc{ComputePatternN}(\Pi, \prec);$$

in which $\textsc{ComputePatternN}(\Pi, \prec)$ is assumed to return a n-complete pattern.

It is clear that SPP(Π) as in Algorithm 2.1 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.

2.2.2 A Correct and Complete SPP Encoding for Numeric Planning Problems

Given the pattern $\prec = a_1; \ldots; a_k$, $k \geq 0$, for $i \in [0, k]$, we write $\prec_i$ as an abbreviation of the initial pattern $a_1; a_2; \ldots; a_i$ of $\prec = \prec_k$, with $\prec_0 = \epsilon$.

We now formally define a correct and complete $\prec$-encoding $\Pi^{\prec}$ of Π, which amounts to define the $\prec$-symbolic transition relation $\mathcal{T}^{\prec}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$.

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

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 action a in it (its position is not relevant at this time). Intuitively, as proposed by [Scala et al., 2016d], the value assumed by the action variable $a \in \mathcal{A}^{\prec}$ represents the number ≥ 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(Π) 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 difficult 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 ψ a linear expression not containing any of the variables assigned by a, as for the effects of the action `exch` and `lftr` in Eq 2.1, 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 Eq. 2.1, and

   (b) *self-interfering assignments*, otherwise.

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

1. a does not contain a self-interfering assignment, and

2. 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 2.2. *[Scala et al. (2016d)] Let Π 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

[3]Here, we consider just the cases $\alpha = 0$ and $\alpha = 1$ of Theorem 1 in [Scala et al., 2016d], which (quoting) "cover a very general class of dynamics, where rates of change are described by linear or constant equations".

1. *for each* $v = \bot$ *(resp.* $w = \top$*) in* $\mathrm{pre}(a)$*,* $s(v) = \bot$ *and* $v := \top \notin \mathrm{eff}(a)$ *(resp.* $s(w) = \top$ *and* $w := \bot \notin \mathrm{eff}(a)$*), and*

2. *for each numeric precondition* $\psi \trianglerighteq 0$ *in* $\mathrm{pre}(a)$*,*

$$s(\psi) \trianglerighteq 0, \quad s(\psi[m]) \trianglerighteq 0, \tag{2.4}$$

   where $\psi[m]$ *is the linear expression obtained from* ψ *by substituting each variable* x *with*

   (a) $x + (m-1) \times \psi'$*, whenever* $x \mathrel{+}= \psi' \in \mathtt{eff}(a)$ *is a linear increment,*

   (b) ψ'*, whenever* $x := \psi' \in \mathrm{eff}(a)$ *is a simple assignment.*

3. *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 Eq. 2.4 ensure that $\psi \trianglerighteq 0$ holds in the states in which the first and the last execution of a_i happens. The satisfaction of these two conditions ensure that each precondition $\psi \trianglerighteq 0$ of a_i is satisfied also in the intermediate states s in between the first and the last execution of a_i. This is a consequence of the fact, proved in [Scala et al., 2016d], that the function $\psi[a_i]$ is monotonic in a_i assuming the action is eligible for rolling.

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

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 $v := \top \in \mathrm{eff}(a_i)$, 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 $v := \bot \in \mathrm{eff}(a_i)$, 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 $v \mathrel{+}= \psi \in \mathrm{eff}(a_i)$ is a linear increment, the value of v will be incremented by the value of ψ 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 ψ with $\sigma_{i-1}(v)$;

5. if $v := \psi \in \mathrm{eff}(a_i)$ 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

$$\sigma_i(v) = \mathrm{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v)),$$

where $\mathrm{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v))$ 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 [Barrett et al., 2016].

Example 2.2. *Consider Eq. 2.1, and assume $\prec$ is*

$$\texttt{lre}; \texttt{rle}; \texttt{lftr}; \texttt{rgtl}; \texttt{conn}; \texttt{exch}; \texttt{disc}; \texttt{rgtr}; \texttt{lftl}. \tag{2.5}$$

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 ≥ 0 times, is

1. *for the Boolean variable* p*,*

$$\sigma(p) = (p \vee \texttt{conn} > 0) \wedge \texttt{disc} = 0,$$

2. *and, for the numeric variables in* $V_N = \{x_l, x_r, q_l, q_r, q\}$*,*

$$\begin{aligned}
\sigma(x_l) &= x_l + \texttt{rgtl} - \texttt{lftl},\\
\sigma(x_r) &= x_r - \texttt{lftr} + \texttt{rgtr},\\
\sigma(q_l) &= q_l - \texttt{exch} \times q^{\texttt{rle}},\\
\sigma(q_r) &= q_r + \texttt{exch} \times q^{\texttt{rle}},\\
\sigma(q) &= q^{\texttt{rle}}.
\end{aligned}$$

in which $q^{\texttt{rle}}$ *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*

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

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:*

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

modelling that 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*

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

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.

If $\prec$ is

$$\mathtt{lre}1; \mathtt{rle}1; \mathtt{lftr}; \mathtt{rgtl}; \mathtt{conn}; \mathtt{exch}; \mathtt{disc}; \mathtt{rgtr}; \mathtt{lftl}; \mathtt{lre}2; \mathtt{rle}2,$$

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:

$$\sigma(q_l) = q_l - \mathtt{exch} \times q^{\mathtt{rle}1},$$
$$\sigma(q_r) = q_r + \mathtt{exch} \times q^{\mathtt{rle}1},$$
$$\sigma(q) = q^{\mathtt{rle}2}.$$

in which $q^{\mathtt{rle}1}$ abbreviates the term $\mathrm{ITE}(rle1 > 0, -1, \mathrm{ITE}(lre1 > 0, 1, q))$*, and $q^{\mathtt{rle}2}$ abbreviates the term* $\mathrm{ITE}(rle2 > 0, -1, \mathrm{ITE}(lre2 > 0, 1, q^{\mathtt{rle}1}))$.

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

$$a_i = 0 \vee a_i = 1,$$

whenever the action a_i is not eligible for rolling.[4]

[4] If the action is not eligible for rolling then a can be defined as a Boolean variable.

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

$$a_i > 0 \rightarrow \bigwedge_{v=\top \in \mathrm{pre}(a_i)} \sigma_{i-1}(v) \wedge \bigwedge_{v=\bot \in \mathrm{pre}(a_i)} \neg\sigma_{i-1}(w),$$

and, for each numeric precondition $\psi \trianglerighteq 0$ in $\mathrm{pre}(a_i)$,

$$a_i > 0 \rightarrow \sigma_{i-1}(\psi) \trianglerighteq 0, \quad a_i > 1 \rightarrow \sigma_{i-i}(\psi[a_i]) \trianglerighteq 0.$$

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

$$v' \leftrightarrow \sigma_k(v), \qquad w' = \sigma_k(w).$$

Example 2.3. *Assume the pattern $\prec$ is as shown in Eq. 2.5, i.e.,*

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

In this case,

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

$$\begin{aligned} \texttt{lre} = 0 \vee \texttt{lre} = 1, \qquad & \texttt{rle} = 0 \vee \texttt{rle} = 1, \\ \texttt{conn} = 0 \vee \texttt{conn} = 1, \qquad & \texttt{disc} = 0 \vee \texttt{disc} = 1. \end{aligned}$$

2. $\mathrm{pre}^{\prec}(A)$ *is equivalent to*

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

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

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

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

$\Pi^{\prec}$ *is the conjunction of the above formulas together with the formulas encoding the initial and goal states.* $\Pi^{\prec}$ *is satisfiable, and the plan in Eq. 2.2 corresponds to a model of* $\Pi^{\prec}$.

Indeed, if Π is the domain in the example and COMPUTEPATTERN(Π) in the SPP(Π) procedure in Figure 2.1 returns the complete pattern in Eq. 2.5, SPP(Π) will return a plan when $n = 1$, i.e., at the first iteration in which $\prec$ is not empty.

Theorem 2.3. *Let* Π *be a numeric planning problem. Let* $\prec$ *be a pattern. The* SPP $\prec$*-encoding* $\Pi^{\prec}$ *is correct and complete.*

Proof. Let $\prec = a_1; a_2; \ldots; a_k$, $k \geq 0$. For $\Pi = \langle V_B, V_N, A, I, G\rangle$, let $\Pi_\emptyset$ be the numeric planning problem Π 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 μ is a model of $\Pi_\emptyset^{\prec}$ then $\pi = a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \ldots; a_k^{\mu(a_k)}$ is a plan of $\Pi_\emptyset$ and $s_k = res(\pi, I)$ is such that, for each state variable $v \in V_B \cup V_N$, $s_k(v) = \mu(\sigma_k(v)) = \mu(v')$. 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 $\Pi_\emptyset^{\prec}$ reduces to

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

If $k = i+1 > 0$, let $\pi_i = a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \ldots; a_i^{\mu(a_i)}$ 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, both $s_k = res(a_k^{\mu(a_k)}, s_i)$ is defined, and for each state variable $v \in V_B \cup V_N$, $s_k(v) = \mu(\sigma_k(v)) = \mu(v')$, hold 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 2.2. Now consider a model μ of $\Pi^{\prec}$. Then, μ is also a model of $\Pi^{\prec}_{\emptyset}$ and π 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 μ satisfies $\mathcal{G}(\mathcal{X}')$.

Completeness. Given the definition of completeness for the $\prec$-encoding, we have to prove that if Π admits a plan which is a subsequence of $\prec$, then $\Pi^{\prec}$ is satisfiable. Let π be a valid plan of length $n \leq k$ of Π which is also a subsequence of $\prec$. Let s_n be the last state induced by π. Then, if we consider π as a pattern and build Π^{π}, the assignment μ 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 Π^{π}. The proof is by induction on n and analogous to the proof done for correctness, by first considering $\Pi_{\emptyset}$. Then, Π^{π} is equivalent to the formula obtained from $\Pi^{\prec}$ substituting each action variable not in π with 0, and hence $\Pi^{\prec}$ is satisfiable. □

Due to Theorem 2.1 and Theorem 2.3, for any numeric planning problem Π, the SPP(Π) procedure in Figure 2.1 is correct and complete.

2.2.3 Pattern Computation

Consider a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G \rangle$.

Though the SPP(Π) procedure in Algorithm 2.1 is guaranteed to be correct and complete for any complete pattern $\prec_I$ returned by COMPUTEPATTERN(Π), it is clear that its performance may critically depend on $\prec_I$, as shown also by our running example.

Example 2.4. *As already seen, if the pattern $\prec$ is as shown in Eq. 2.5, then $\Pi^{\prec}$ is satisfiable. Thus, if $\prec_I = \prec$, then* SPP(Π) *returns a plan after $n = 1$ concatenations of $\prec_I$. On the other hand, if $\prec_I$ is the sequence obtained reversing Eq. 2.5, i.e.,*

$$\texttt{lftl}; \texttt{rgtr}; \texttt{disc}; \texttt{exch}; \texttt{conn}; \texttt{rgtl}; \texttt{lftr}; \texttt{rle}; \texttt{lre} \tag{2.6}$$

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

Even considering the SPP(Π) procedure in Algorithm 2.1 modified to compute a (possibly) brand new n-complete pattern at each iteration (as discussed at the end of subsection 2.2.1), the problem is how to easily (i.e., in polynomial time) compute a "good" pattern. To address this problem, consider a pattern $\prec = a_1; \ldots; a_k$, $k \geq 0$. Our desiderata are 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 Π' possibly differing from Π 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 2.4. *Let Π be a numeric planning problem. Let $\prec$ be a pattern. Let a be an action in Π. Let $\prec + a$ be a pattern obtained by inserting a in $\prec$. $\prec + a$ strongly dominates $\prec$.*

Proof. Let $\prec = a_1; \ldots; a_k$ and $\prec + a = a_1; \ldots; a_i; a; a_{i+1}; \ldots; a_k$, $0 \leq i \leq k$. $\prec + a$ strongly dominates $\prec$, since each model μ of $\mathcal{T}^{\prec}$ can be extended to a model μ' 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 less 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, $m \geq 0$ and also $m \leq 1$ 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 $\prec'$ and for any state s, $R_s^{\prec} \subseteq R_s^{\prec'}$ if and only if $\prec'$ strongly dominates $\prec$.

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 $m \geq 0$. Indeed, checking whether this condition holds is far from being trivial, since in general it amounts to check the unsatisfiability of the formula

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

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 in which a_i is the first action `lre` in the pattern in Eq. 2.5), 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 a superset of $R_I^{\prec_{i-1}}$ (assuming such superset can be easily computed).

Theorem 2.5. *Let Π be a numeric planning problem. Let $\prec = a_1; \ldots; a_k$ be a pattern, $k \geq 0$. 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 μ of $\Pi^{\prec}$, the assignment μ' differing from μ 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 $R_I^{\prec_{i-1}} = R_I^{\prec_i}$ and hence the thesis.

□

Corollary 2.1. *In the hypothesis of the previous theorem, the pattern*

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

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

Proof. From Theorem 2.5, we can remove a_i from $\prec$ and obtain an equivalent pattern $\prec'$. From Theorem 2.4, adding an action to $\prec'$ leads to a new pattern dominating $\prec'$ and hence also $\prec$. □

Considering the theorem, it is relatively easy to check when the first condition of the Theorem is met. For instance, if we consider the pattern obtained by concatenating the actions in Eq. 2.5 and in Eq. 2.6, 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 [Scala et al., 2016b]. An ARPG is a digraph of alternating state (S_i) and action (A_i) layers, which, starting from the initial state layer, outputs a partition $A_1, \ldots, A_k$ on the set of actions. In state layers, numeric variables are represented as intervals and boolean variables as literals. In the state layer S_0 appear unit intervals (e.g., $[3, 3]$) and literals based on the initial condition. In the action layer A_i appear only actions which preconditions can be achieved by the intervals and literals in S_i. The state layer at S_{i+1} is obtained by extending the intervals and literal of S_i with the effects of all the actions of A_i. For example, if the interval for x in S_i is the unit interval $[3, 3]$ and an action in A_i increases x by 1, then the interval in S_{i+1} becomes $[3, \infty]$ since the action could be repeated multiple times, if instead an action of A_i deletes v, then S_{i+1} will contain $\neg v$. Notice that bot v and $\neg v$ can appear in the same state layer. The layers are extended until a fix point is reached. Since repetitions are not allowed, the construction of the ARPG is guaranteed to reach a fix point at the layer $k+1$ when $A_{k+1} \cap A_k = \emptyset$. If $a \in A_{i+1}$ then any sequence of actions which contains a and which is executable in the initial state, contains at least an action $b \in A_i$ ($0 \leq i < k$). In the computed pattern, a precedes b if $a \in A_i$ and $b \in A_j$ with $1 \leq i < j \leq k$, while actions in the same partition are arbitrarily ordered.

We will denote the superset of $R_I^{\prec i-1}$ computed with ARPG with $R_{\text{ARPG}}^{\prec i-1}$. Here we convey the basic ideas of the ARPG and properties by considering our running example, which will be used also to show an application of the first condition of the Theorem.

Example 2.5. *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,

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

given the i-th action a_i whose preconditions are satisfied by some relaxed state in $R^{\prec_i}_{\text{ARPG}}$, $R^{\prec_i;a_i}_{\text{ARPG}}$ 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 $R_I^{\texttt{lre};\texttt{rle};\texttt{lftr};\texttt{rgtl}}$ representing the set of states reachable with the initial pattern $\texttt{lre};\texttt{rle};\texttt{lftr};\texttt{rgtl}$ of the pattern in Eq. 2.5, the corresponding superset $R^{\texttt{lre};\texttt{rle};\texttt{lftr};\texttt{rgtl}}_{\text{ARPG}}$ 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 [Scala et al., 2016b] for more details. Thus, assuming $\prec$ is as shown in 2.5, 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 a 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 as shown in Eq. 2.6, then the set $R^{\texttt{lftl};\texttt{rgtr}}_{\text{ARPG}}$, representing the superset of the states reachable from $R^{\epsilon}_{\text{ARPG}}$ by executing the first 2 actions in the pattern of Eq. 2.6, 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 $\texttt{disc}$ is not executable in any state represented by $S^{\texttt{lftl};\texttt{rgtr}}_{\text{ARPG}}$ (and thus $R^{\texttt{lftl};\texttt{rgtr};\texttt{disc}}_{\text{ARPG}} = R^{\texttt{lftl};\texttt{rgtr}}_{\text{ARPG}}$), and similarly for the actions $\texttt{exch}$, and $\texttt{conn}$. Thus, we can remove such actions from Eq. 2.6 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:*

$$\begin{array}{ll} level\ 0: & \texttt{lftl}, \texttt{rgtl}, \texttt{lftr}, \texttt{rgtr}, \texttt{rle}, \texttt{lre}, \\ level\ 1: & \texttt{conn}, \\ level\ 2: & \texttt{disc}, \texttt{exch}, \end{array} \tag{2.7}$$

corresponding to a partial order of 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.

Improving Patterns by Swapping Action Occurrences

Consider a pattern $\prec' = a_1; \ldots; a_{i-1}; a_{i+1}; a_i; a_{i+2}; \ldots; a_k$, $0 \leq i \leq k$, 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, $m \geq 1$, (resp. a_{i+1}^n, $n \geq 1$) 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) $x := e$ is also a simple assignment of a';

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 2.6. *Let* Π *be a numeric planning problem. Let* $\prec = a_1; \ldots; a_k$ *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 $m, n \geq 0$,

1. $res(a_{i+1}^n, res(a_i^m, s))$ is defined if and only if both a_i^m and a_{i+1}^n are executable in s,

2. $res(a_i^m, res(a_{i+1}^n, s))$ is defined if and only if both a_i^m and a_{i+1}^n are executable in s,

3. $res(a_i^m, res(a_{i+1}^n, s)) = res(a_{i+1}^n, res(a_i^m, s))$ if both a_i^m and a_{i+1}^n are executable in 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 a_i blocks a_{i+1} or 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 $\mathtt{eff}(a)$, 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 $v := e \in \mathtt{eff}(a)$.

The above definitions of an action blocking/supporting another action are similar to the notion of disabling/enabling action in [Wehrle and Rintanen, 2007] in the classical setting. In particular, when restricting to the classical case, our notion of blocking is equivalent to the notion of disabling, while if a supports a' our definition is stricter since it entails that a does not block a', a condition which does not necessarily hold if a enables a', see [Wehrle and Rintanen, 2007]. 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 and a_{i+1} do not mutually interfere in their effects, then $\prec'$ strongly dominates $\prec$.

Theorem 2.7. *Let Π be a numeric planning problem. Let $\prec = a_1; \ldots; a_k$ 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 and a_{i+1} do not mutually interfere in their effects. 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. In the hypotheses of the theorem, 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 2.6. *According to our definitions and considering the sets of actions at each level of the* ARPG *as in Eq. 2.7*

1. *for level 0, each pair of distinct actions at this level do not mutually interfere except for the pairs* $\{\texttt{lftl}, \texttt{lftr}\}, \{\texttt{rgtl}, \texttt{rgtr}\}, \{\texttt{lre}, \texttt{rle}\}$. *Further, no action at this level blocks or supports 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 2.7:* `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 2.7 allows us to conclude that the pattern* `conn`; `exch`; `disc` *strongly dominates the other 5 orderings.*

2.2.4 Plan Quality

Thanks to Theorem 2.3, we know that any model μ of $\Pi^{\prec}$ corresponds to the valid plan $\pi = a_1^{\mu(a_1)}; a_2^{\mu(a_2)}; \ldots; a_k^{\mu(a_k)}$. However, the discovered plan may include redundant actions.

Example 2.7. *Assume, as in Example 2.1, 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 2.7, the pattern* $\prec_I$ *returned by* COMPUTEPATTERN(Π) *in Algorithm 2.1 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(Π) *will determine the existence of a plan after two concatenations of the above pattern, i.e., with*

$$\begin{aligned} \prec = \ & \texttt{lftl}1; \texttt{rgtl}1; \texttt{lftr}1; \texttt{rgtr}1; \texttt{rle}1; \texttt{lre}1; \texttt{conn}1; \texttt{exch}1; \texttt{disc}1; \\ & \texttt{lftl}2; \texttt{rgtl}2; \texttt{lftr}2; \texttt{rgtr}2; \texttt{rle}2; \texttt{lre}2; \texttt{conn}2; \texttt{exch}2; \texttt{disc}2. \end{aligned}$$

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

$$\begin{gathered} \mu(\texttt{lftl}1) = k, \qquad \mu(\texttt{rgtl}1) = k + X_I, \\ \mu(\texttt{lftr}1) = X_I, \qquad \mu(\texttt{rgtr}1) = 0, \\ \mu(\texttt{rle}1) = m, \qquad \mu(\texttt{lre}1) = n, \\ \mu(\texttt{conn}1) = 1, \qquad \mu(\texttt{exch}1) = Q, \qquad \mu(\texttt{disc}1) = 1, \\ \mu(\texttt{lftl}2) = p + X_I, \qquad \mu(\texttt{rgtl}2) = p, \\ \mu(\texttt{lftr}2) = 0, \qquad \mu(\texttt{rgtr}2) = X, \\ \mu(\texttt{rle}2) = q, \qquad \mu(\texttt{lre}2) = r, \\ \mu(\texttt{conn}2) = \mu(\texttt{exch}2) = \mu(\texttt{disc}2) = 0, \end{gathered}$$

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 (*$\mu(\texttt{lftl}1) = k$*) and then to the right for* $k + X$ *times (*$\mu(\texttt{rgtl}1) = k + X_I$*) to reach the origin,*

2. *having the right robot going directly to the origin ($\mu(\texttt{lftr}1) = X_I$, $\mu(\texttt{rgtr}1) = 0$),*
3. *possibly enabling the right-to-left exchange ($\mu(\texttt{rle}1) = m \in [0, 1]$) and then surely enabling the left-to-right exchange ($\mu(\texttt{lre}1) = 1$) when $\mu(\texttt{rle}1) = 1$,*
4. *connecting, exchanging Q objects and disconnecting ($\mu(\texttt{conn}1) = 1$, $\mu(\texttt{exch}1) = Q$, $\mu(\texttt{disc}1) = 1$),*
5. *having the left robot going to the left for $p+X_I$ times ($\mu(\texttt{lftl}2) = p+X_I$) and then to the right for p times ($\mu(\texttt{rgtl}2) = p$) to reach the position it originally had,*
6. *having the right robot go directly to its original position ($\mu(\texttt{lftr}2) = 0$, $\mu(\texttt{rgtr}2) = X_I$),*
7. *enable or not the left-to-right and/or the right-to-left exchange ($\mu(\texttt{rle}2) = q$, $\mu(\texttt{lre}2) = r$).*

In such plans, some actions can be executed even if unnecessary (e.g., $\texttt{lftl}1$, $\texttt{lre}$) or can be executed more times than necessary (e.g., $\texttt{lftl}1$). 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 $\textsc{Solve}(\Pi^{\prec})$ 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 $\texttt{lftl}$ useless action occurs before the $\texttt{exch}$ action, and (ii) no useless $\texttt{rgtl}$ action occurs after the $\texttt{exch}$ action.*

Analogously, if in $\textsc{SPP}(\Pi)$ 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 $\textsc{Solve}(\Pi^{\prec})$ becomes satisfiable, 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(Π) 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 μ whose corresponding plan π is minimal according to some criteria. In the literature, the following minimality conditions have been defined and studied, especially in the classical setting (see, e.g., [Bercher et al., 2024] for a recent survey on the topic):

1. $\prec$-*minimality*: there should not be another model μ' whose corresponding plan has fewer actions than μ,

2. π-*minimality*: there should not be another model μ' whose corresponding plan is a subsequence of π.

Any model satisfying one of the above two conditions corresponds to an non-redundant plan π: removing some actions in π leads to an invalid plan. Though returning an non-redundant plan may be a desirable property, it comes with an extra price, since it is well known that checking whether a plan is non-redundant is already co-NP-hard in the classical setting with no numeric variables (see, e.g., [Bercher et al., 2024]).

Indeed, checking non-redundancy amounts to verifying the non-existence of a shorter plan. In our case, a model μ with the corresponding plan π is $\prec$-optimal if it minimizes $\sum_{i=1}^{k} a_i$, and is π-optimal if it minimizes $\sum_{i=1}^{k} a_i$ and also satisfies $a \leq \mu(a)$, for each action a.

It is thus possible to find a $\prec$-optimal and/or π-optimal plans if the SMT solver also supports the minimization of $\sum_{i=1}^{k} a_i$, as, e.g., `Z3` v4.12.2 [de Moura and Bjørner, 2008], does. Other solutions to improve the quality of the returned plan are possible. Bofill et al. (2016) propose (i) to call a standard SMT solver for finding an initial model μ of $\Pi^{\prec}$, and then (ii) call a MaxSMT solver on the problem $\Pi^{\mu} \cup \{a_i = 0 : \mu(a_i) = 0, i \in [1, k]\}$ 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 μ 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 [Cimatti et al., 2013], offer native support for specifying the order in which the heuristic should operate. Both these methods allow computing a non-redundant plan, assuming rolling actions is not possible.

In any case, while these methods may reduce the number of executed actions, they do not guarantee the return of an optimal plan, i.e., of the shortest possible plan of Π. Indeed, such an optimal plan may not be a subsequence of the selected $\prec$ and thus it may not correspond to a model of $\Pi^{\prec}$.

2.3 Relation to Planning as Satisfiability Encodings

Let $\Pi = \langle V_B, V_N, A, I, G \rangle$ be a numeric planning problem.

As outlined in Chapter 1, 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., [Kautz and Selman, 1992]. 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 present the rolled-up and standard encodings (subsection 2.3.1), the $R^2\exists$ encoding (subsection 2.3.2), and how they are related to our pattern encoding when used in the planning as satisfiability framework (subsection 2.3.3).

2.3.1 Rolled-up and Standard Encodings

In the state-of-the-art *rolled-up encoding* Π^R *of* Π proposed in [Scala et al., 2016d], 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.[5] Then, the symbolic transition relation $\mathcal{T}^R(\mathcal{X}, \mathcal{A}, \mathcal{X}')$ of Π^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 $\mathrm{pre}(a)$,

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

where $\psi[a]$ is the linear expression obtained from ψ 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) ψ_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 [Scala et al., 2016d]).

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 $w \in V_N$,

$$\left(\bigwedge_{a : v := \top \in \mathrm{eff}(a)} a = 0 \wedge \bigwedge_{a : v := \bot \in \mathrm{eff}(a)} a = 0\right) \rightarrow v' \equiv v, R$$
$$\bigwedge_{a : w := \psi \in \mathrm{eff}(a)} a = 0 \rightarrow w' = w.$$

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

[5] To ease the presentation, our definition of Π^R considers just the cases $\alpha = 0$ and $\alpha = 1$ of Theorem 1 in [Scala et al., 2016d], as we did in the previous section.

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 $\mathrm{eff}(a2)$.[6]

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 Π^R reduces to the standard encoding as defined, e.g., in [Leofante et al., 2020]. Equivalently, in the *standard encoding* Π^S *of* Π, 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 μ 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 [Scala et al., 2016d].

Theorem 2.8 (Scala et al. (2016d))**.** *Let Π be a numeric planning problem. The planning as satisfiability rolled-up encoding Π^R and the standard encoding Π^S are both correct and complete.*

2.3.2 Relaxed-Relaxed $\exists$ ($R^2\exists$) 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., [Rintanen et al., 2006]. Indeed, allowing to set more actions to a value > 0 while maintaining correctness and completeness of the encoding, allows finding solutions to Eq. 1.2 with a lower value for the bound. Several proposals along these lines have been made. Here we present

[6] Notice that if two actions are in mutex, then they are also in mutual exclusion, 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 may allow for models not corresponding to valid plans.

the $R^2\exists$ encoding presented in [Bofill et al., 2017] which is arguably the state-of-the-art encoding in which actions are encoded as Boolean variables (though there exist cases in which the $\exists$-encoding presented in [Rintanen et al., 2006] allows solving Eq. 1.2 with a value for the bound lower than the one needed by the $R^2\exists$ encoding).

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 a simple and complete pattern.

Consider a simple and complete pattern $\prec = a_1; a_2; \ldots; a_k$, $k \geq 0$. We denote the $R^2\exists$ $\prec$*-encoding of* Π 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 ψ 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 $w \in V_N$,

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

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 μ 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 Π. 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 Π is correct and complete.

Theorem 2.9 (Bofill et al. (2017))**.** *Let Π be a numeric planning problem. Let $\prec$ be a simple and complete pattern. The planning as satisfiability $R^2\exists$ $\prec$-encoding $\Pi^{R^2\exists,\prec}$ is correct and complete.*

2.3.3 Relationships Among the Standard, Rolled-up, Relaxed-Relaxed Exists and Pattern Encodings

Consider a simple and complete pattern $\prec$. Since $\prec$ is simple 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 in Eq. 1.1 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 subsection 2.2.2, 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 2.10. *Let Π be a numeric planning problem. Let $\prec$ be a simple 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 2.3 does not hold for any problem Π. □

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

1. the multiple consecutive execution of the same action as in the Π^{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 Π^{R} and $\Pi^{R^2\exists,\prec}$, and the latter two dominate the standard encoding Π^{S}. Given two planning as satisfiability encoding 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_1 will find a plan with a lower bound than E_2.

Theorem 2.11. *Let Π be a numeric planning problem. Let $\prec$ be a simple and complete pattern. The planning as satisfiability* SPP *$\prec$-encoding $\Pi^{S,\prec}$ dominates the rolled-up encoding Π^{R} and the $R^2\exists$ $\prec$-encoding $\Pi^{R^2\exists,\prec}$. Both Π^{R} and $\Pi^{R^2\exists,\prec}$ dominate the standard encoding Π^{S}.*

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

$\Pi^{S,\prec}$ dominates Π^{R}. We have to prove that, for any bound n, if Π_n^{R} is satisfiable then also $\Pi_n^{S,\prec}$ is satisfiable, which follows from the fact that any model μ 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 μ be a model of $\mathcal{T}^{R}(\mathcal{X},\mathcal{A},\mathcal{X}')$ and α be the sequence of actions corresponding to the model μ. Clearly, α is a valid plan for the planning problem $\Pi_\mu = \langle V_B, V_N, A, I_\mu, G_\mu\rangle$ in which I_μ is the restriction of μ 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 μ to the variables in $\mathcal{X} = V_B \cup V_N$ and $\mathcal{X}'$. From the completeness of the pattern encoding, the pattern α-encoding of Π_μ is satisfiable. Then, also the SPP $\prec$-encoding of Π_μ is satisfiable since:

1. any two actions in α do not mutually interfere, and thus we can reorder the actions in α as to respect the ordering in $\prec$ (Theorem 2.7), and

2. for each action $a \notin \alpha$, $\mu(a) = 0$.

$\Pi^{S,\prec}$ dominates $\Pi^{R^2\exists,\prec}$. As in the previous case, we prove that any model μ 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 μ be a model of $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X},\mathcal{A},\mathcal{X}')$ and α be the sequence of actions corresponding to the model μ. The sequence α is a subsequence of $\prec$ and thus, by the completeness of the SPP $\prec$-encoding, is also a model of $\mathcal{T}^{\prec}(\mathcal{X},\mathcal{A},\mathcal{X}')$.

Π^{R} dominates Π^{S}. The fact that Π^{R} dominates Π^{S} follows from the monotonicity of first order logic: the formulas in Π^{S} are a subset of the formulas in Π^{R}, and thus if Π^{S} is satisfiable, so Π^{R} is.

$\Pi^{R^2\exists,\prec}$ dominates Π^{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 μ be a model of $\mathcal{T}^{S}(\mathcal{X},\mathcal{A},\mathcal{X}')$. Because of the effect and mutex axioms in Π^{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 μ 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 2.8. *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 in Eq. 2.2, $\Pi^{S,\prec}_n$ is satisfiable when $n = n_{S,\prec} = 1 < n_R$, while $\Pi^{R^2\exists,\prec}_n$ 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 $n_{R^2\exists,\prec} \leq n_R$ if and only if $2(X_I - 1) + Q \leq 5$. If actions in $\prec$ are not ordered as in the plan in Eq. 2.2, 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 Π^{R} (resp. Π^{S}), and this happens when actions in $\prec$ are in reverse order wrt the plan in Eq. 2.2.*

	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	-	-
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
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	**273**	**267**	**256**	**249**	**227**	**83**	**79**	**133**	**2**	**0**	**248**	**188**	**156**	**115**	**92**

Table 2.1 Comparative analysis between PATTY_A, PATTY_E and PATTY_R. PATTY_R^{min}, PATTY_R^{med}, and PATTY_R^{max} results indicate the performance that can be expected in the best, median, and worst case, when using a randomly generated pattern. Each domain is labelled 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 Taitler et al. [2024] for more details. A "-" indicates that no problem in the domain has been solved with the given resources. Best results are in bold

2.4 Implementation and Experimental Analysis

In this section, we first experimentally analyse the performance of the basic procedure in Algorithm 2.1 when

1. exploiting the pattern selection procedure of the ARPG enhanced with the results presented in subsection 2.2.3 (subsection 2.4.1), and

2. implementing the strategies presented in subsection 2.2.4 to return higher quality plans (subsection 2.4.2).

We end the section with a comparative analysis with all the publicly available state-of-the-art symbolic (subsection 2.4.3) and search-based numeric planners (subsection 2.4.4).

For the experiments, we considered all the domains and problems of the 2023 Numeric International Planning Competition (IPC) [Taitler et al., 2024].

We also considered the LINEEXCHANGE domain, which generalizes the domain in the example 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 $Q \in \mathbb{N}$ items and the goal is to transfer all the items to the last robot in the line.

We then considered the same settings used in the Agile Track of the IPC, and thus with a time limit of 5 minutes. Analyses have been run on an Intel Xeon Platinum 8000 3.1GHz with 8 GB of RAM. We performed some experiments with a time-limit of 30 minutes and obtained the same qualitative results. All the symbolic planners have been run using Z3 v4.12.2 [de Moura and Bjørner, 2008] for checking the satisfiability of the formula in Eq. 1.2, represented as a set of assertions in the SMTLIB format [Barrett et al., 2016]. In the tables, we show the results only for those domains for which at least one planner was able to solve a problem with the given resources.

All our systems have been implemented as part of the PATTY system, and are publicly available at https://github.com/matteocarde/patty, together with the LINEEXCHANGE domain and the problems used in this chapter.

2.4.1 Impact of the Computing Pattern Procedure

Consider a numeric planning problem Π. As already discussed in the previous sections, how the pattern is selected can have a dramatic impact on the number n of iterations in the SPP procedure needed to find a plan, thus on the number of calls to the SMT, and ultimately on performance. Assuming the existence of a plan of length n, the SPP procedure in Algorithm 2.1 needs from 1 to n iterations before finding it, how many depending on the characteristics of the planning problem and of the selected pattern.

We saw previously how the pattern can be selected by exploiting the ARPG construction informally presented in subsection 2.2.3. Here we extend the system PATTY_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 PATTY_E. Additionally, in both PATTY_A and PATTY_E, if we do not have an ordering relation between

two actions a_1 and a_2, they are lexicographically ordered in the respective patterns.

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 2.1 summarizes the results. In the sub-tables/columns, we show:

1. the name of the domain (sub-table Domain);
2. the number of problems solved (sub-table Solved);
3. the average time needed to find a solution, counting the time limit when the solution could not be found (sub-table Time),
4. the average number of calls to the SMT solver needed to find a solution, computed considering only the problems solved by all the planners able to solve at least one problem in the domain (sub-table SMT calls), and
5. on how many of the 380 problems, a system obtained the best result (last row Best).

Considering the SPP procedure in Algorithm 2.1, 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 PATTY_{E} dominates PATTY_{A}, as the latter never exhibits a lower number of calls to the SMT solver than the former. Further, the enhanced pattern computation of

PATTY$_{\text{E}}$ produces some effects on 6 out of 12 domains whose problems required more than one call to the SMT solver. Still, although PATTY$_{\text{E}}$ dominates PATTY$_{\text{A}}$, the latter solves more problems in two domains (balanced by 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 PATTY$_{\text{A}}$ encoding while it fails on PATTY$_{\text{E}}$ encoding.

Considering also the performance of PATTY$_{\text{R}}^{min}$, PATTY$_{\text{R}}^{med}$ and 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, these problems, on average, require more than 5 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 exploited by PATTY$_{\text{I}}$ causes an increment in the number of solved problems of the 7%/10%/20% wrt PATTY$_{\text{R}}^{min}$/ PATTY$_{\text{R}}^{med}$/ PATTY$_{\text{R}}^{max}$. Not surprisingly, PATTY$_{\text{R}}^{min}$ has the best performance on most problems: indeed, PATTY$_{\text{R}}^{min}$ has by construction the best results on each problem out of 5 runs, even if with a randomly generated pattern.

2.4.2 Quality of the Computed Plan

As discussed in subsection 2.2.4, it is indeed possible for the returned plan to contain redundant actions. To assess the extent of this issue, we will compare PATTY$_{\text{E}}$ with:

	Solved (out of 20)			Time (s)			Plan length		
Domain	$\mathrm{P_E}$	$\mathrm{P_M}$	$\mathrm{P_I}$	$\mathrm{P_E}$	$\mathrm{P_M}$	$\mathrm{P_I}$	$\mathrm{P_E}$	$\mathrm{P_M}$	$\mathrm{P_I}$
BLGRP (S)	**20**	**20**	**20**	**1.8**	32.1	2.4	602	**216**	266
CNT (S)	**20**	19	**20**	**0.9**	34.4	1.3	533	**352**	408
CNT (L)	**20**	11	14	**1.0**	144.6	140.1	52	**26**	31
DEL (S)	**5**	3	**5**	**226.4**	255.4	227.2	19	**16**	17
DRN (S)	**3**	**3**	**3**	**255.3**	256.5	**255.3**	25	**12**	**12**
EXP (S)	2	**3**	2	270.2	**255.7**	270.2	38	**28**	**28**
FARM (S)	**20**	12	19	**2.4**	154.8	49.5	740	**275**	339
FARM (L)	**20**	4	15	**2.7**	265.0	108.0	412	**16**	178
HPWR (S)	**20**	4	18	**9.4**	269.8	54.5	56	**32**	38
MPRIME (S)	**12**	8	11	**137.7**	194.5	144.1	52	**7**	8
PATHM (S)	**18**	6	6	**42.3**	237.8	215.5	684	**124**	166
PLWAT (S)	**6**	4	**6**	223.3	257.4	**217.4**	301	**151**	196
RVR (S)	**15**	7	**15**	101.4	197.2	**95.6**	58	**14**	15
SAIL (S)	**20**	8	15	**3.6**	208.0	135.2	932	**435**	760
SAIL (L)	**19**	8	**19**	**16.2**	225.6	44.5	343	**63**	201
SGR (S)	**20**	10	**20**	10.3	177.0	**9.9**	44	**20**	25
TPP (L)	**2**	**2**	**2**	**270.2**	272.3	**270.2**	13	**8**	10
ZENO (S)	**11**	10	**11**	**136.4**	155.3	136.6	22	**15**	18
LINE (L)	**20**	**20**	**20**	**1.2**	6.1	1.7	383	**315**	**315**
Best	**273**	**162**	**241**	**233**	**16**	**29**	**41**	**162**	**141**

Table 2.2 Comparative analysis between $\mathrm{PATTY_E}$, $\mathrm{PATTY_M}$ and $\mathrm{PATTY_I}$. 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 Taitler et al. [2024] for more details. Best results are in bold.

1. $\mathrm{PATTY_M}$, i.e., $\mathrm{PATTY_E}$ where the solver is instructed to return a solution that minimizes $\sum_{i=1}^{k} a_i$,

2. $\mathrm{PATTY_I}$, i.e., $\mathrm{PATTY_E}$ where the first computed model μ is used to find an improved model that minimizes $\sum_{i=1}^{k} a_i$ while also satisfying $\wedge_{i=1}^{k} a_i \leq \mu(a_i)$.

$\mathrm{PATTY_M}$ and $\mathrm{PATTY_I}$ are thus guaranteed to return a $\prec$-optimal and π-optimal model, respectively, as discussed in subsection 2.2.4. The results are in Table 2.2.

The table shows the number of solved problems, time, and average length of the returned plan (sub-table Plan length), the latter computed considering only the problems solved by all the considered planners. Regarding the

	Solved (out of 20)				Time (s)				SMT calls				Plan length				Variables				Assertions			
Domain	P_E	$R^2\exists$	OMT	SR	P_E	$R^2\exists$	OMT	SR	P_E	$R^2\exists$	OMT	SR	P_E	$R^2\exists$	OMT	SR	P_E	$R^2\exists$	OMT	SR	P_E	$R^2\exists$	OMT	SR
BLGRP (S)	**20**	17	2	**20**	1.8	83.4	270.2	**1.6**	**1.0**	6.0	8.5	**1.0**	124	**22**	60	74	**40**	1.4k	265	**40**	**101**	1.7k	776	122
CNT (S)	**20**	11	18	**20**	**0.9**	169.2	92.7	**0.9**	**1.0**	13.3	13.3	**1.0**	164	125	163	**112**	**45**	7.6k	619	**45**	**114**	8.9k	1.8k	137
CNT (L)	**20**	4	3	6	**1.0**	240.3	255.3	227.6	**1.0**	1.7	5.3	2.7	10	8	**5**	7	**26**	208	122	60	**58**	276	1.5k	297
DEL (S)	**5**	1	1	-	**226.4**	285.7	295.6	-	**1.0**	2.0	10.0	-	**10**	**10**	**10**	-	**250**	15k	1.9k	-	**662**	16k	246k	-
DRN (S)	**3**	**3**	**3**	**3**	**255.3**	259.0	256.3	256.3	**5.7**	8.3	12.3	9.7	25	14	**12**	15	**142**	1.3k	299	232	**344**	1.6k	5.1k	2.4k
EXP (S)	**2**	-	**2**	-	**270.2**	-	276.2	-	**3.0**	-	16.0	-	38	-	**28**	-	**254**	-	1.1k	-	**612**	-	35k	-
FARM (S)	**20**	-	-	**20**	2.4	-	-	**1.3**	**1.0**	-	-	2.2	786	-	-	**334**	**63**	-	-	134	**120**	-	-	1.3k
FARM (L)	**20**	2	1	-	**2.7**	270.2	286.2	-	**1.0**	8.0	12.0	-	145	**14**	19	-	**19**	338	112	-	**32**	460	545	-
HPWR (S)	**20**	-	-	-	**9.4**	-	-	-	**1.0**	-	-	-	**93**	-	-	-	**444**	-	-	-	**788**	-	-	-
MPRIME (S)	**12**	5	6	10	**137.7**	233.2	229.6	174.2	**1.2**	2.2	4.2	5.2	54	**7**	8	34	**364**	36k	1.1k	1.4k	**918**	37k	86k	59k
PATHM (S)	**18**	1	3	1	**42.3**	286.1	262.4	286.0	**1.0**	6.0	9.0	3.0	57	**12**	28	57	**186**	19k	986	416	**318**	20k	5.2k	1.4k
PLWAT (S)	**6**	-	-	-	**223.3**	-	-	-	**8.2**	-	-	-	**342**	-	-	-	**476**	-	-	-	**1.3k**	-	-	-
RVR (S)	**15**	8	7	11	**101.4**	202.6	232.6	181.8	**1.4**	2.0	7.7	7.7	70	**16**	17	19	**481**	39k	1.5k	2.0k	**1.1k**	40k	151k	79k
SAIL (S)	**20**	-	-	**20**	**3.6**	-	-	5.5	**3.3**	-	-	7.3	6.1k	-	-	**1.2k**	**135**	-	-	286	**266**	-	-	2.1k
SAIL (L)	**19**	-	1	-	**16.2**	-	285.8	-	**1.0**	-	13.0	-	161	-	**59**	-	**84**	-	874	-	**200**	-	5.8k	-
SGR (S)	**20**	1	18	-	**10.3**	288.2	113.7	-	**2.0**	**2.0**	5.0	-	32	29	**18**	-	**814**	55k	1.7k	-	**2.0k**	56k	92k	-
TPP (L)	**2**	**2**	-	-	**270.2**	271.9	-	-	**2.5**	**2.5**	-	-	13	**10**	-	-	**237**	2.6k	-	-	**604**	3.0k	-	-
ZENO (S)	**11**	9	7	-	**136.4**	174.6	209.6	-	**1.6**	**1.6**	5.3	-	17	16	**13**	-	**241**	7.0k	931	-	**700**	7.4k	74k	-
LINE (L)	**20**	-	-	5	**1.2**	-	-	262.4	**3.0**	-	-	26.0	**110**	-	-	158	**161**	-	-	1.1k	**381**	-	-	4.2k
Best	**273**	**64**	**72**	**116**	**229**	**0**	**5**	**39**	**273**	**17**	**0**	**43**	**138**	**43**	**36**	**63**	**273**	**0**	**0**	**43**	**273**	**0**	**0**	**0**

Table 2.3 Comparative analysis between PATTY_E and other symbolic planners. In the table, names have been abbreviated to save space. See Taitler et al. [2024] for more details. A "-" indicates that no problem in the domain has been solved with the given resources. Best results are in bold.

previous Table 2.1, we do not show the number of calls to the SMT solver since PATTY_E, PATTY_M and PATTY_I use the same pattern.

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

1. the average length of the computed plan is smallest for PATTY_M and highest for PATTY_E,

2. Vice versa, the average number of solved problems is highest for PATTY_E and lowest for PATTY_M for all domains except for Exp (S), where PATTY_M is able to not only solve one more problem than the others, but also prove its optimality,

3. the difference between the time needed by PATTY_E vs PATTY_I varies between being (almost) null (for DRN (S)) and very significant (e.g., CNT (L)). Occasionally, a shorter time for PATTY_I or PATTY_M is reported, which can be attributed to the varying performance of the SMT solver, even when executed on the same instance.

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)).

	Solved (out of 20)				Time (s)				Plan Length			
Domain	P_E	EN	NFD	FF	P_E	EN	NFD	FF	P_E	EN	NFD	FF
BLGRP (S)	**20**	16	-	2	**1.8**	81.5	-	270.2	124	**22**	-	24
CNT (S)	**20**	12	11	15	**0.9**	133.8	149.8	95.7	164	**106**	110	107
CNT (L)	**20**	10	6	8	**1.0**	170.9	214.0	180.0	30	29	16	**13**
DEL (S)	5	13	9	**18**	226.4	121.7	165.2	**41.2**	**25**	31	35	**25**
DRN (S)	3	**16**	**16**	2	255.3	**62.9**	66.0	268.4	16	8	**7**	**7**
EXP (S)	2	**6**	3	-	270.2	**212.3**	253.7	-	**36**	72	54	-
FARM (S)	**20**	**20**	15	9	2.4	**0.9**	85.3	188.1	701	**292**	**292**	341
FARM (L)	**20**	18	11	15	**2.7**	48.6	151.2	80.5	864	34	**21**	34
HPWR (S)	**20**	2	1	1	**9.4**	270.3	285.1	285.0	64	20	35	**16**
MPRIME (S)	12	**17**	14	**17**	137.7	68.1	127.2	**45.1**	63	9	**7**	8
PATHM (S)	**18**	2	1	10	**42.3**	272.2	284.2	154.9	57	18	**12**	14
PLWAT (S)	6	**16**	14	3	223.3	**101.3**	167.2	268.3	**285**	429	393	455
RVR (S)	**15**	8	4	10	**101.4**	197.4	240.8	133.3	34	33	**9**	**9**
SAIL (S)	**20**	**20**	10	1	3.6	**2.0**	150.3	285.0	**174**	**174**	**174**	179
SAIL (L)	**19**	2	15	8	**16.2**	270.6	96.8	182.8	177	81	174	**66**
SGR (S)	**20**	8	4	13	**10.3**	182.5	245.7	122.5	*	*	*	*
TPP (L)	2	**3**	2	2	270.2	**255.2**	270.0	266.7	13	11	**5**	9
ZENO (S)	11	**19**	9	11	136.4	**28.1**	172.5	135.0	20	**14**	21	**14**
LINE (L)	**20**	9	6	6	**1.2**	175.4	235.0	211.6	211	276	234	**187**
Best	**273**	**217**	**151**	**151**	**150**	**49**	**38**	**93**	**90**	**109**	**94**	**100**

Table 2.4 Comparative analysis between PATTY_E and other publicly available search-based planners. In the table, names have been abbreviated to save space. A "-" indicates that no problem in the domain has been solved with the given resources. The "*" in the SUGAR domain, indicates that there was not a single problem solved by all the considered planners. Best results are in bold.

2.4.3 Comparative Analysis with SOTA Symbolic Planners

We compared our planner PATTY_E with the three planning as satisfiability planners SPRINGROLL (based on the rolled-up Π^R encoding [Scala et al., 2016d]), 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 Π^S standard encoding). OMTPLAN is one of the two planners that competed in the last IPC, ranking second. We remind that for the symbolic planners, the number n of calls to the SMT solver is equal

1. to the number of times the initial pattern $\prec_I$ needs to be concatenated to find a plan with our planners, and

2. to the bound needed by the considered planning as satisfiability planners to find a plan.

Table 2.3 presents the results for PATTY_E and all the above-mentioned symbolic planners. The meaning of the labels in the sub-tables is as before.

From the table, two main observations are in order. First, PATTY$_{\text{E}}$ always find 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 2.11). Consequently, PATTY$_{\text{E}}$ produce formulas with (far) fewer variables (in sub-table Variables) and assertions (in sub-table Assertions) than $R^2\exists$, OMTPLAN and SPRINGROLL when the plan is found. The lower number of variables and assertions of PATTY$_{\text{E}}$ is also due to the particular encoding in which no variables representing the intermediate states are used. Interestingly, on some domains PATTY$_{\text{E}}$ good results are due to its action rolling ability (see, e.g, BLGRIP (S), CNT (S)) and in some other domains are due to its ability to allow for sequences of mutually interfering actions (see, e.g, SGR (S), TPP (L), ZENO (S)).

Second, PATTY$_{\text{E}}$ outperforms all the other planners in almost every domain: PATTY$_{\text{E}}$ has always solves more problem and on only two domains it exhibits a longer average solving time.

Interestingly, on at least 50% of the problems that the other planners solve, they can return a shorter plan than PATTY$_{\text{E}}$. However, as discussed in subsection 2.2.4, if rolling is not allowed (as it is for $R^2\exists$ and OMTPLAN), the set of models of the respective encoding is surely finite while this is not necessarily the case for PATTY$_{\text{E}}$, and, when this happens, PATTY$_{\text{E}}$ may return an arbitrarily long plan. Similarly, as the example in subsection 2.2.4 shows, our encoding can have infinitely many models while SPRINGROLL may not, given that the latter does not allow to execute mutually conflicting actions at the same step.

2.4.4 Comparative Analysis with SOTA Search-Based Planners

We compared our planner PATTY$_{\text{E}}$ with the three search-based planners ENHSP [Scala et al., 2016b], METRICFF [Hoffmann, 2003] and NUMERICFASTDOWNWARD (NFD) [Kuroiwa et al., 2022]. NFD competed in the last IPC, ranking first.

The results are reported in Table 2.4. Here, beside the number of solved problems and time, we just report the average length of the computed plan,

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 12 domains, followed by ENHSP in 7 domains. For average plan length, PATTY$_{\mathrm{E}}$ generates shorter plans than the other systems in 4 domains, while ENHSP, NFD, and FF lead in 5, 8, and 8 domains, respectively.

2.4.5 Overall Comparative Analysis

The cactus plot in Figure 2.1 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 but PATTY$_{\mathrm{M}}$, have better performance than the other systems. We remind that PATTY$_{\mathrm{I}}$ and PATTY$_{\mathrm{M}}$ are guaranteed to return a non-redundant plan, and that PATTY$_{\mathrm{M}}$ plan is also guaranteed to have the minimal number of actions among the plans corresponding to $\Pi^{\prec}$ models. The positive results achieved even when considering PATTY$_{\mathrm{R}}^{max}$–representing PATTY's lowest performance across 5 runs per problem with a randomly generated pattern– demonstrate the robust effectiveness of the SPP approach.

Overall, of the 380 problems we considered, PATTY$_{\mathrm{E}}$ successfully solved 273, while ENHSP and SPRINGROLL solved 221 and 116, respectively. ENHSP and SPRINGROLL were the top performers among the search-based and symbolic planners, respectively.

2.5 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

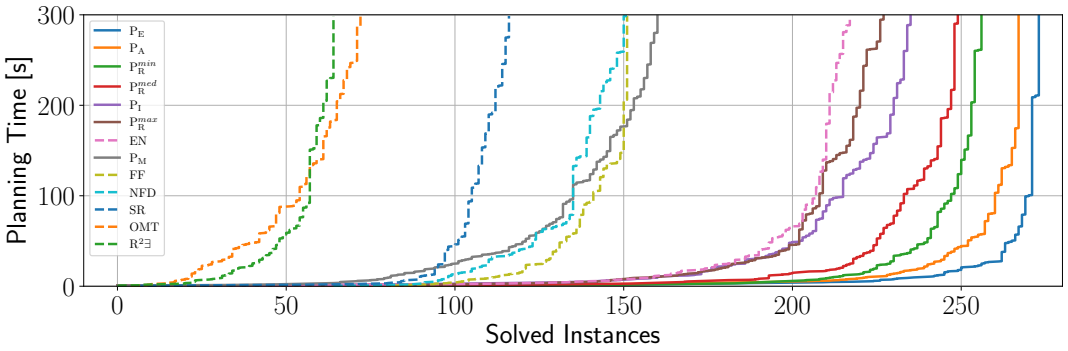

Fig. 2.1 Number of problems solved (x-axis) in a given time (y-axis), by all the presented systems. P_E stands for PATTY_E, and similarly for P_A/P_R^{min}/P_R^{med}/P_R^{max}/P_I/P_M. The different versions of PATTY are represented with solid lines. EN/FF/NFD/SR/OMT/$R^2\exists$ stand for the ENHSP/FF/NFD/SPRINGROLL/OMTPLAN/$R^2\exists$ 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.

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 bound n on the length of the planning as satisfiability encoding.

Experimentally, we considered the basic SPP procedure in which an initial simple 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 last IPC, numeric track and showed that the resulting planner PATTY outperforms all the currently available planners, both symbolic and search-based.

2.5.1 Considerations on Classical Planning

As stated in the preliminary section, a classical planning task can be considered as a numeric planning task $\Pi = \langle V_B, V_N, A, I, G \rangle$ with $V_N = \emptyset$. Thus, our approach can also handle classical planning problems. Classical planning – as the word *classical* suggests – was the first planning flavour to be introduced in 1971 and was the first one in which planners competed back in the original IPC of '98 [McDermott, 2000]. Being also the most simple one, in the last 26 years, the competition has evolved and nowadays classical IPCs present domains which are no longer to be considered classical planning problems, but *lifted* classical planning problems.

In our formulation, we assumed that a planning task to be a tuple $\Pi = \langle V_B, V_N, A, I, G \rangle$ where V_B and V_N are sets of variables. This representation is called *grounded*. A *lifted* representation instead makes use of *objects*, *predicate symbols* and *action symbols* [Lauer et al., 2021]. Let's make an example of two trucks $T = \{t_1, t_2\}$ and three places $P = \{p_1, p_2, p_3\}$. The objects are the trucks and places. A predicate symbol can be, for example, $at(t, p)$, signalling that a generic truck t is at a position p. An action symbol can be, for example, $move(t, a, b)$, representing the action of a generic truck t moving from a generic place a to a generic place b. The predicate symbols and actions symbols are then *grounded* on the objects, meaning that $at(t, p)$ becomes the *ground variables* $\{at(t, p) \mid (t, p) \in T \times P\}$ and $move(t, a, b)$ becomes the *ground actions* $\{move(t, a, b) \mid (t, a, b) \in T \times P \times P\}$. It is clear that a lifted representation is much more compact than a ground one. A *grounder* is a function G that takes as input a lifted representation and outputs a ground representation.

In the last 2023 IPC competition [Taitler et al., 2024], in the classical track, out of the 7 domains which were presented, all of them had compact lifted representation but which results in huge ground representation when grounded. On these domains, we tried the state-of-the-art grounders of ENHSP [Scala and Vallati, 2021] and of the Unified Planning's Grounder[7] and could not ground even the most basic problems in under 10 minutes. It is no surprise then that the podium of the classical track of the 2023 IPC saw as winners only *lifted planners* – i.e., planners able to directly deal with

[7] https://github.com/aiplan4eu/unified-planning

the lifted representation. These solvers are RAGNAROK [Drexler, 2023] and LEVITRON [Corrêa et al., 2023], together with SCORPION [Corrêa et al., 2023] which implemented a very advanced grounder based on GRINGO [Gebser et al., 2011], a grounder for Answer Set Programming [Gelfond and Lifschitz, 1991], which can actually perform some partial solving to better ground the planning task.

For this reason, our solver PATTY could not compete with any of these planners on these domains, since the ground representation is too huge to be able to be encoded into a SMT encoding. In the future, we thus plan to employ some techniques based on planning with *first-order logic* (see, e.g., [Höller and Behnke, 2022]) to instrument PATTY to be a lifted planner, able to directly deal with the lifted representation of a planning task.

Chapter 3

SPP in Classical Planning with Conditional Effects

As stated in Chapter 1, in *classical planning*, the environment in which agents operate is represented only through Boolean variables. In this fragment, actions are *idempotent*, i.e., applying the same action once or multiple times result in the same state. The idempotence property falls if we consider classical planning with *conditional effects* (CEs), i.e., effects of actions which are applied only if some conditions hold. Dealing with CEs has already been extensively studied in the literature and two main approaches exist: either (i) one deals with CEs in a "native" way, i.e., by encapsulating the logic of CEs directly in the procedure searching for a plan [Rintanen, 2011; Röger et al., 2014; Katz, 2019], or (ii) one *compiles* each action with CEs into multiple actions without CEs [Gazen and Knoblock, 1997; Nebel, 2000; Gerevini et al., 2024]. Both approaches have their pros and cons, the first requires a more tailored approach and a specialized solver, since many heuristics (see, e.g., Bonet and Geffner [2001]) cannot directly deal with CEs but it is, in general, faster, while the latter allows the planning task to be solved by any classical planner, but the compilation either causes and exponential blow-up of the number of actions [Gazen and Knoblock, 1997] or a polynomial increase in the length of the plan [Nebel, 2000; Gerevini et al., 2024] (as seen in Chapter 1).

As shown above, dealing with *one* application of an action with CEs has been extensively studied. However, the consecutive repeated execution of the same action with CEs (i.e., *rolling*), has never been previously explored. Rolling has been introduced in [Scala et al., 2016d] for numeric planning, where the lack of idempotence is intrinsic. The numeric planner SPRINGROLL in [Scala et al., 2016d] and this thesis' numeric planner PATTY, employ shortcuts when dealing with rolling of actions with numeric effects (e.g., increasing a variable by 5 for 10 times is equivalent of increasing it by 50 only one time) thus avoiding explicitly considering the multiple repetition of the same action, and speeding the planning process.

A deep connection between classical planning with CEs and (bounded) numeric planning has been theoretically shown by Gigante and Scala [2023] and recapped in Chapter 1, showing that any bounded numeric planning problem can be translated into a classical planning problem with CEs, and vice versa. For this reason, in this chapter, we ask ourselves if rolling can be exploited directly in classical planning with CEs, to speed up planning[1]. Firstly, we substantiate our claim that rolling actions with CEs is a complex and interesting problem by itself: recalling that deciding the existence of a plan for a classical planning task with (resp. without) CEs is PSPACE-complete [Bylander, 1994] (resp. [Nebel, 2000]), in this chapter we show that the problem remains PSPACE-complete even when we consider finding a planning task with only one action with CEs, i.e., deciding how many times the action is consecutively executed is a difficult problem on its own. Secondly, we show an SPP approach, able to deal with multiple execution of the same action with CEs in a single step. In fact, in all the PAS approaches in the literature dealing with CEs (see, e.g., Rintanen [2011]), applying an action k consecutive times requires a bound $n \geq k$, i.e., a step for each application of the action. In our approach, instead, we aim to make the bound n independent of k, or at least orders of magnitude lower. Given a non-idempotent action a, to exploit the rolling of a in a PAS setting, we first compute the *transition relation* of a, denoted with $\mathcal{T}_a$, i.e., a logic formula implicitly representing all the states reachable after *one* application of a, and then we perform the computation

[1] In numeric planning, rolling can be beneficial only in the case where actions have linear effects, due to the monotonicity of the applications of the effects multiple times. Instead, Gigante and Scala's compilation produces, in general, non-linear effects, and thus employing rolling in the compiled planning task is not straight-forward

of the *transitive closure* of $\mathcal{T}_a$ (see, e.g., Matsunaga et al. [1993]) denoted with $\mathcal{T}_a^+$, i.e., a logic formula implicitly representing all the possible states reachable after all possible (finite) applications of a.

Computing the transitive closure for an idempotent action, given our complexity results, is of course a PSPACE-complete problem on itself – otherwise planning with a single non-idempotent action would simply amount to computing the transitive closure and then checking whether the goal state is reachable from the initial state. However, to compute the transitive closure, we employ *Reduced Ordered Binary Decision Diagrams* (simply BDDs) [Bryant, 1985, 1992], which have been very successfully employed for this purpose in the *Model Checking* community [Burch et al., 1992; Clarke et al., 1996] due to their compact representation of propositional formulas and the ability to rapidly perform conjunctions, disjunction, negations, and substitution on these formulas. Even if computing the transitive closure becomes unpractical, its computation procedure produces some intermediate formula $\mathcal{T}_a^m$ with $m \geq 0$ that models the states reachable in *up to* 2^m repetitions of the action a. We can thus impose a time limit on the computation of the transitive closure and then employ the last intermediate formula $\mathcal{T}_a^m$ computed – which in the best case will be $\mathcal{T}_a^m \equiv \mathcal{T}_a^+$ and in the worst case (i.e., $m = 0$) is $\mathcal{T}_a^m \equiv \mathcal{T}_a$ – in a SPP approach, reducing by a factor of 2^m the bound required to find a valid plan.

Finally, given a pattern $\prec$ – as defined in the previous chapter – and a planning task Π we will employ an SPP encoding $\Pi^{\prec+}$, in which, the transition relation $\mathcal{T}_a$ of each non-idempotent action a is substituted with the transitive closure $\mathcal{T}_a^+$, and a propositional variable a^+ is used to represent whether the action a has been executed *at least* one time. After finding, at some bound n, a valid solution for $\Pi_n^{\prec+}$, we can then find a valid plan π for Π, where, for each action a we employ the intermediate formulas $\mathcal{T}_a^1, \ldots, \mathcal{T}_a^+$ found when computing the transitive closure of $\mathcal{T}_a$, and the states before and after the application of a^+ in $\Pi_n^{\prec+}$ to retrieve the number of times a must be repeated in the plan π.

Throughout the chapter, we will employ a motivating example of a counter of two numbers, expressed with binary representation, to illustrate our approach.

3.1 Preliminaries

3.1.1 Classical Planning Task with Conditional Effects

A *classical planning task with Conditional Effects* (in the following, just *planning task*) is a tuple $\Pi = \langle V, A, I, G \rangle$. The set V contains *propositional variables* with domain $\{\top, \bot\}$, the symbols for truth and falsity. The set V induces the set of *literals* $\mathrm{lit}(V) = \{v \mid v \in V\} \cup \{\neg v \mid v \in V\}$. The *negation* $\neg l$ of a literal l is $\neg v$ if $l = v$ or v if $l = \neg v$. Let $L \subseteq \mathrm{lit}(V)$, we can thus define $L^+ = \{v \mid v \in L\}$, $L^- = \{v \mid \neg v \in L\}$, and $\mathrm{vars}(L) = L^+ \cup L^-$. In the following, we will always assume that any subset $L \subseteq \mathrm{lit}(V)$ is *consistent*, i.e. $L^+ \cap L^- = \emptyset$. An *action* $a \in A$ is a tuple $\langle \mathrm{pre}(a), \mathrm{post}(a) \rangle$ where $\mathrm{pre}(a) \subseteq \mathrm{lit}(V)$ are the *preconditions* of a and $\mathrm{post}(a)$ is a set of *conditional effects* of a. A conditional effect e is a tuple $\langle \mathrm{cond}(e), \mathrm{eff}(e) \rangle$ where $\mathrm{cond}(e) \subseteq \mathrm{lit}(V)$ are the conditions which allow applying the effect $\mathrm{eff}(e) \subseteq \mathrm{lit}(V)$. If $v \in \mathrm{eff}(e)$ we say that e *adds* v, if $\neg w \in \mathrm{eff}(e)$ we say that e *deletes* w. We denote with $\mathrm{add}(a) \subseteq V$ and $\mathrm{del}(a) \subseteq V$ the set of all variables added or deleted by the effects of a, i.e.,

$$\mathrm{add}(a) = \bigcup_{e \in \mathrm{post}(a)} \mathrm{eff}(e)^+, \qquad \mathrm{del}(a) = \bigcup_{e \in \mathrm{post}(a)} \mathrm{eff}(e)^-.$$

Let v be a variable, we denote the set of CEs of an action a that adds v as $\Delta_a^+(v) = \{e \in \mathrm{post}(a) \mid v \in \mathrm{eff}(e)^+\}$ and that deletes v as $\Delta_a^-(v) = \{e \in \mathrm{post}(a) \mid v \in \mathrm{eff}(e)^-\}$. Let e_1 and e_2 be two CEs, we say that (i) e_1 *blocks* e_2 if $\mathrm{eff}(e_1)^- \cap \mathrm{cond}(e_2)^+ = \emptyset$ or $\mathrm{eff}(e_1)^+ \cap \mathrm{cond}(e_2)^- = \emptyset$, and (ii) e_1 *allows* e_2 if $\mathrm{eff}(e_1)^+ \cap \mathrm{cond}(e_2)^+ \neq \emptyset$ or $\mathrm{eff}(e_1)^- \cap \mathrm{cond}(e_2)^- \neq \emptyset$. An action a is *non-idempotent* if there exist two CEs $e_1, e_2 \in \mathrm{post}(a)$, $e_1 \neq e_2$, such that e_1 allows and doesn't block e_2. If for each $e \in \mathrm{post}(a)$, we have $\mathrm{cond}(e) = \emptyset$, then a is *without* CEs, and *with* CEs otherwise. If, for each $a \in A$, a is without CEs (resp. with CEs), then also Π is.

A *state* is a set $s \subseteq V$ containing the variables which are true in the state, while all the others are assumed to be false. We denote with $S = 2^V$ the set of all possible states. Finally, we have the *initial state* $I \subseteq V$ and the *goal condition* $G \subseteq \mathrm{lit}(V)$. We say that a state s *respects* a subset of literals L,

denoted with $s \models L$, if $L^+ \subseteq s$ and $L^- \cap s \neq \emptyset$. An action a is *applicable* in a state s if (i) $s \models \mathrm{pre}(a)$, (ii) there exists at least one CE $e \in \mathrm{post}(a)$ such that $s \models \mathrm{cond}(e)$, and (iii) there are no *conflicting* CEs, i.e., for each $v \in V$, $e^+ \in \Delta_a^+(v)$ and $e^- \in \Delta_a^-(v)$ it does not hold that $s \models \mathrm{cond}(e^+) \cup \mathrm{cond}(e^-)$. Applying an action a in a state $s \in S$ results in a state $s' \in S$, denoted as $s' = res(s, a)$, such that s' is undefined if s is undefined or a is not applicable in s, or for each $v \in V$ we have $v \in s'$ iff either (i) $v \in s$ and it does not exist a CE $e \in \Delta_a^-(v)$ such that $s \models \mathrm{cond}(e)$, or (ii) there is at least one CE $e \in \Delta_a^+(v)$ such that $s \models \mathrm{cond}(e)$. Applying a sequence of n actions $\pi = a_0; \ldots; a_{n-1}$ from a state s_0 induces a sequence of $n{+}1$ states $s_0; \ldots; s_n$ where $s_{i+1} = res(s_i, a_i)$ with $i \in [0, n)$, and we say that $s_n = res(s_0, \pi)$. A *valid plan* π for a planning task $\Pi = \langle V, A, I, G \rangle$ is a sequence of actions $a_0; \ldots; a_{n-1}$ such that $s_n = res(I, \pi) \models G$. We denote with $|\pi| = n$ the length of a plan $\pi = a_1; \ldots; a_n$. We denote with a^k, $k \geq 0$ the sequence where a is repeated k times.

Example 3.1 (COUNTERS)**.** *Let's introduce now the motivating example of this chapter. Let $B \in \mathbb{N}$, and let $x, y \in \mathbb{N}$ be two counters bounded by $2^B - 1$ and represented as B-bits binary numbers through the propositional variables $X = \{x_B, \ldots, x_1\}$ and $Y = \{y_B, \ldots, y_1\}$, where x_B and y_B represent the most significant bit. The numbers x and y can be increased and decreased by one unit by changing the respective bits in X and Y accordingly. If a number reaches the value $2^B - 1$, a further increase overflows it to 0 and when it reaches 0 a further decrease underflows it to $2^B - 1$. The two counters can be locked, meaning that the counters cannot change their value any longer, signalled by the variables $L = \{lx, ly\}$. When locking, all pairs of bits x_i and y_i which have the same value are marked as equal by the variable eq_i in $EQ = \{eq_B, \ldots, eq_1\}$. Starting from a predetermined value of the bits, we need to reach a state in which all the bits are equal, i.e. $x = y$. The following problem can be expressed by the planning task $\Pi = \langle V, A, I, G \rangle$ where*

$$
\begin{aligned}
V &= X \cup Y \cup EQ \cup L, \quad I \subseteq X \cup Y, \qquad G = L,\\
A &= \{\mathtt{inx}, \mathtt{iny}, \mathtt{dex}, \mathtt{dey}, \mathtt{lck}\},\\
\mathtt{inx} &= \langle \neg lx, \{\mathtt{ix}_1, \ldots, \mathtt{ix}_B, \mathtt{ox}\}\rangle, \mathtt{dex} = \langle \neg lx, \{\mathtt{dx}_1, \ldots, \mathtt{dx}_B, \mathtt{ux}\}\rangle,\\
\mathtt{iny} &= \langle \neg ly, \{\mathtt{iy}_1, \ldots, \mathtt{iy}_B, \mathtt{oy}\}\rangle, \mathtt{dey} = \langle \neg ly, \{\mathtt{dy}_1, \ldots, \mathtt{dy}_B, \mathtt{uy}\}\rangle,\\
\mathtt{lck} &= \langle \emptyset, \{\mathtt{tt}_1, \mathtt{ff}_1, \ldots, \mathtt{tt}_B, \mathtt{ff}_B, \langle \emptyset, \{lx, ly\}\rangle\}\rangle.
\end{aligned}
$$

and $\mathtt{ix}_i, \mathtt{dx}_i, \mathtt{iy}_i, \mathtt{dy}_i$*, are* CE*s to increase and decrease* x *and* y*,* $\mathtt{ox}, \mathtt{ux}, \mathtt{oy}, \mathtt{uy}$ *to overflow or underflow* x *and* y *when they have reached their bound, and* $\mathtt{tt}_i, \mathtt{ff}_i$ *to lock two bits when both are true (*$\mathtt{tt}$*) or false (*$\mathtt{ff}$*):*

$$\begin{aligned}
\mathtt{ix}_i &= \langle\{\neg x_i, x_{i-1}, \ldots, x_1\}, \{x_i, \neg x_{i-1}, \ldots, \neg x_1\}\rangle,\\
\mathtt{ox} &= \langle\{x_B, \ldots, x_1\}, \{\neg x_B, \ldots, \neg x_1\}\rangle,\\
\mathtt{dx}_i &= \langle\{x_i, \neg x_{i-1}, \ldots, \neg x_1\}, \{\neg x_i, x_{i-1}, \ldots, x_1\}\rangle,\\
\mathtt{ux} &= \langle\{\neg x_B, \ldots, \neg x_1\}, \{x_B, \ldots, x_1\}\rangle,\\
\mathtt{tt}_i &= \langle\{x_i, y_i\}, \{l_i\}\rangle, \qquad \mathtt{ff}_i = \langle\{\neg x_i, \neg y_i\}, \{l_i\}\rangle.
\end{aligned}$$

and accordingly for $\mathtt{iy}_i$*,* $\mathtt{dy}_i$*,* $\mathtt{oy}$ *and* $\mathtt{uy}$*. According to the definition, the actions* $\mathtt{inx}, \mathtt{iny}, \mathtt{dex}$ *and* $\mathtt{dey}$ *are non-idempotent, while* $\mathtt{lck}$ *is idempotent. Let's suppose, with* B *bits, that initially* $x = 0 = 0\ldots0_2$ *and* $y = 2^{B-1} = 10\ldots0_2$*, and thus* $I = \{y_{B-1}\}$ *and* $G = \{l_B, \ldots l_1\}$*. One possible plan is*

$$\pi = \mathtt{inx}^{(2^{B-2})}; \mathtt{dey}^{(2^{B-2})}; \mathtt{lck}$$

3.1.2 Propositional Formulas and Binary Decision Diagrams

A *propositional formula* with n variables is a function $f : \{\top, \bot\}^n \mapsto \{\top, \bot\}$. We say that two propositional formulas f_1 and f_2 are *logically equivalent*, denoted with $f_1 \equiv f_2$, when for each possible assignment to their variables in $\{\top, \bot\}^n$ they produce the same result in $\{\top, \bot\}$. For a propositional formula $f(x_1, \ldots, x_n)$, we denote with

$$\begin{aligned}
f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)|_{x_i \leftarrow \top} &= f(x_1, \ldots, x_{i-1}, \top, x_{i+1}, \ldots, x_n),\\
f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)|_{x_i \leftarrow \bot} &= f(x_1, \ldots, x_{i-1}, \bot, x_{i+1}, \ldots, x_n)
\end{aligned}$$

i.e., the formulas obtained by substituting x_i with either $\top$ or $\bot$ for $i \in [1, n]$. The *existential quantification* of a formula, denoted with $\exists x_i : f(x_1, \ldots, x_n)$, asks whether there exists at least an assignment to x_i (in $\{\top, \bot\}$) such that f is evaluated to $\top$. The existential quantifier can be *eliminated* using the *smoothing operator* S [McGeer, 1989; Lin et al., 1990], i.e.,

$$\mathsf{S}_{x_i} f(x_1, \ldots, x_n) = f|_{x_i \leftarrow \top} \vee f|_{x_i \leftarrow \bot},$$

when the existential quantification is on multiple variables, e.g.,

$$\exists_{x_1,\ldots,x_n} f(x_1,\ldots,x_n),$$

the elimination consists of an iterated application of S, i.e.,

$$\exists_{x_1,\ldots,x_n} f(x_1,\ldots,x_n) \;\leftrightarrow\; \mathsf{S}_{x_1}\cdots\mathsf{S}_{x_n} f(x_1,\ldots,x_n).$$

The iterated repetition $\mathsf{S}_{x_1}\cdots\mathsf{S}_{x_n}$ is denoted with $\mathsf{S}_{x_1,\ldots,x_n}$.

Let f be a propositional formula with n variables $x_1,\ldots,x_n$. A *Binary Decision Diagram* BDD of f [Bryant, 1985, 1992] is a *rooted directed acyclic graph* $\langle V, E, \lambda\rangle$ with V being the *nodes* of the graph, $E \subseteq V \times V$ being the *edges* of the graph and $\lambda : V \mapsto \{1,\ldots,n,\top,\bot\}$ being a *labelling function*. A node $v \in V$ can either be of two types: a *terminal node*, if it is labelled with $\lambda(v) \in \{\top,\bot\}$ or *non-terminal node*, if $\lambda(v) \in \{1,\ldots,n\}$. A non-terminal node v can have two children node, $\mathrm{high}(v), \mathrm{low}(v) \in V$. Intuitively, a non-terminal node v with $\lambda(v) = i$ can be read as a propositional formula recursively defined as

$$f_v = (x_i \wedge f_{high(v)}) \vee (\neg x_i \wedge f_{low(v)}).$$

We say that a BDD is *reduced* when there are no *redundant nodes*, i.e., a node v where $f_{\mathrm{high}(v)} \equiv f_{\mathrm{low}(v)}$. We say that a BDD is *ordered* when we have a total order $<$ between the variables $x_1,\ldots,x_n$ of f such that, for each node v with $\lambda(v) = i$, $\lambda(high(v)) = j$, and $\lambda(low(v)) = k$, we have $x_i < x_j$ and $x_i < x_k$.

Figure 3.1 shows a reduced and ordered BDD for the propositional formula $f = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$. As it can be seen, the structure of the BDD, and most importantly the number of its nodes, is very sensitive to the order given to the variables. Fig 3.1a shows a BDD of f constructed with the order $x_1; x_2; x_3; x_4; x_5; x_6$, having $|V| = 6$, while Fig 3.1b shows a BDD constructed on the same formula f but with the order $x_1; x_3; x_5; x_2; x_5; x_6$, having $|V| = 14$. A reduced and ordered BDD is a *canonical representation* of a propositional formula, meaning that, given two propositional formulas f_1 and f_2 with n variables, the BDDs $\langle V_1, E_1, \lambda_1\rangle$ and $\langle V_2, E_2, \lambda_2\rangle$ constructed with the same order of variables for f_1 and for f_2 are equal – i.e. $V_1 = V_2$, $E_1 = E_2$ and $\lambda_1 = \lambda_2$ – if and only if f_1 and f_2 are logically equivalent. In the rest of the

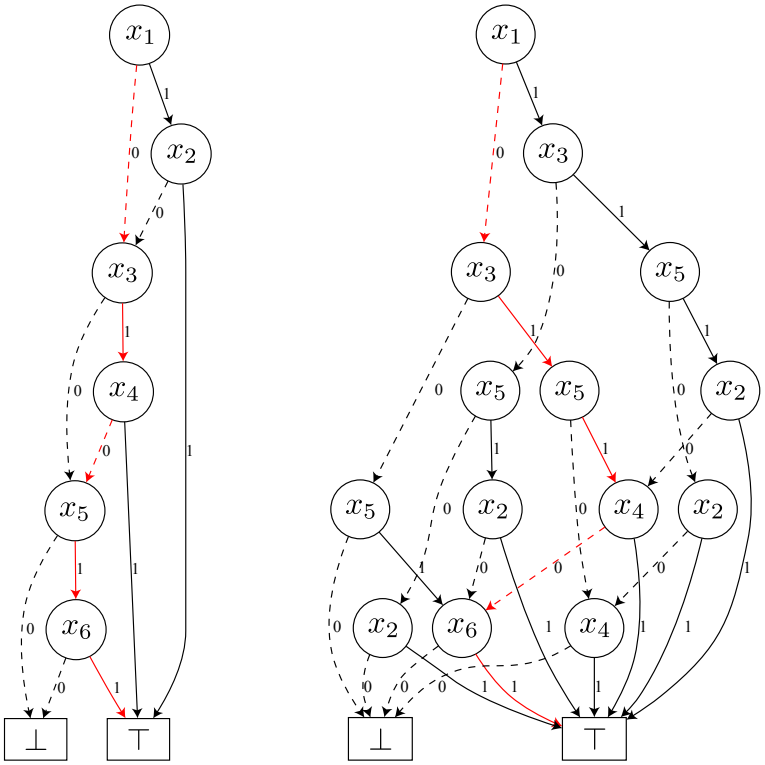

Fig. 3.1 Two BDDs representing the propositional formula $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$ with (a) having the order $x_1; x_2; x_3; x_4; x_5; x_6$ while (b) having the order $x_1; x_3; x_5; x_2; x_4; x_6$. Dashed lines represent the edge $(v, \text{low}(v))$ and solid lines the edge $(v, \text{high}(v))$. Nodes are labelled with the variable they represent. Red edges denote paths from the route to a terminal node v with $\lambda(v) = \top$, i.e., the models $\{\neg x_1, x_2, x_3, \neg x_4, x_5, x_6\}$ and $\{\neg x_1, \neg x_2, x_3, \neg x_4, x_5, x_6\}$.

chapter, we will denote Reduced Ordered Binary Decision Diagrams simply as BDDs.

The use of BDDs makes some operations on propositional formulas very trivial, e.g.,

1. testing equivalence between two propositional formulas amounts to testing whether the BDDs of the two formulas are the same,

2. performing a substitution $f|_{x_i \leftarrow \top}$ (resp. $f|_{x_i \leftarrow \bot}$) amounts to removing from the BDD representation all nodes v with $\lambda(v) = i$ and replacing all edges (w, v) with $(w, \text{high}(v))$ (resp. $(w, \text{low}(v))$),

3. conjunction ($\wedge$), disjunction ($\vee$) or negation ($\neg$) can be performed on the nodes iteratively, e.g., to compute $f = f_1 \wedge f_2$, starting from the routes v_1 and v_2 of f_1 and f_2, respectively, we can iteratively perform $v_1 \wedge v_2$ as $\text{high}(v_1) \wedge \text{high}(v_2)$ and $\text{low}(v_1) \wedge \text{low}(v_2)$, stopping when encountering a terminal node and returning a value based on the semantic of the operator (i.e., $\top \wedge \bot = \bot$),

4. *satisfiability*, i.e., knowing if there is at least one assignment in $\{\top, \bot\}^n$ to the variables of a function f such that the function evaluates to $\top$, can be trivial, since any unsatisfiable formula will have a BDD representation of only a terminal node v with $\lambda(v) = \bot$,

5. if the formula is satisfiable, finding the assignments to the variables of f which satisfy f – i.e., the *models* of f –, simply amounts to finding a path from the root to a terminal node v with $\lambda(v) = \top$, like the red path in Fig. 3.1.

Constructing BDDs which compactly represents a propositional formula depends heavily on the order of the variable, in the worst-case leading to an exponential number of nodes. Finding the optimal order, i.e., the order of variables such that $|V|$ is minimal, is a NP-complete task [Bryant, 1986]. However, in the average case, BDDs can be compact in size, boosting and improving the performances of real-world applications employing propositional formulas, like in Model Checking [Burch et al., 1992].

3.2 Complexity of Rolling

In this section, we present our main novel complexity result for classical planning with CEs. We will prove that determining the plan existence remains PSPACE-complete even when considering a planning task with a single action with CEs.

We recall from Chapter 1 that the problem of determining whether a valid plan exists for a planning task Π with and without CEs is PSPACE-complete [Bylander, 1994; Nebel, 2000].

Theorem 3.1. *Determining whether a valid plan exists for a classical planning task with* CEs $\Pi = \langle V, \{a\}, I, G\rangle$*, where* a *is the only action of* Π*, is* PSPACE*-complete.*

Proof. (Membership) The problem is in PSPACE because the size of a state is bounded by $|V|$ and we can iteratively apply a from I until we reach $s_n \models G$. Since there are at most $2^{|V|}$ possible states, no more than $2^{|V|}$ repetitions of a are required to reach s_n. *(Hardness)*Let M be a *Deterministic Turing Machine* (DTM) with bounded tape, i.e., $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, n\rangle$ (see, e.g., Sipser [1997]). Q is the set of *states,* Σ the *input alphabet,* Γ the *tape alphabet* including Σ and the symbol $\sqcup$ for *"blank"*, δ is a partial function $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$, the states q_0, $q_a \in Q$ are the *initial* and *accepting states,* respectively, $n \in \mathbb{N}$ is the bounded length of the tape. Let the input of the DTM be $y_1; \ldots; y_m$ with $y_i \in \Sigma$ and $m \leq n$. We can reduce the DTM M to the planning task $\Pi = \langle V, \{a\}, I, G\rangle$ such that

$$\begin{aligned} V &= \{tp_i \mid i \in [0, n+1]\} \cup \{at_{i,q} \mid i \in [1, n], q \in Q\} \\ &\quad \cup \{in_{i,x} \mid i \in [1, n], x \in \Gamma\} \cup \{accept\}, \\ I &= \{tp_i \mid i \in [1, n]\} \cup \{at_{1,q_0}\} \\ &\quad \cup \{in_{i,y_i} \mid i \in [1, m]\} \cup \{in_{i,\sqcup} \mid i \in (m, n]\}, \\ G &= \{accept\}. \end{aligned}$$

Where tp_i is used to control the allowed cells of the tape, $at_{i,q}$ signals that the DTM is in tape's cell i and state q, and $in_{i,x}$ signals that cell i contains symbol x. The only action in Π is $a = \langle \emptyset, \mathrm{post}(a)\rangle$ where, for each $i \in [1, n]$ and $(q, x) \in Q \times \Gamma$, if $\delta(q, x) = (q', x', L)$ then $\mathrm{post}(a)$ contains

$$\langle \{at_{i,q}, in_{i,x}, tp_{i-1}\}, \{\neg at_{i,q}, at_{i-1,q'}, \neg in_{i,x}, in_{i,x'}\}\rangle,$$

i.e., if the DTM is in cell i and state q, reading x, and the cell on the left is allowed, then the DTM moves left to cell $i-1$ and to state q' and the symbol in cell i is replaced with x'. Similarly, if $\delta(q, x) = (q', x', R)$ then $\mathrm{post}(a)$ contains

$$\langle \{at_{i,q}, in_{i,x}, tp_{i+1}\}, \{\neg at_{i,q}, at_{i+1,q'}, \neg in_{i,x}, in_{i,x'}\}\rangle.$$

Finally, for each $i \in [1, n]$, $\mathrm{post}(a)$ contains

$$\langle \{at_{i,q_a}\} \cup \{\neg in_{i,x} \mid x \in \Gamma, q' = \sigma(q_a, x)\}, \{accept\} \rangle,$$

i.e., the DTM *accepts* if it is in the accepting state, and it is not reading any symbol that would change the DTM state.

Since the CEs encode the transition δ, a valid plan can be found iff M accepts. Since $|Q| \times |\Gamma| \times n$ is polynomial w.r.t. the size of M, we conclude that a DTM with polynomially bounded tape and its input can be polynomially reduced to a planning task with only one action with CEs. □

This theorem shows how rolling, i.e., the consecutive repetition of an action, even when it is the only action in the planning task, is a difficult problem on its own.

3.3 Rolling Actions with Conditional Effects

In this section, we show how we can exploit rolling when dealing with actions with conditional effects. Firstly, we will describe the concept of *transition relations*, i.e., logic formulas implicitly representing all the states reachable after one application of an action. Then, we will show how it is possible to compute the *transitive closures* of these transitive relations, obtaining a logic formula implicitly representing all the possible states reachable after all possible (finite) applications of the actions.

3.3.1 Transition Functions and Transition Relations

Let $\Pi = \langle V, A, I, G \rangle$ be a planning task and a be an action of Π. The *transition function* of a is a function $T_a : S \times S \mapsto \{\top, \bot\}$, with $S = 2^V$ being the set of all possible states, such that, for each $s, s' \in S$, we have $T_a(s, s') = \top$ iff (i) a is applicable in s, and (ii) s' is the state resulting by the application of a in s, i.e., $s' = res(s, a)$.

As standard for PAS encodings, we model the action's transition function T_a through a *transition relation*. A transition relation is a propositional formula

$\mathcal{T}_a(V, V')$, where V' is a copy of the variables V, i.e., $V' = \{v' \mid v \in V\}$. The purpose of this formula is to implicitly represent all the states $s, s' \in S$ where a is applicable in s, and s' can be reached from s applying a. The variables in V and V' thus model the states s and s', respectively. For the case of classical planning with CEs, the transition relation $\mathcal{T}_a(V, V')$ is the conjunction of the union of the following sets of formulas

1. $\mathrm{pre}_a(V)$ which contains

$$\bigwedge_{v \in \mathrm{pre}(a)} v \wedge \bigwedge_{\neg w \in \mathrm{pre}(a)} \neg w \wedge \bigvee_{e \in \mathrm{post}(a)} c(e),$$

   where $c(e)$ is the formula obtained by the conjunction of all the literals of $\mathrm{cond}(e)$, i.e.,

$$c(e) = \bigwedge_{v \in \mathrm{cond}(e)} v \wedge \bigwedge_{\neg v \in \mathrm{cond}(e)} \neg v,$$

   ensuring that the preconditions $\mathrm{pre}(a)$ hold in s and that there is at least one CE in $\mathrm{post}(a)$ with its condition respected by s,

2. $\mathrm{eff}_a(V)$ which contains, for each $v \in V$,

$$v' \leftrightarrow (v \wedge \bigwedge_{e \in \Delta_a^-(v)} \neg c(e) \vee \bigvee_{e \in \Delta_a^+(v)} c(e), \tag{3.1}$$

   ensuring $s' = res(s, a)$, i.e., v' is true if v is true and no condition of a CE deleting v (in $\Delta_a^-(v)$) is respected, or at least one condition of a CE adding v (in $\Delta_a^+(v)$) is respected,

3. $\mathrm{conflict}_a(V)$, containing for each $v \in V$ and for each $e_1 \in \Delta_a^+(v)$ and $e_2 \in \Delta_a^-(v)$

$$\neg(c(e_1) \wedge c(e_2)),$$

   ensuring that no conflicting CEs' conditions can be respected in s.

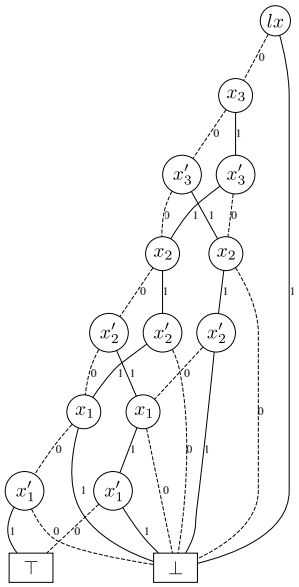

Fig. 3.2 BDD representation of the transition relation $\mathcal{T}_{\texttt{inx}}(V, V')$ of the COUNTERS example.

Example 3.2 (COUNTERS (ctd.))**.** *In our* COUNTERS *example, let's suppose* $B = 3$ *bits, for the action* `inx` *we have*

$$\begin{aligned}
\Delta^{+}_{inx}(x_1) &= \{\texttt{i}x_1\}, \qquad & \Delta^{-}_{inx}(x_1) &= \{\texttt{i}x_2, \texttt{i}x_3, \texttt{o}x\},\\
\Delta^{+}_{inx}(x_2) &= \{\texttt{i}x_2\}, \qquad & \Delta^{-}_{inx}(x_2) &= \{\texttt{i}x_3, \texttt{o}x\},\\
\Delta^{+}_{inx}(x_3) &= \{\texttt{i}x_3\}, \qquad & \Delta^{-}_{inx}(x_1) &= \{\texttt{o}x\},
\end{aligned}$$

and the transition relation $\mathcal{T}_{inx}(V, V')$ *for the action* `inx` *is composed of*

1. $\mathrm{pre}_a(V)$*, containing*

$$\neg lx,$$

2. $\mathrm{eff}_a(V)$*, containing*

$$\begin{aligned}
x_1' &\leftrightarrow \big(x_1 \wedge \neg(\neg x_2 \wedge x_1) \wedge \neg(\neg x_3 \wedge x_2 \wedge x_1) \wedge \neg(x_3 \wedge x_2 \wedge x_1)\big) \vee (\neg x_1),\\
x_2' &\leftrightarrow \big(x_2 \wedge \neg(\neg x_3 \wedge x_2 \wedge x_1) \wedge \neg(x_3 \wedge x_2 \wedge x_1)\big) \vee (\neg x_2 \wedge x_1),\\
x_3' &\leftrightarrow \big(x_3 \wedge \neg(x_3 \wedge x_2 \wedge x_1)\big) \vee (\neg x_3 \wedge x_2 \wedge x_1),\\
lx' &\leftrightarrow lx,
\end{aligned}$$

3. $\mathrm{conflict}_a(V)$ *containing, for example, for* x_1*, and for* $e_1 = \mathtt{i}x_1 \in \Delta^+_{inx}(x_1)$ *and* $e_2 = \mathtt{i}x_2 \in \Delta^-_{inx}(x_1)$

$$\neg(\neg x_1 \wedge \neg x_2 \wedge x_1),$$

which is equivalent to $\top$*, and similarly for all other* CE*s and variables.*

When representing the transition relations via BDDs, as seen in Section 3.1.2, choosing the correct order of the variables is of paramount importance to obtain a compact BDD. For the transition relation of an action with CEs, we respect the following rules. Let a be an action, in the order of variables of the BDD we impose that

1. the variable v' should follow immediately the corresponding variable v,
2. the variables that appear in $\mathrm{pre}(a)$ should be put first, to prune earlier all nodes in which the precondition is not respected,
3. for each CE $e \in \mathrm{post}(a)$, if $v \in \mathrm{cond}(e)$ and $w \in \mathrm{eff}(e)$, then $w < v$, to prune earlier the nodes in which the effects are not possible.

It is clear that these rules, are just heuristics, but indeed produce good results in the experimental analysis. Moreover, these rules could produce cyclic dependencies between variables, that need to be arbitrarily broken.

Example 3.3 (COUNTERS (ctd.))**.** *The above rules produce the order*

$$lx; lx'; x_3; x'_3; \ldots; x_1; x'_1$$

and thus the BDD *for the formula* $\mathcal{T}(V, V')$ *is as shown in Fig. 3.2.*

3.3.2 Computing the Transitive Closure

Let $\Pi = \langle V, A, I, G\rangle$ be a planning task and a be an action of Π. Let T_a be the transition function of a. The *transitive closure function* of T_a is a function $T^+_a : S \times S \mapsto \{\top, \bot\}$ such that for each state $s, s' \in S$, (i) a is applicable in s, (ii) there exists a $k \geq 1$ such that $s' = res(s, a^k)$, where a^k is the sequence of k repetitions of a.

The transitive closure function T_a^+ can be computed iteratively, starting from the transition function T_a. Let's rename T_a as T_a^0, i.e., the *first step transition of* $T_a(s, s')$. Then for each $i > 0$ we define the $i+1$*-th step transition of* $T_a(s, s')$, i.e., $T_a^i : S \times S \mapsto \{\top, \bot\}$ such that for each $s, s'' \in S$

$$T_a^i(s, s'') \leftrightarrow \exists s' \in S : T_a^{i-1}(s, s') \wedge (T_a^{i-1}(s', s'') \vee s' = s''), \tag{3.2}$$

Due to the finiteness of the set of states S, we know there exists a $p \geq 1$ such that

$$\forall s, s' \in S, \quad T_a^p(s, s') \leftrightarrow T_a^{p+1}(s, s'), \tag{3.3}$$

called the *fix-point function*. The transitive closure function is thus equivalent to the fix-point function T_a^p. We call p the *fix-point index*.

Lemma 3.2. *Let a be an action and let T_a^i be the $i+1$-th step transition function in the computation of the transitive closure function T_a^+, with $i \in [0, k]$. Let $s, s' \in S$ be two states and let a be applicable in s. If $T_a^i(s, s')$, then there exists a $r \in [1, 2^i]$ such that $s' = res(s, a^r)$.*

Sketch Proof. For $i = 0$, $T_a^0(s, s') = T_a(s, s')$ which by definition models $s' = res(s, a)$, i.e., $2^0 = 1$ applications of a. Then, given the computation procedure outlined by Eq. 3.2, T_a^1 models all the state reachable after *up to* 2 consecutive applications of a. We can reach states with 2 applications, because $T_a^1(s, s')$ checks the applicability of a in s and $T_a^1(s', s'')$ checks the applicability of a in s'. We can reach states with 1 applications because when $s' = s''$ in Eq. 3.2, $T^1(s, s'')$ coincides with $T^0(s, s'')$. Continuing the computation, we see that T_a^2 can model up to 4 consecutive applications of a, since we have T_a^1 to reach s' from s in at most 2 steps and again T_a^1 to reach s'' from s' in at most 2 steps. By induction, we conclude the proof. □

The $i+1$-th step transition relation function $T_a^i(s, s')$ thus specifies if s' is reachable from s in *at most* 2^i applications of a. In the following sections we will also need to know if the state is reachable in *exactly* 2^i applications of a. For this reason, we introduce the $i+1$*-th step exponential reachability function* of an action a as $R_a^i : S \times S \mapsto \{\top, \bot\}$ defined inductively such that

for each $s, s'' \in S$

$$\begin{aligned} R_a^0(s, s'') &\leftrightarrow T_a(s, s''), \\ R_a^i(s, s'') &\leftrightarrow \exists s' \in S : R_a^{i-1}(s, s') \wedge R_a^{i-1}(s', s''). \end{aligned} \tag{3.4}$$

Lemma 3.3. *Let a be an action and let R_a^i be its $i+1$-th step exponential reachability function.*

$$R_a^i(s, s') \leftrightarrow s' = res(s, a^{2^i}).$$

Sketch Proof. As for Lem. 3.2, by induction on Eq. 3.4. □

Lemma 3.4. *Let s be a state in which a is applicable. For any $i \geq 1$, there exists at most one state s' such that $R_a^i(s, s') = \top$.*

Sketch Proof. The proof follows easily from the determinism of the transition function $T_a(s, s')$, since, applying a in any state s leads to only one state s'. □

The computation of the transitive closure function and of the exponential reachability function presents some difficulties due to the existential quantifier ($\exists$) in Eq. 3.2 and 3.4 and the universal quantifier ($\forall$) in Eq. 3.3. To eliminate these quantifiers, we can instead compute the *transitive closure relation* $\mathcal{T}_a^+(V, V')$ from the transition relation $\mathcal{T}_a(V, V')$, where V and V' are copies of the variables in V. Eq. 3.2 thus becomes

$$\mathcal{T}_a^i(V, V'') \leftrightarrow \exists V' : \mathcal{T}_a^{i-1}(V, V') \wedge (\mathcal{T}_a^{i-1}(V', V'') \vee V' \equiv V''),$$

and distributing we obtain

$$\mathcal{T}_a^i(V, V'') \leftrightarrow \mathcal{T}_a^{i-1}(V, V'') \vee (\exists V' : \mathcal{T}_a^{i-1}(V, V') \wedge \mathcal{T}_a^{i-1}(V', V'')).$$

Thus, we can compute the $i+1$-th step transition $\mathcal{T}_a^i$ directly from the i-th step through iterated applications of the smoothing operator S as presented in Section 3.1.2 on all the variables in $V' = \{v'_1, \ldots, v'_n\}$ as

$$\mathcal{T}_a^i(V, V'') = \mathcal{T}_a^{i-1}(V, V'') \vee \mathsf{S}_{v' \in V'}(\mathcal{T}_a^{i-1}(V, V') \wedge \mathcal{T}_a^{i-1}(V', V'')). \tag{3.5}$$

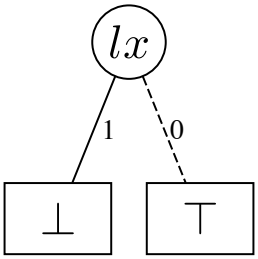

Fig. 3.3 BDD representation of the transitive closure $\mathcal{T}^{+}_{\texttt{inx}}(V, V')$ of the COUNTERS example.

To check whether a fix point has been reached, instead of relying on the universal quantifier in Eq. 3.3, we can simply check whether the propositional formulas $\mathcal{T}^{p}_{a}(V, V')$ and $\mathcal{T}^{p+1}_{a}(V, V')$ are equivalent. If we employ BDDs to represent the two formulas, this check is straightforward, due to the canonical representation provided by the BDDs. As for the transitive closure function, there must exist a *fix-point relation* $\mathcal{T}^{p}_{a}(V, V')$ such that $\mathcal{T}^{p}_{a}(V, V') \equiv \mathcal{T}^{p+1}_{a}(V, V')$ and thus the transitive closure is

$$\mathcal{T}^{+}_{a}(V, V') = \mathcal{T}^{p}_{a}(V, V'). \tag{3.6}$$

For the exponential reachability function R^{i}_{a}, we can again employ the *exponential reachability relation* $\mathcal{R}^{i}_{a}$ computed as

$$\mathcal{R}^{i}_{a}(V, V'') = \mathsf{S}_{v' \in V'} \mathcal{R}^{i-1}_{a}(V, V') \wedge \mathcal{R}^{i-1}_{a}(V', V''). \tag{3.7}$$

As stated in the introduction of this chapter, computing the transitive closure, can become intractable, due to formulas becoming exponentially long, even when employing BDDs. However, the computation of the transitive closure can be stopped at any time, before its fix-point index p, and the last computed relation $\mathcal{T}^{m}_{a}(V, V')$ with $m \in [0, p]$ is returned, which models the state reachable with up to 2^m repetitions of a (Lem. 3.2). It is clear that if a is idempotent, then $m = p = 0$. We name $\mathcal{T}^{m}_{a}(V, V')$ as the *timeout transition relation* and with m the *timeout index*.

Example 3.4 (COUNTERS (ctd.))**.** *The* BDD *representation of the transitive closure relation* $\mathcal{T}^{+}_{\texttt{inx}}(V, V')$*, for the action* `inx` *with* $B = 3$ *bits, is represented*

in Fig. 3.3. It is clear, that since we allow overflowing ($111 \rightarrow 000$), then from any number we can reach any number, only if the precondition is respected.

3.4 The Transitive $\prec$-Encoding for Classical Planning with CEs

Let $\Pi = \langle V, A, I, G \rangle$ be a classical planning task with CEs. Let $\prec = a_1; \ldots; a_k$ be a pattern. For each action a_i in the pattern $\prec$, let $\mathcal{T}_{a_i}(V, V')$ be the transition relation of a_i and let $\mathcal{T}^+_{a_i}(V, V')$ be its transitive closure[2]. We now present the transitive SPP encoding for Π, namely $\Pi^{\prec+}$. As for the numeric case presented in Chapter 2, we assume to have the sets of variables $\mathcal{X}$, $\mathcal{A}^{\prec}$, $\mathcal{X}'$, representing the *current state*, *action* and *next state* variables.

1. In $\mathcal{A}^{\prec}$ we have a propositional variable a_i^+ for each action a_i in the pattern, denoting that the action a_i in $\prec$ is executed consecutively at least one time.

2. The current state variables $\mathcal{X}$ contains
   (a) all the variables in V, and
   (b) for each non-idempotent action a_i in the pattern $\prec$ and for each $v \in V$, a propositional variable v_i.

The set $\mathcal{X}'$ is, as usual, a copy of $\mathcal{X}$. Thus, the transitive $\prec$-encoding for Π 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, defined as

$$\bigwedge_{v \in I} v \wedge \bigwedge_{w \in V \setminus I} \neg w,$$

[2]As stated before, the procedure presented will remain correct and complete even if, for each action we do not employ its transitive closure relation, but we stop at its timeout relation. For readability, we will still use $\mathcal{T}^+_{a_i}$ instead of $\mathcal{T}^m_{a_i}$.

2. $\mathcal{G}(\mathcal{X}')$ is the goal state formula, defined as

$$\bigwedge_{v \in G} v \wedge \bigwedge_{\neg w \in G} \neg w.$$

3. $\mathcal{T}^{\prec+}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ is a *transitive $\prec$-symbolic transition relation*, providing a definition of each variable in $\mathcal{X}'$ as a function of the variables in $\mathcal{X} \cup \mathcal{A}^{\prec}$.

As for the numeric planning SPP encoding, for each $i \in [0, k]$ we now define the value $\sigma_i(v)$ of each variable $v \in V$ denoting the value of v after the (possible and repeated) application of a_i in $\prec$, i.e., a function of the variables in $\mathcal{X} \cup \{a_1^+, \ldots, a_i^+\}$. By extension, we have $\sigma_i = \{\sigma_i(v) \mid v \in V\}$. Clearly, if $i = 0$, $\sigma_0(v) = v$ while if $i \in [1, k]$, $\sigma_i(v)$ is recursively defined as follows:

1. if $v \notin \mathrm{add}(a) \cup \mathrm{del}(a)$, the value of v does not change, and thus

$$\sigma_i(v) = \sigma_{i-1}(v),$$

2. if a_i is idempotent, then

$$\sigma_i(v) = \Big(\sigma_{i-1}(v) \wedge \Big(\neg a_i^+ \vee \bigwedge_{e \in \Delta_a^-(v)} \neg \sigma_{i-1}(\psi_e)\Big)\Big) \vee \Big(a_i^+ \wedge \bigvee_{e \in \Delta_a^+(v)} \sigma_{i-1}(\psi_e)\Big),$$

   which, if $a_i^+ = \bot$ collapses to $\sigma_i(v) = \sigma_{i-1}(v)$, if $a_i^+ = \top$ is the same as Eq. 3.1,

3. if a_i is non-idempotent, then

$$\sigma_i(v) = \mathrm{ITE}(a_i^+, v_i, \sigma_{i-1}(v))$$

The transitive $\prec$-symbolic transition relation $\mathcal{T}^{\prec+}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ is thus the conjunction of the union of the following sets:

1. $\mathrm{closure}^{\prec+}(A, V)$, which contains, for each action a_i in the pattern,

$$a_i^+ \rightarrow \mathcal{T}_{a_i}^+(\sigma_{i-1}, \sigma_i),$$

   where $\mathcal{T}_{a_i}^+(\sigma_{i-1}, \sigma_i)$ is the formula $\mathcal{T}_{a_i}^+(V, V')$ where each $v \in V$ and $v' \in V'$ are substituted with $\sigma_{i-1}(v)$ and $\sigma_i(v)$, respectively – i.e., if a_i^+ is

Algorithm 3.1 SPP algorithm applied to the case of CES

```
1: function CES-SPP(Π)      /* Π = ⟨V, A, I, G⟩ */
2:     n ← 0; ≺ ← ϵ;
3:     ≺_I ← COMPUTEPATTERN(Π);
4:     while (TRUE) do
5:         Π^{≺+} ← I(X) ∧ T^{≺+}(X, A^{≺}, X') ∧ G(X');
6:         μ ← SOLVE(Π^{≺+});
7:         if (μ ≠ 0) then
8:             return GETPLANCLOSURE(μ, ≺);
9:         end if
10:        ≺ ← ≺; ≺_I;
11:        n ← n + 1;
12:    end while
13: end function
```

executed, the state represented by σ_i must be reachable from the state represented by σ_{i-1},

2. $\text{frame}^{\prec+}(V)$, which contains, for each $v \in V$,

$$v' \;\leftrightarrow\; \sigma_k(v).$$

3.5 Valid Plan with the Transitive $\prec$-Encoding.

Let $\Pi = \langle V, A, I, G\rangle$ be a planning task with CES and let $\prec = a_1; \ldots; a_k$ be a pattern. As for the numeric case, we can directly employ the SPP procedure outlined in Alg. 2.1 of Chapter 2, repeated in Alg. 3.1 for readability, with the only exception being the call GETPLANCLOSURE$(\mu, \prec)$. In fact, in any satisfiable model $\mu : \mathcal{X} \cup \mathcal{A} \cup \mathcal{X}' \mapsto \{\top, \bot\}$ of $\Pi^{\prec+}$ and for each action a_i in the pattern $\prec$, we have $\mu(a_i^+) = \top$ if a_i must be executed *at least* one time. We need thus a procedure able to compute exactly how many times a_i must be repeated, and thus produce a valid plan π for Π.

Let μ be a solution of the encoding $\Pi^{\prec+}$, found in Line 6 of Algorithm 3.1. Let a_i be an action of $\prec$ such that $\mu(a_i^+) = \top$. We know that the value of a variable $v \in V$ before and after the rolling of a_i is expressed through the expressions $\sigma_{i-1}(v)$ and $\sigma_i(v)$ respectively. The states s_{i-1}, s_i before and after

Algorithm 3.2 Computation of the rolling of a needed to reach s'' from s.

```
1: global T_a^0, ..., T_a^p, R_a^0, ..., R_a^p    //previously computed
2: function ROLLING(a, s, s'', p)
3:     if T_a^0(s, s'') then
4:         return 1
5:     end if
6:     for j ∈ [1, p] do
7:         if T_a^j(s, s'') then
8:             s' ← s' : R_a^{j-1}(s, s')
9:             return 2^{j-1} + ROLLING(a, s', s'', j − 1)
10:        end if
11:    end for
12:    return -1
13: end function
```

the rolling of a_i can thus be computed as

$$s_{i-1} = \{v \mid \mu[\sigma_{i-1}(v)] = \top\} \qquad s_i = \{v \mid \mu[\sigma_i(v)] = \top\}, \tag{3.8}$$

where $\mu[\sigma_i(v)]$ (resp. $\mu[\sigma_{i-1}(v)]$) is the expression obtained by replacing each variable $w \in \mathcal{X}$ in $\sigma_i(v)$ (resp. $\sigma_{i-1}(v)$) with $\mu(w)$. Let $\mathcal{T}_{a_i}^0, \ldots, \mathcal{T}_{a_i}^+$ be the sequence of transition relations computed when computing the transitive closure $\mathcal{T}_{a_i}^+$, and let $\mathcal{R}_{a_i}^0, \ldots, \mathcal{R}_{a_i}^+$ be the correspondent exponential reachability relations. We know that the transition function $T_a^i(s, s')$ is true if and only if the transition relation $\mathcal{T}_a^i(V, V')$ is equivalent to $\top$ when substituting each $v \in V$ (resp $v' \in V'$) with $\top$ if $v \in s$ (resp. $v \in s'$) and with $\bot$ otherwise. Alg. 3.2 shows the structure of the ROLLING algorithm, which takes as input the action a, the state before (s) and after (s'') the rolling of a and, initially, the index p is either fix-point's index or the timeout index. The algorithm, starting from $j = 0$ explores whether s'' is reachable from s in at most 2^j times (Lem. 3.2), increasing j upon failure. When such a j is found, it means that the rolling of a lies in $(2^{j-1}, 2^j]$. For this reason, Line 8 employs Lem. 3.4 and, using the exponential reachability function R_a^{j-1}, finds the intermediate state s' after 2^{j-1} repetitions of a and calls again the function ROLLING, searching the rolling of a to reach s'' from s', bounded by at most 2^{j-1}. If state s'' is not reachable from s' in up to 2^p repetitions of a, then ROLLING returns -1, signalling unreachability.

Theorem 3.5. *Let s, s'' be two states, a be an action applicable in s, and let $p \geq 0$. $r = \text{ROLLING}(a, s, s'', p) > 0$ implies $s'' = res(s, a^r)$.*

Proof. By induction. If $s'' = res(s, a)$, then $T_a^0(s, s'') = T_a(s, s'') = \top$ by construction and $\text{ROLLING}(a, s, s'', p) = 1$. Suppose that the thesis holds for all $r \in [1, 2^{j-1}]$, for some $j \in [1, m]$. If $r \in (2^{j-1}, 2^j]$, then, by Lem. 3.2, we know $T_a^0(s, s'') = \cdots = T_a^{j-1}(s, s'') = \bot$ and $T_a^j(s, s'') = \top$. Thus by Lem. 3.4, we know there exists only one state s' such that $R_a^{j-1}(s, s') = \top$ and, by Lem. 3.3 we know that $s' = res(s, a^{(2^{j-1})})$ thus, to reach s'' from s', we still need $r' = r - 2^{j-1}$ repetitions. Since $r \in (2^{j-1}, 2^j]$, then $r' \in [1, 2^{j-1}]$ and following the inductive hypothesis, r' can be computed as $\text{ROLLING}(a, s', s'', j - 1)$. □

Finally, let $\pi = a_1; \ldots; a_k$ be the pattern such that $\Pi^{\prec+}$ is satisfiable by model μ, i.e., Line 7 of Alg. 3.1, the function GETPLANCLOSURE of Line 8 returns the sequence

$$\pi = a_1^{r_1}; \ldots; a_k^{r_k},$$

where r_i for $i \in [1, k]$ is computed as

$$r_i = \text{ROLLING}(a_i, s_{i-1}, s_i, p_i)$$

with s_{i-1} and s_i as described in Eq. 3.8 and p_i being the index of the fix point at which $\mathcal{T}_{a_i}^+(V, V')$ is found.

3.6 Correctness, Completeness, and Domination

Let $\Pi = \langle V, A, I, G \rangle$ be a classical planning task with CEs. In this section, we will prove some theoretical properties – namely correctness, completeness, and domination – of the $\Pi^{\prec+}$ encoding we presented in the previous sections.

3.6.1 Correctness and Completeness

We recall from Chapter 2 that, as standard for PAS encodings, the transitive $\prec$-encoding $\Pi^{\prec+}$ has to guarantee the properties of *correctness* and *completeness*.

1. Correctness: each model μ of $\mathcal{T}^{\prec+}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ has to correspond to at least one sequence of actions α such that (i) α is applicable in the state $s = \{v \mid \mu(v) = \top\}$ and (ii) the last state induced by α executed is the state $s' = \{v \mid \mu(v') = \top\}$.

2. Completeness: for each state s and action a in A, if s' is the state resulting from the application of a in s, then there must be a model μ of $\mathcal{T}^{\prec+}(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}')$ such that for each variable $v \in V$, $v \in s$ iff $\mu(v) = \top$ and $v \in s'$ iff $\mu(v') = \top$

Let $\Pi^{\prec 0}$ be the $\prec$-encoding constructed the same way as $\Pi^{\prec+}$ but where, for each action a_i in $\prec$, the transitive closure $\mathcal{T}^{+}_{a_i}(V, V')$ is substituted with the transitive relation $\mathcal{T}^{0}_{a_i}(V, V')$, i.e., each action a_i in $\prec$ can be executed at most once.

Theorem 3.6. *Let Π be a classical planning task with* CE*s, let $\prec$ be a complete pattern. The $\prec$-encoding $\Pi^{\prec 0}$ is correct and complete.*

Proof. The correctness and completeness proof is a simplification of the numeric case in Thm. 2.3 where $V_N = \emptyset$ and $V_B = V$ and the effect axioms modified as the one in Eq. 3.1. □

Let M be a *timeout* function, assigning to each action a_i in the pattern $\prec$ a positive number $M(a_i)$. We denote with $\Pi^{\prec M}$ the $\prec$-encoding constructed the same way as $\Pi^{\prec+}$ but where, for each action a_i in $\prec$, the transitive closure $\mathcal{T}^{+}_{a_i}(V, V')$ is substituted with the transitive relation $\mathcal{T}^{M(a_i)}_{a_i}(V, V')$, i.e., action a_i can be applied at most $2^{M(a_i)}$ times.

Theorem 3.7. *Let Π be a classical planning task with* CE*s, let $\prec$ be a complete pattern. The $\prec$-encoding $\Pi^{\prec M}$ is correct and complete.*

Proof. If, for each a_i in $\prec$, $M(a_i) = 0$, we fall back to Thm. 3.6. If, instead, $M(a_i) \in [1, p_i]$ for some action a_i in $\prec$, where p_i is the index of fix point of the computation of the transitive closure $\mathcal{T}^{+}_{a_i}$. Correctness comes from the correctness of the computation of the transitive closure in Lem. 3.2. Completeness comes from the fact that one application of the action a^{+}_i models between 1 and $2^{M(a_i)}$ executions of a and thus a model μ of $\mathcal{T}^{0}_{a_i}(V, V')$ is also a model of $\mathcal{T}^{M(a_i)}_{a_i}(V, V')$. If $M(a_i) > p_i$, due to the fix-point being reached, we have $\mathcal{T}^{M(a_i)}_{a_i} \equiv \mathcal{T}^{+}_{a_i}$ and we fall back to the previous case. □

Theorem 3.8. *Let Π be a classical planning task with* CEs*, let $\prec$ be a complete pattern. The $\prec$-encoding $\Pi^{\prec+}$ is correct and complete.*

Proof. This is a special case of Thm. 3.7, where, for each a_i in $\prec$, $M(a_i) = p_i$, with p_i being the index of fix point of the computation of the transitive closure □

Theorem 3.9. *Let Π be a classical planning task with* CEs*.* CES-SPP(Π) *is correct and complete.*

Sketch Proof. The correctness follows directly from the correctness of $\Pi^{\prec+}$ and Thm. 3.5. For the completeness, according to Thm. 2.1, a plan of length n will be found at most at the n-th iteration of the CES-SPP(Π) algorithm. □

3.6.2 Domination

Let $\Pi = \langle V, A, I, G\rangle$ be a classical planning task with CEs. In the literature, the most advanced approach for dealing with Classical Planning with CEs is the $R^2\exists$-encoding $\Pi^{R^2\exists,\prec}$, already presented in Section 2.3.2 for the numeric case, where we consider the case in which $V_N = \emptyset$ and $V_B = V$ and the use of the formula in Eq. 3.1 for the effect axioms with CEs [Rintanen, 2011; Wehrle and Rintanen, 2007; Balyo, 2013]. The $\Pi^{R^2\exists,\prec}$ encoding can thus be mapped in our encoding $\Pi^{\prec 0}$, since the transitive closure is not employed, modelling, in the transition relation $\mathcal{T}^{R^2\exists,\prec}(\mathcal{X}, \mathcal{A}, \mathcal{X}')$, only a single application of each action $a \in A$.

Theorem 3.10. *Let Π be a numeric planning problem. Let $\prec$ be a simple and complete pattern and let M be a timeout function. The $\prec$-encoding $\Pi^{\prec M}$ dominates the $\prec$-encoding $\Pi^{\prec 0}$ and the $\prec$-encoding $\Pi^{\prec+}$ dominates the $\prec$-encoding $\Pi^{\prec M}$.*

Proof. ($\Pi^{\prec M}$ dominates $\Pi^{\prec 0}$) To prove dominance we have to demonstrate that, for any bound n, if $\Pi_n^{\prec 0}$ is satisfiable the also $\Pi_n^{\prec M}$ is satisfiable. This follows directly from the fact that, for each action a_i in the pattern, due to Lem. 3.2, if $\mathcal{T}_{a_i}^0(V, V')$ is satisfiable (1 repetition), then also $\mathcal{T}_{a_i}^{M(a_i)}(V, V')$ is satisfiable (at least 1 repetition), since $M(a_i) \geq 0$.

($\Pi^{\prec +}$ dominates $\Pi^{\prec M}$) For each action a_i, let $M(a_i)$ be in $[0, p_i]$ with p_i being the fix point index of a_i. Since $\mathcal{T}_{a_i}^{+}(V, V')$ is equivalent by construction to $\mathcal{T}_{a_i}^{p_i}(V, V')$, by Lem. 3.2, we have that if $\mathcal{T}_{a_i}^{M(a_i)}(V, V')$ is satisfiable (between 1 and $2^{M(a_i)}$ repetitions) then $\mathcal{T}_{a_i}^{p_i}(V, V')$ is satisfiable (between 1 and 2^{p_i} repetitions). If, for each action a_i, $M(a_i) > p_i$, then we have $\mathcal{T}_{a_i}^{M(a_i)}(V, V') \equiv \mathcal{T}_{a_i}^{p_i}(V, V')$. □

3.7 Experimental Analysis

To analyse our approach, we present an experimental analysis run on the COUNTERS domain, the motivating example of this chapter, slightly extended to deal with multiple counters instead of just two. Figure 3.4 shows the analysis run on the COUNTERS domain with 5 counters and on problems with increasing bits B, from 2 to 12. We employed the search-based solver ENHSP [Scala et al., 2016c], the PAS-based MADAGASCAR solver (MPC) [Rintanen, 2011], which is the only PAS solver in the literature able to deal with CES, and which implements the $R^2\exists$-encoding, and our solver PATTY, modified to deal with CES with the presented approach. For each solver, we provided a timeout of 5 minutes for finding a valid plan and for the PATTY solver, we provided a timeout of 1 minute for computing the transitive closure for each action. After this timeout, the last found transition relation was returned. Experiments have been run on Intel Xeon Platinum 8000 3.1GHz with 8 GB of RAM.

It is worth noticing that, in the case with 5 counters $\{c_1, \ldots, c_5\}$, there are 14 actions in A: 5 actions to increase each counter, 5 actions to decrease them and 4 actions to lock adjacent counters, i.e., if c_i and c_{i+1} have the same bits, with $i \in [1, 4]$. The actions to lock the counters are idempotent, and thus the transitive closure computation can be skipped. For the other 10 actions, is sufficient to compute the transitive closure for the action to increase c_1 and the action to decrease c_1, and then substitute the variables in the resulting formula for all the other 4 counters. For each problem with B bits, the initial state sets the counters c_1,c_3 and c_5 to 2^{B-1}, and counters c_2 and c_4 to 0. Thus, each counter must be increased/decreased at least 2^{B-2} times to reach the adjacent counter.

Figure 3.4, shows that the ENHSP solver can rapidly solve the problems up to 6 bits, starting to struggle with 7 bits and being unable to solve the problem with 8 or more bits before the timeout. The MADAGASCAR solver instead cannot solve problems with more than 5 bits, due to the exponential nature of the bound. It can be noted how with B bits, MADAGASCAR can solve the problem with bound $n = 2^{B-2} + 1$, i.e., 2^{B-2} steps to increase all the counters by 2^{B-2} and another step to lock all the adjacent counters. The PATTY solver, instead, can solve all the problems up to 10 bits with bound $n = 1$. The dashed black line represents the time it takes to compute the transitive closure of the two actions. It is clear, that with B bits, for each action a, the transitive closure $\mathcal{T}_a^+$ is equivalent to $\mathcal{T}_a^B$, since either the increase or the decrease can be repeated at most $2^B - 1$ times (Lem. 3.2). When there are 11 bits, the computation of the transitive closure for the two actions timeouts, and it is returned the last transition relation found, in this case, for each action a, it is $\mathcal{T}_a^8$, modelling up to 2^8 repetitions of either increase or decrease of each counter. For this reason, with $B = 11$, the bound is equal to $n = 2$, since

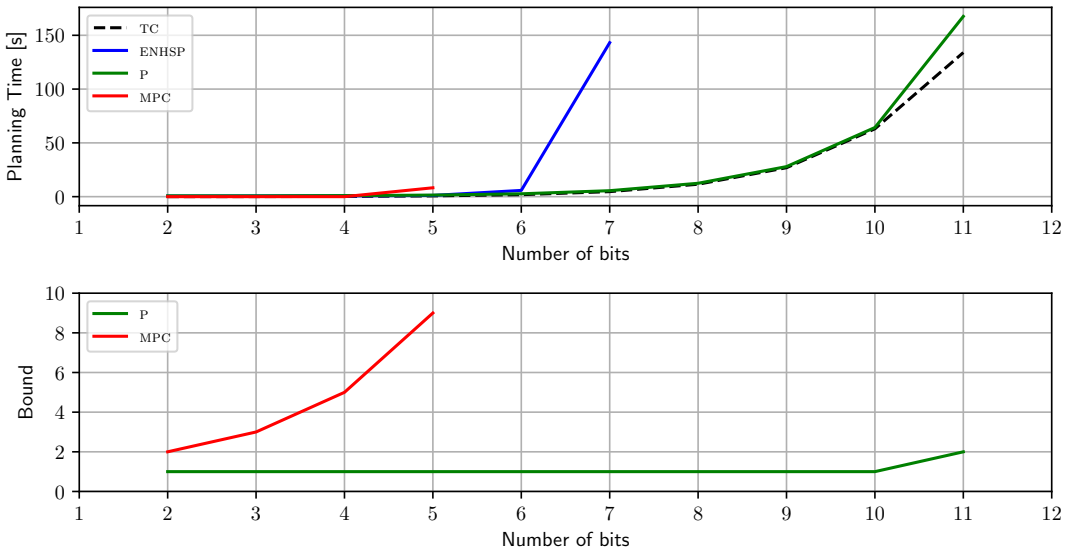

Fig. 3.4 Experimental analysis run on the COUNTERS domain with 5 counters and on problems with increasing bits B, from 2 to 12. The comparison is made with the ENHSP solver, the Madagascar (MPC) solver and our PATTY, modified to deal with CES. The dashed black line indicates the time to compute the transitive closure. The top line-chart measures the planning time in seconds w.r.t. the number of bits and the bottom line-chart measures the bound at which the solution was found.

the first bound takes care of 2^8 increases/decreases of the counters and the second bound performs the other 2^8, for a total of $2^{B-2} = 2^9$ repetitions. It can be noted how the domination proved in Thm. 3.10 is confirmed in the experimental analysis, since the bound at which MADAGASCAR finds a solution is always greater than the one in which PATTY finds a solution.

3.8 Conclusion and Future Work

In this chapter, we examinated the flavour of classical planning with CEs. Given the symmetry proposed by [Gigante and Scala, 2023], we investigated how we could bring the concept of rolling also to classical planning with CEs, where non-idempotent actions can be consecutively repeated. We found the answer to our investigation in the computation of the transitive closure, and we proposed a SPP approach which outperforms state-of-the-art solvers and provably dominated all PAS approaches existing in the literature.

In the experimental analysis, we saw that most of the computation is dedicated to computing the transitive closure of each non-idempotent action. In our approach, we applied a naive and basic computation of the transitive closure, but in the past three decades, there has been a lot of effort in finding faster ways to compute the transitive closure, either through partitioning of variables and formulas [Cabodi et al., 1997; Geldenhuys and Valmari, 2001], use of transformations to integer functions [Ciardo et al., 2001], matrix manipulations [van Dijk et al., 2019; Brand et al., 2023]. Moreover, the use of BDDs has already been proven very effective in heuristics for classical planning [Edelkamp and Reffel, 1998; Kissmann and Hoffmann, 2013, 2014] and thus it could be worthwhile to explore strategies to improve the transitive closure computation exploiting the special case of Classical Planning with CEs.

Chapter 4

SPP in Numeric Temporal Planning

In this chapter[1], 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 chapter, we extend the recently proposed Symbolic Pattern Planning (SPP) approach of Chapter 2 to handle temporal numeric problems. 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 performance 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 can find a valid plan with a lower bound on all the problems.

[1]Part of this chapter has been published in [Cardellini and Giunchiglia, 2025]

4.1 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 $\langle b^{\vdash}, b^{\mapsto}, b^{\dashv}, [L, U]\rangle$, where $b^{\vdash}, b^{\mapsto}, b^{\dashv}$ 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 $b^{\mapsto}$ has no effects, and its preconditions $\mathrm{pre}(b^{\mapsto})$ 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 $\langle b^{\vdash}, b^{\dashv}, b^{\mapsto}, [L, U]\rangle$ 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 does not matter, 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 $\langle t, \langle b^{\vdash}, b^{\dashv}, b^{\vdash\!\!\dashv}, d\rangle\rangle$ in π, if $b^{\vdash}$ and $b^{\dashv}$ are executed at $t_i = t$ and $t_j = t_i + d$ respectively, the preconditions of $b^{\vdash\!\!\dashv}$ are satisfied in each state $s_i, \ldots, s_{j-1}$.

4.2 Standard Encodings in SMT

Several approaches for computing a valid plan of Π have been proposed, either based on search (see, e.g., [Benton et al., 2012; Gerevini et al., 2010; Eyerich

et al., 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 et al., 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 $\mathcal{X}_{i+1}$ on the basis of 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. (2016d), the $R^2\exists$ encoding by Bofill et al. (2016), and the pattern $\prec$-encoding of Chapter 2. 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* $A = \{nc_k \mid k \in [1,q]\} \cup \{pr_{i,j} \mid i \in [1,p], j \in (p,q]\}$ *whose actions are:*

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

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* $pr_{1,2}$*,* nc_1 *and* nc_2*. Considering the starting/ending actions,* $pr^{\vdash}_{1,2}$ *is mutex with* $nc^{\vdash}_1$*,* $nc^{\dashv}_1$*.* $nc^{\vdash}_2$*,* $nc^{\dashv}_2$*. If the bottles are initially capped and the durations allow to pour all the litres with just one execution of* nc_1 *and* 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* $pr_{1,2}$ *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* $pr_{1,2}$ *(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.*

4.3 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.

4.3.1 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 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.

In order to have simple variable names associated to each action in the pattern, 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.

4.3.2 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) $v = V \in \mathrm{pre}(b^{\vdash\!\dashv}) \cup \mathrm{pre}(b^{\dashv})$ 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.

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,

$$\sigma_i(v) = \mathrm{ITE}(a_i > 0, \sigma_{i-1}(\psi), \sigma_{i-1}(v)).$$

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 4.1. *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 Thm. 2.2). □

Example (cont'd). *For $i \in [1, p]$, $j \in (p, q]$, the pouring action $\mathtt{pr}_{i,j}$ is eligible for rolling while both $\mathtt{nc}_i$ and $\mathtt{nc}_j$ are not. Action $\mathtt{pr}_{i,j}$ 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.*

4.3.3 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 4.1):
   $$a_i > 0 \rightarrow \sigma_{i-1}(\psi[0, b^{\vdash}, 0, b^{\dashv}]) \trianglerighteq 0,$$
   $$a_i > 1 \rightarrow \sigma_{i-1}(\psi[a_i - 1, b^{\vdash}, a_i - 1, b^{\dashv}]) \trianglerighteq 0,$$
   (c) if a_i is ending b, (i.e., if $a_i = b^{\dashv}$) (Theorem 4.1, 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*

$$\begin{array}{l} nc_1^{\vdash}; nc_2^{\vdash}; nc_3^{\vdash}; nc_4^{\vdash}; pr_{1,3}^{\vdash}; pr_{1,4}^{\vdash}; pr_{2,3}^{\vdash}; pr_{2,4}^{\vdash}; \\ nc_1^{\dashv}; nc_2^{\dashv}; nc_3^{\dashv}; nc_4^{\dashv}; pr_{1,3}^{\dashv}; pr_{1,4}^{\dashv}; pr_{2,3}^{\dashv}; pr_{2,4}^{\dashv}. \end{array} \tag{4.1}$$

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

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

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., 2016d], and (ii) the combination of multiple even contradictory effects on a same variable by different actions, as in the $R^2\exists$ encoding [Bofill et al., 2016].

4.3.4 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 muteq:chapter4: 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 $b^{\vdash\!\dashv}$ of b. Hence, the formulas checking the executability of $b^{\vdash\!\dashv}$ 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 $\prec$:

$$\begin{gathered} a_i > 0 \rightarrow t_i \geq t_0 + \epsilon, \\ a_i = 0 \rightarrow t_i = t_0 \wedge d_i = 0, \quad a_j = 0 \rightarrow t_j = t_0, \\ a_i > 0 \rightarrow a_i \times (L + \epsilon_b) \leq d_i + \epsilon_b \leq a_i \times (U + \epsilon_b). \end{gathered}$$

   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:

$$\begin{aligned} a_i > 0 &\rightarrow \textstyle\bigvee_{j \in E_i}(a_i = a_j \wedge t_j = t_i + d_i), \\ a_j > 0 &\rightarrow \textstyle\bigvee_{i \in S_j}(a_i = a_j \wedge t_j = t_i + d_i), \end{aligned}$$

   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':

$$\begin{aligned} a_i > 1 \rightarrow (&t_i \geq t_j + d_j \vee t_j \geq t_i + d_i \vee \\ &a_j = 1 \wedge t_i \geq t_j \wedge t_i + d_i \leq t_j + d_j). \end{aligned}$$

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. $\mathrm{noOverlap}^{\prec}(A)$: 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. $\mathrm{lasting}^{\prec}(A)$: for each durative action b with $\mathrm{pre}(b^{\vdash}) \neq \emptyset$, 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 $\mathrm{pre}(b^{\vdash})$ (Theorem 4.1):

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

   (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 $\mathrm{pre}(b^{\vdash})$:

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

Example (cont'd). *For $pr^{\vdash}_{i,j}$ (resp. $nc^{\vdash}_k$), 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 $pr_{i,j}$ and nc_k are 1 and 5 respectively, the temporal pattern encoding entails:*

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

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.

4.3.5 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 $b^{\vdash}_1/b^{\dashv}_1$ mutex with $b^{\vdash}_2$, it might be useful to have a pattern including $b^{\vdash}_1; b^{\dashv}_1; b^{\vdash}_2; b^{\vdash}_1; b^{\dashv}_1$ 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

$$\Pi^{\prec} = I(\mathcal{X}) \wedge T^{\prec}_s(\mathcal{X}, \mathcal{A}^{\prec}, \mathcal{X}') \wedge T^{\prec}_t(\mathcal{A}^{\prec}, \mathcal{T}^{\prec}) \wedge G(\mathcal{X}'), \tag{4.2}$$

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 (4.2) by considering $\prec^n$ as the pattern $\prec$.

Theorem 4.2. *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 $\prec_\pi$. □

	Coverage (%)						Time (s)						Bound (Common)		
Domain	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 4.1 Comparative analysis between our planner PATTY$_T$, the logic-based solvers ANMLSMT, ITSAT and the search-based solvers LPG, OPTIC and TFD. A "-" means that the planner cannot parse the problems in the domain. Best results are in bold.

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.

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 (4.1), the ending action $pr_{i,j}^{\dashv}$ of the pouring actions are after the ending action $nc_k^{\dashv}$ that caps the bottle. However, such two actions are not in mutex and our pattern time encoding does not enforce $t_{i,j}^{\dashv} > t_k^{\dashv}$, 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.*

4.4 Experimental Results

Table 4.1 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 chapter. SHAKE allows emptying a bottle by shaking it while uncapped. PACK calls for concurrently pairing two bottles together to be packed. The domain BOTTLESALL puts together all the actions and characteristics of the three aforementioned domains. Of these 10 domains, only POUR and BOTTLESALL, contain actions eligible for rolling.

The analysis compares our system PATTY_T implemented by modifying the planner PATTY and using the SMT-solver Z3 v4.8.7 [de Moura and Bjørner, 2008]; the symbolic planners ANMLSMT (which corresponds to $\text{ANML}^{\text{OMT}}_{\text{INC}}$ (OMSAT) in [Panjkovic and Micheli, 2023]) and ITSAT [Rankooh and Ghassem-Sani, 2015]; and the search-based planners OPTIC [Benton et al., 2012], LPG [Gerevini et al., 2010] and TEMPORALFASTDOWNWARD (TFD) [Eyerich et al., 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 et al., 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 sub-tables/columns, 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 $\prec$ has been initially computed using the Asymptotic Relaxed Planning Graph, introduced in [Scala et al., 2016b].

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 of 4 for the second best). The Painter domain is the only one where PATTY_T has 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.

4.5 Conclusion

We extended the SPP approach proposed in Chapter 2 to the temporal numeric setting. We proved its correctness and completeness. We experimentally showed the benefits of our encoding 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. Indeed, it is well known that the main issue of planning as satisfiability is the value of the bound: the higher the bound, the more calls to the solver. Furthermore, the number of variables and the encoding size increase with the bound, making problems more difficult as the bound increases. For these reasons, much of the work, even in the classical setting, has focused on defining encodings that allow to find valid plans with a lower value for the bound, see, e.g., [Kautz et al., 1996; Rintanen, 2007; Wehrle and Rintanen, 2007; Rankooh and Ghassem-Sani, 2015; Scala et al., 2016d; Cardellini et al., 2024b]. This is the first work following this line of research in the temporal numeric setting.

Chapter 5

Boosting SPP with Symbolic Search

In this chapter, we push the envelope of planning with patterns, and show how to symbolically search for a valid plan by iteratively extending (adding actions to) and simplifying (removing actions from) the initially computed pattern. Specifically, the idea is to concatenate

1. an initial, simplified pattern that allows reaching a state s satisfying a subset of the goals, and

2. another pattern computed from s, which is extended until another state satisfying some new subgoals can be reached.

At the beginning, the initial pattern is computed from the initial state, and the two steps are iterated until all the subgoals have been satisfied, at which point a valid plan is returned. The proposed procedure is proven correct (any returned plan is valid) and complete (if there exists a valid plan, one will be returned). On the experimental side, the analysis shows that the proposed procedure outperforms both the previous SPP approach and all the other publicly available planners on the domains and problems of the 2023 International Planning Competition (IPC). In particular, our procedure, compared to the original SPP approach, can solve more problems (i.e., it has higher coverage) on 6 out of the 10 domains whose problems were not already all solved. Considering all the other publicly available numeric planners, our procedure achieves the highest coverage on 14/19 of the domains, compared to 9/19 for the second-highest one. Ablation studies reveal that simplifying the initial pattern by removing actions that are not necessary to reach the

state where the pattern is recomputed has the most significant impact on the performance of our procedure.

Summing up, the main contributions of the chapter are:

1. We present a SPP search-based procedure for numeric planning.
2. We prove its correctness and completeness.
3. We show that it outperforms all the publicly available planners on the benchmarks of the 2023 IPC.
4. We conduct ablation studies to highlight which technique is most effective.

After the background on planning as satisfiability with patterns, a simple motivating example illustrates the drawbacks of the original SPP approach, followed by our procedure, its behaviour on the motivating example, and the experimental analysis.

5.1 Motivating Example

In a relay race, there are $N+1$, $N > 0$, runners $r_0, r_1, \ldots, r_N$ running on a linear track (an x axis) of length $(N+1) \times L$ with $L \geq 1$, passing a baton to each other. The position x_i of runner r_i ranges in $[L \times i, L \times (i+1)]$, with $i \in [0, N]$. Each runner can run forward or backward, increasing or decreasing its position by 1, only if it is holding the baton. To exchange the baton, two runners r_i and r_{i+1} must be in the same position. We assume that $b_i = 1$ if r_i has the baton and $b_i = 0$ otherwise,[1] while btd_i is a Boolean variable denoting if r_i has touched the baton. In all the problems in this domain, we assume that initially the runner r_0 has the baton, that he is the only one who has touched the baton and that each runner r_i has position $L \times i$. In this scenario, for each $i \in [0, N]$ and $j \in [1, N]$, we have the actions fw_i, bw_i and bxc_j modelling respectively the running forward and backward of r_i, and the baton exchange

[1]We use $b_i \in \{0, 1\}$ instead of a Boolean variable to allow for a concise modelling of the baton exchange action.

between r_{j-1} and r_j, where

$$\begin{aligned} fw_i &: \langle \{x_i < L \times (i+1), b_i > 0\}, \{x_i \mathrel{+}= 1\} \rangle, \\ bw_i &: \langle \{x_i > L \times i, b_i > 0\}, \{x_i \mathrel{-}= 1\} \rangle, \\ bxc_j &: \langle \{x_j = x_{j-1}, b_j + b_{j-1} > 0\}, \\ &\qquad \{b_j := b_{j-1}, b_{j-1} := b_j, btd_j := \top\} \rangle. \end{aligned}$$

Assume the pattern $\prec$, using the ARPG construction from the initial state, is

$$\prec = fw_0; bw_0; bxc_1; fw_1; bw_1; \ldots; bxc_N; fw_N; bw_N.$$

Then, in the $\prec$-encoding of Π, $\mathrm{pre}^{\prec}(A)$ contains, for the actions fw_0, bw_0 and bxc_1, formulas entailing

$$\begin{gathered} fw_0 > 0 \rightarrow (x_0 < L) \wedge (b_0 > 0), \\ fw_0 > 1 \rightarrow (x_0 + (fw_0 - 1) < L), \\ bw_0 > 0 \rightarrow (x_0 + fw_0 > 0) \wedge (b_0 > 0), \\ bw_0 > 1 \rightarrow (x_0 + fw_0 - (bw_0 - 1) > 0), \\ bxc_1 > 0 \rightarrow (x_0 + rt_0 - lf_0 = x_1) \wedge (b_1 + b_0 > 0), \end{gathered}$$

and likewise for all other actions in $\prec$. For each $j \in [1, N]$, the action bxc_j is not eligible for rolling, and thus

$$bxc_j = 0 \vee bxc_j = 1,$$

belongs to $\mathrm{amo}^{\prec}(A)$. Finally, the frame axioms in $\mathrm{frame}^{\prec}(V_B \cup V_N)$, for each $i \in [0, N]$ and $j \in [1, N]$ are:

$$\begin{gathered} x_i' = x_i + fw_i - bw_i, \;\; btd_0' = btd_0, \;\; btd_j' = btd_j \vee bxc_j, \\ b_j' = \mathrm{ITE}(bxc_j > 0, \mathrm{ITE}(bxc_{j-1} > 0, \mathrm{ITE}(\ldots), b_{j-1}), b_j). \end{gathered}$$

As the frame axioms make clear, the $\prec$-encoding allows in a single state transition (i) the multiple consecutive execution of the same action, as in the rolled-up encoding Scala et al. [2016d], and (ii) the combination of multiple even contradictory effects on a same variable by different actions, as in the $R^2\exists$ encoding Bofill et al. [2016].

Assuming the goal is that all the runners have to touch the baton, then the $\prec$-encoding of Π with $n = 1$, is satisfiable. However, if the goal also includes returning the baton to the initial runner r_0, the $\prec$-encoding of Π requires a bound $n = N + 1$. This is because returning the baton from r_N to r_0 necessitates a plan where bxc_j is executed before bxc_{j-1} for $j \in [1, N]$, while their order in the pattern $\prec$ is the opposite. The fact that the defined pattern $\prec$ leads to negative results is not surprising: any simple single pattern is likely to offer limited guidance when the problem demands the non-consecutive execution of the same action multiple times.

5.2 Pushing Numeric Pattern Planning

Consider a numeric planning problem Π. The issue highlighted by the motivating example arises because (i) a simple and complete pattern $\prec$ is computed only once starting from the initial state, and (ii) then exploited at every step $i \in [0, n-1]$ in the $\prec$-encoding of Π with bound n (the formula $\Pi_n^{\prec}$), without considering that:

1. A not empty subset P of the set G of subgoals may have been already satisfied at a step $i < n$.

2. To satisfy the remaining subgoals in $G \setminus P$, it may be (far) better to use a pattern entirely different from the one used to satisfy the subgoals in P.

3. To facilitate the solution of the SMT formula, it may be better to discard the actions in the steps $< i$ that are useless for satisfying the goals in P.

Assuming π is a valid plan, the objective is to find the smallest possible pattern $\prec$ covering π. Ideally, $\prec$ should be the pattern of π. A pattern $\prec$ *covers a plan* π if $\prec$ is a supersequence of the pattern of π. A sequence of actions $\prec$ is the *pattern of a plan* π if $\prec$ is obtained from π by replacing consecutive occurrences of each action a eligible for rolling with a single instance of a. If a pattern $\prec$ covers a valid plan, the formula $\Pi_1^{\prec}$, representing the pattern $\prec$-encoding of Π with bound $n = 1$, is satisfiable.

Theorem 5.1. *Let Π be a numeric planning problem. Let $\prec$ be a pattern covering a valid plan. $\Pi_1^{\prec}$ is satisfiable.*

Proof (hint). Three steps. First, considering π as a pattern, the pattern π-encoding of Π with bound $n = 1$ (Π_1^{π}) is satisfiable. Then, if $\prec_\pi$ is the pattern of π, $\Pi_1^{\prec_\pi}$ is satisfiable. Finally, if $\prec$ is a supersequence of $\prec_\pi$, $\Pi_1^{\prec}$ is satisfiable. □

Of course, we cannot directly exploit the above theorem to find a pattern covering a valid plan. In practice, with ARPG or, more in general, with any ordering on the set A of actions, we just have a simple and complete pattern computed from a given initial state. Such pattern can be arbitrarily extended, but, for effectiveness, this needs to be done with some care, since

1. different patterns, even when one is a permutation of the other, cover different plans, and

2. each newly introduced action in the pattern adds another variable to the encoding, thereby increasing the search space of the SMT solver.

We now show how to symbolically search for a valid plan by iteratively extending and simplifying the initial pattern. The final procedure, called PATTY$_{\text{F}}$, incorporates three ideas:

1. With patterns, in the formula of Eq. 1.2 it is not necessary to duplicate n-times the symbolic transition relation: Given the initially computed pattern $\prec_h$ and a pattern $\prec_g$ initially set of $\prec_h$, we can iterate the procedure of concatenating $\prec_h$ to $\prec_g$ till $\prec_g$ covers a valid plan, i.e., allows reaching a state in which all the subgoals in G are satisfied.

2. In the above outlined procedure, it is not necessary to keep the same $\prec_h$ at each iteration: Given a pattern $\prec_g$ allowing us to reach a state s satisfying a strict subset P of the subgoals in G, we can (i) dynamically recompute the pattern $\prec_h$ whenever we reach a state s satisfying $> |P|$ subgoals, and then (ii) iterate the procedure, exiting when all the subgoals in G are satisfied.

3. In the above outlined procedure, $\prec_g$ allows us to compute a plan π leading to the state s from which $\prec_h$ is recomputed: We can exploit the

plan π and use the pattern of π instead of $\prec_g$, thereby eliminating the actions in $\prec_g$ which are not necessary to reach the state s.

In the following three subsections, we present procedures that exploit the three aforementioned ideas. This allows us to formally state their correctness and completeness. Additionally, the first two procedures correspond to ablation studies of the third, which incorporates all three ideas. The correctness and completeness of the procedures rely on the following theorem. A pattern $\prec$ is *n-complete* if $\prec$ is a supersequence of a pattern obtained by concatenating n simple and complete patterns.

Theorem 5.2. *Let Π be a numeric planning problem having a valid plan of length n. Let $\prec$ be a n-complete pattern. $\Pi_1^{\prec}$ is satisfiable.*

Proof (hint). A plan π of length n is a supersequence of the pattern of π and is a subsequence of any n-complete pattern. Thus, $\prec$ covers π and the thesis follows by Thm. 5.1. □

Given that we are going to present procedures that extend the pattern $\prec$ while keeping $n = 1$ in the formula in Eq. 1.2, we will simply refer to $\Pi_1^{\prec}$, $\mathcal{X}_0$, $\mathcal{A}_0^{\prec}$ and $\mathcal{X}_1$ as $\Pi^{\prec}$, $\mathcal{X}$, $\mathcal{A}^{\prec}$ and $\mathcal{X}'$, respectively.

5.2.1 Concatenating Patterns

Algorithm 5.1 shows the pseudocode for PATTY$_{\text{G}}$, the version of PATTY that incorporates the idea of iteratively concatenating the initially computed pattern, as already presented in Chapter 2. In PATTY$_{\text{G}}$:

1. COMPUTEPATTERN(s, A, G) returns a simple and complete pattern, that in practice we compute using the ARPG construction starting from the state $s = I$. The goal G is passed as a parameter because the ARPG construction can sometimes immediately reveal that the problem Π is not solvable, i.e., that G is not reachable from state s (see Scala et al. [2016b]). To simplify the algorithm, we omit the check for this case.

2. SOLVE$(\Pi^{\prec_g})$ calls an SMT solver which returns a model of the given formula if it is satisfiable, and 0 otherwise.

Algorithm 5.1 PATTY$_{\text{G}}$ algorithm. Input: a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$. Output: a valid plan for Π.

```
1: function PATTY_G(Π)
2:     ≺_g ← COMPUTEPATTERN(I, A, G)
3:     ≺_h ← ≺_g
4:     while (TRUE) do
5:         Π^{≺_g} ← I(X) ∧ T^{≺_g}(X, A^{≺_g}, X') ∧ G(X')
6:         μ ← SOLVE(Π^{≺_g})
7:         if (μ ≠ 0) then return GETPLAN(μ, ≺_g)
8:         end if
9:         ≺_g ← ≺_g; ≺_h
10:    end while
11: end function
```

3. GETPLAN$(\mu, \prec_g)$ returns the sequence of actions ordered as in $\prec_g$, each action a_i repeated $\mu(\mathsf{a}_i)$ times.

In PATTY$_{\text{G}}$, the pattern $\prec_h$ is computed only once in the initial state I and we start considering the pattern $\prec_g$ equal to $\prec_h$. At Line 6 we check for satisfiability of the formula $\Pi^{\prec_g}$, i.e., we check if G can be satisfied considering $\prec_g$. If no model is returned, i.e., if $\Pi^{\prec_g}$ is not satisfiable, we skip to Line 9, and we concatenate $\prec_h$ to $\prec_g$ and we start again from Line 5. The pattern $\prec_g$ is continuously extended, each time concatenating $\prec_h$ to it. By Thm. 5.2, if a valid plan of length n exists, we are guaranteed to find it after at most n iterations and calls to SOLVE$(\Pi^{\prec_g})$. Once a satisfiable model μ is found, a valid plan is returned at Line 7.

For any numeric planning problem Π, PATTY$_{\text{G}}(\Pi)$ is *correct* (any returned plan is valid) and *complete* (if a valid plan exists, PATTY$_{\text{G}}(\Pi)$ will return one).

Theorem 5.3. *Let Π be a numeric planning problem.* PATTY$_{\text{G}}(\Pi)$ *is correct and complete.*

Proof (hint). Correctness follows from Thm. 2.3 and completeness from Thm. 5.2: a valid plan of length n is found by the n-th iteration, when $\prec_g$ becomes n-complete. □

Algorithm 5.2 PATTY$_{\text{H}}$ and PATTY$_{\text{F}}$ algorithms. PATTY$_{\text{H}}$ (resp. PATTY$_{\text{F}}$) includes Line 10 (resp. Line 11) and not Line 11 (resp. Line 10). Input: a numeric planning problem $\Pi = \langle V_B, V_N, A, I, G\rangle$. Output: a valid plan for Π.

```
 1: function PATTY_H(Π)/PATTY_F(Π)
 2:     ≺_g ← ϵ, P ← ∅
 3:     ≺_h ← COMPUTEPATTERN(I, A, G)
 4:     while (TRUE) do
 5:         ≺_f ← ≺_g; ≺_h
 6:         μ ← MAXSOLVE(I(X) ∧ T^{≺_f}(X, A^{≺_f}, X'), G)
 7:         if (|SATG(μ, G)| = |G|) then
 8:           return GETPLAN(μ, ≺_f)
 9:         else if (|SATG(μ, G)| > |P|) then
10:           ≺_g ← ≺_f // if PATTY_H
11:           ≺_g ← SIMPLIFY(GETPLAN(μ, ≺_f)) // if PATTY_F
12:           P ← SATG(μ, G)
13:           s ← GETSTATE(s, GETPLAN(μ, ≺_f))
14:           ≺_h ← COMPUTEPATTERN(s, A, G)
15:         else
16:           ≺_g ← ≺_f
17:         end if
18:     end while
19: end function
```

5.2.2 Changing the Pattern During the Search

As the example makes clear, having a single pattern may have a dramatic impact on the number of iterations and calls to the SMT solver. In our example, the pattern computed from the initial state allows finding a plan satisfying $|G-1|$ out of the $|G|$ subgoals in the first iteration, but we struggle to satisfy the last subgoal: once the baton is being held by r_N, any pattern (even a random one) would allow finding a plan for delivering the baton back to r_0 in a number of iterations $\leq N$. Algorithm 5.2 shows the pseudocode for PATTY$_{\text{H}}$, the version of PATTY that incorporates the idea to dynamically update the pattern by recomputing it whenever new subgoals are achieved. In PATTY$_{\text{H}}$,

1. MAXSOLVE$(\mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec_f}(\mathcal{X},\mathcal{A}^{\prec_f},\mathcal{X}'), G)$ calls a MAX-SMT solver returning an assignment satisfying $\mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec_f}(\mathcal{X},\mathcal{A}^{\prec_f},\mathcal{X}')$ and the maximum number of subgoals in G.[2]

[2]Notice that $\mathcal{I}(\mathcal{X}) \wedge \mathcal{T}^{\prec_f}(\mathcal{X},\mathcal{A}^{\prec_f},\mathcal{X}')$ is by construction satisfied, e.g., by the assignment setting all the action variables to 0.

2. GETSTATE(s, π) returns the state resulting from the execution of the sequence π of actions in state s.

3. SATG(μ, G) returns the set of subgoals in G satisfied by the assignment μ.

In PATTY$_{\text{H}}$, before the search starts, we assign $\prec_g$ to the empty pattern, the set P (meant to contain the subset of subgoals that can be satisfied with $\prec_g$) to the empty set, and we compute an initial pattern $\prec_h$ from the initial state. Then,

1. we set the pattern $\prec_f$ used for the search to $\prec_g; \prec_h$ (Line 5) and then check whether all the subgoals in G are satisfied (Line 7) and,

2. if not, we check whether $\prec_f$ allows satisfying at least one more subgoal (Line 9), in which case we (i) set $\prec_g$ to $\prec_f$, (ii) update the set P to the new subset of satisfied subgoals, (iii) recompute the pattern $\prec_h$ considering s as initial state, and (iv) restart the loop thereby concatenating the newly computed $\prec_h$ at the next iteration,

3. otherwise, (Line 16) we set $\prec_g$ to $\prec_f$, thereby concatenating $\prec_h$ once more at the next iteration.

For any numeric planning problem Π, PATTY$_{\text{H}}(\Pi)$ is correct and complete.

Theorem 5.4. *Let Π be a numeric planning problem.* PATTY$_{\text{H}}(\Pi)$ *is correct and complete.*

Proof (hint). Correctness follows from Thm. 2.3. Completeness follows from Thm. 5.2: at each iteration, a complete pattern is concatenated to $\prec_f$ and a valid plan of length n is found at most at the n-th iteration, since at that point $\prec_f$ is n-complete and all the subgoals in G can be satisfied. □

5.2.3 Simplifying the Pattern During the Search

In our example, though PATTY$_{\text{H}}$ can find a valid plan with just two iterations, it is still possible to further optimize it by simplifying the pattern $\prec_f$ that led to the state s in which a new subgoal has been satisfied. Indeed, we can

	Coverage (%)				Time (s)				n - Calls to SMT solver				$\lvert\mathcal{X}\cup\mathcal{A}^{\prec}\cup\mathcal{X}'\rvert$				$\lvert\mathcal{T}^{\prec}(\mathcal{X},\mathcal{A}^{\prec},\mathcal{X}')\rvert$			
Domain	P_O	P_G	P_H	P_F	P_O	P_G	P_H	P_F	P_O	P_G	P_H	P_F	P_O	P_G	P_H	P_F	P_O	P_G	P_H	P_F
BLGRP (S)	**100**	**100**	**100**	**100**	**1.6**	**1.6**	1.8	1.8	**1.0**	**1.0**	**1.0**	**1.0**	**182**	**182**	**182**	**182**	**448**	**448**	**448**	**448**
CNT (S)	**100**	**100**	**100**	**100**	**0.9**	**0.9**	**0.9**	**0.9**	**1.0**	**1.0**	**1.0**	**1.0**	**92**	**92**	**92**	**92**	**228**	**228**	**228**	**228**
CNT (L)	**100**	**100**	**100**	**100**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**96**	**96**	**96**	**96**	**202**	**202**	**202**	**202**
DEL (S)	15	15	25	**40**	255.6	255.6	241.8	**191.1**	**3.3**	**3.3**	**3.3**	**3.3**	1082	809	809	**507**	3277	2946	2946	**1736**
DRN (S)	15	15	15	**80**	255.3	255.3	255.3	**81.9**	**5.7**	**5.7**	**5.7**	8.3	301	137	137	**95**	778	322	322	**194**
EXP (S)	**15**	**15**	**15**	**15**	**262.0**	**262.0**	267.3	266.7	**7.7**	**7.7**	8.0	8.3	778	420	434	**410**	1814	1333	1383	**1299**
FARM (S)	**100**	**100**	**100**	**100**	**0.9**	**0.9**	**0.9**	**0.9**	**1.0**	**1.0**	**1.0**	**1.0**	**50**	**50**	**50**	**50**	**107**	**107**	**107**	**107**
FARM (L)	**100**	**100**	**100**	**100**	6.2	5.4	2.1	**2.0**	**1.0**	**1.0**	**1.0**	**1.0**	**64**	**64**	**64**	**64**	**129**	**129**	**129**	**129**
HPWR (S)	**100**	**100**	**100**	**100**	**11.6**	12.2	12.0	12.1	**1.0**	**1.0**	**1.0**	**1.0**	**352**	**352**	**352**	**352**	**725**	**725**	**725**	**725**
MPRIME (S)	**50**	**50**	**50**	**50**	**154.8**	155.8	155.5	157.1	**2.0**	**2.0**	**2.0**	2.2	1570	**1318**	**1318**	1320	4916	**4645**	**4645**	4652
PATHM (S)	**100**	**100**	**100**	**100**	**6.1**	6.2	6.5	6.5	**1.0**	**1.0**	**1.0**	**1.0**	**3079**	**3079**	**3079**	**3079**	**4834**	**4834**	**4834**	**4834**
PLWAT (S)	30	30	30	**100**	220.2	224.0	224.4	**10.2**	**8.2**	**8.2**	8.3	10.7	623	343	348	**237**	1718	915	931	**598**
RVR (S)	50	55	50	**65**	156.1	150.3	165.3	**143.6**	**2.1**	**2.1**	2.3	3.1	1008	562	651	**499**	2251	1847	2198	**1597**
SAT (S)	25	30	25	**35**	250.7	242.1	235.4	**209.0**	2.8	2.8	**2.5**	3.2	2061	1441	1332	**790**	5761	4644	4242	**2187**
SAIL (S)	**100**	**100**	**100**	**100**	1.1	1.1	1.3	**0.9**	**3.3**	**3.3**	**3.3**	**3.3**	163	117	117	**72**	340	254	254	**134**
SAIL (L)	95	95	**100**	**100**	22.2	21.1	4.5	**1.8**	**1.5**	**1.5**	**1.5**	**1.5**	72	65	65	**61**	178	165	165	**151**
SGR (S)	**100**	**100**	**100**	**100**	9.4	8.8	9.5	**6.1**	**3.1**	**3.1**	3.2	4.0	1401	920	931	**725**	3465	2792	2830	**2101**
TPP (L)	**10**	**10**	**10**	**10**	270.5	272.7	270.4	**270.1**	2.5	2.5	**2.0**	**2.0**	452	288	251	**196**	1052	703	588	**408**
ZENO (S)	**55**	**55**	**55**	**55**	137.3	138.4	138.2	**136.6**	**1.6**	**1.6**	**1.6**	1.7	603	515	515	**376**	1855	1707	1707	**1147**
Total	**13**	**13**	**14**	**19**	**8**	**5**	**3**	**14**	**17**	**17**	**15**	**11**	**7**	**8**	**8**	**18**	**7**	**8**	**8**	**18**

Table 5.1 Comparative analysis between PATTY_O, PATTY_G, PATTY_H and PATTY_F. The labels (S) and (L) indicate if the numeric planning task is Simple or Linear, according to the IPC definition.

exploit the plan π that led to such state s and use the pattern of π. The pattern of π will be a subsequence of $\prec_f$ and thus it may not allow reaching all the states that are reachable with $\prec_f$. On the other hand, by reducing the number of variables in the encoding, the task of the SMT solver becomes easier.

Algorithm 5.2 shows the pseudocode for PATTY_F, the version of PATTY, that incorporates the idea of simplifying the pattern. In this case, we have at Line 11 that $\prec_g$ assumes the value of $\text{SIMPLIFY}(\pi)$, with $\pi = \text{GETPLAN}(\mu, \prec_f)$. $\text{SIMPLIFY}(\pi)$ returns the pattern of π. We call PATTY_F this version of PATTY, to reflect the intuition that the pattern construction relies on the subgoals in G, with its initial part allowing to reach (just) the state in which the new pattern is computed. For any numeric planning problem Π, $\text{PATTY}_\text{F}(\Pi)$ is correct and complete.

Theorem 5.5. *Let Π be a numeric planning problem. $\text{PATTY}_\text{F}(\Pi)$ is correct and complete.*

Proof (hint). Correctness follows from Thm. 2.3. For completeness, consider a valid plan of length n. Each time Line 16 is executed, the last computed complete pattern $\prec_h$ is concatenated to $\prec_f$ at Line 5, while each time Line 11 is executed $\prec_f$ is simplified. However, Line 11 can be executed at most $(|G|-1)$ times, each time after at most $(n-1)$ iterations between two consecutive executions. Thus, a valid plan will be found at most at the $p+n$-th iteration, where in the worst case $p = (|G|-1) \times (n-1)$. □

5.3 PATTY$_{\text{F}}$ Behaviour on the Motivating Example

Consider the problem Π in the motivating example in which the baton has to return to the hands of runner r_0, and assume COMPUTEPATTERN(I, A, G) returns the pattern:

$$\prec_h = fw_0; bw_0; bxc_1; fw_1; bw_1; \ldots; bxc_N; fw_N; bw_N.$$

PATTY$_{\text{G}}$ at each iteration $i \in [1, N]$ extends the initial pattern $\prec_g$ by considering $\prec_g = \prec_h; \prec_h^i$ in which $\prec_h^i$ denotes the pattern $\prec_h$ concatenated i times. When $i = N$, $\prec_h^i$ is N-complete, $\prec_h^N$ covers the plan for returning the baton from r_{N+1} back to r_0, and $\Pi^{\prec_g}$ is satisfiable.

PATTY$_{\text{H}}$ starts with $\prec_f = \prec_h$ and at the first iteration it will satisfy the subgoals of having the baton touched by all runners by reaching a state s satisfying, for each $i \in [0, N)$, $x_i = L \times (i + 1)$, $x_N \in [L \times N, L \times (N + 1)]$, $b_0 = \ldots = b_{N-1} = 0$, $b_N = 1$ and $btd_0 = \ldots = btd_N = 1$. Assuming that $s(x_N) = L \times (N + 1)$, the new pattern $\prec_h'$ computed from s is:

$$\prec_h' = bw_N; bxc_N; fw_N; \ldots; bw_1; bxc_1; fw_1; bw_0; fw_0,$$

and PATTY$_{\text{H}}$ will find a valid plan with $\prec_f = \prec_h; \prec_h'$.

PATTY$_{\text{F}}$ starts with $\prec_f = \prec_h$ as PATTY$_{\text{H}}$. Differently from PATTY$_{\text{H}}$, it will compute the pattern $\prec_\pi$ of the plan π that led to the state s in which $\prec_h'$ was computed, i.e.,

$$\prec_\pi = fw_0; bxc_1; fw_1; \ldots; bxc_N; fw_N$$

and PATTY$_{\text{F}}$ will find a valid plan with $\prec_f = \prec_\pi; \prec_h'$.

Every pattern computed by COMPUTEPATTERN(s, A, G) contains $3N + 2$ actions, and PATTY$_{\text{G}}$, PATTY$_{\text{H}}$ and PATTY$_{\text{F}}$ are able to find a valid plan when considering a pattern with $N \times (3N + 2)$, $2 \times (3N + 2) = 6N + 4$ and $(2N + 1) + (3N + 2) = 5N + 3$ actions respectively.

Domain	Coverage (%)				Time (s)			
	P_F	ENHSP	FF	NFD	P_F	ENHSP	FF	NFD
BLOCKGROUPING (S)	**100**	**100**	10	-	**1.8**	48.0	270.2	-
COUNTERS (S)	**100**	**100**	60	50	**1.0**	6.9	129.0	149.1
COUNTERS (L)	**100**	45	40	25	**1.0**	180.5	180.0	225.4
DELIVERY (S)	40	65	**95**	45	191.3	121.2	**48.5**	165.2
DRONE (S)	80	**85**	10	80	82.5	**59.9**	270.0	65.4
EXPEDITION (S)	**15**	10	-	**15**	267.5	270.3	-	**253.7**
FARMLAND (S)	**100**	**100**	35	75	1.0	**0.7**	206.8	85.5
FARMLAND (L)	**100**	75	75	55	**2.2**	96.8	90.7	151.7
HYDROPOWER (S)	**100**	10	5	5	**12.7**	270.4	285.0	285.1
MPRIME (S)	50	**85**	80	65	155.6	49.7	**47.5**	133.6
PATHWAYSMETRIC (S)	**100**	60	50	5	**6.5**	133.9	154.9	285.0
PLANTWATERING (S)	**100**	**100**	10	60	10.2	**9.8**	276.5	185.2
ROVER (S)	**65**	35	50	20	**128.8**	204.5	142.1	241.0
SAILING (S)	**100**	**100**	5	50	**1.0**	1.5	285.0	150.3
SAILING (L)	**100**	20	40	70	**1.9**	241.2	182.9	109.4
SATELLITE (S)	**35**	30	20	20	**209.6**	222.6	229.4	242.2
SUGAR (S)	**100**	95	65	25	**6.1**	23.7	119.9	232.9
TPP (L)	10	**20**	10	10	270.1	**244.3**	268.4	270.0
ZENOTRAVEL (S)	55	**100**	55	45	136.7	**20.4**	135.0	178.5
Total	**14**	**9**	**1**	**1**	**11**	**5**	**2**	**1**

Table 5.2 Comparative analysis of PATTY_F and the search based planners ENHSP, METRICFF and NFD. A "-" means that no problem in the domain was solved by the planner.

5.4 Experimental Results

For the experiments, we adopted the settings used in Chapter 2, and thus (i) we considered all 19 domains and 20 problems per domain from that paper, which included the domains of the 2023 Numeric IPC where at least one planner solved a problem Taitler et al. [2024]; (ii) we used the settings of the Agile Track of the IPC, and thus the systems had a time limit of 5 minutes on an Intel Xeon Platinum 8000 3.1GHz with 8 GB of RAM; and (iii) we used Z3 v4.8.7 de Moura and Bjørner [2008] for computing the model (if any) satisfying the given set of SMT assertions (representing the hard constraints of the encodings) and also the maximum number of soft assertions (representing the subgoals in PATTY_H and PATTY_F).

Table 5.1 compares the original version of PATTY of Chapter 2 (renamed PATTY_O), and PATTY_G, PATTY_H and PATTY_F on the 12 domains for which at least one problem required more than one call to the SMT solver. Indeed, on the 7×20 problems that were solved using just the initially computed pattern, (i) there are no differences between PATTY_O and PATTY_G, and between PATTY_H and PATTY_F, and (ii) the possible differences between PATTY_O/PATTY_G and PATTY_H/PATTY_F (due to the latter maximizing the number of satisfied subgoals) turned out to be almost always insignificant.

In the first three subtables of Table 5.1, we show: the name of the domain (subtable Domain); the percentage of solved instances (subtable Coverage); the average time to find a solution, counting the time limit when the solution could not be found (subtable Time). In the last three subtables, we considered only the problems solved by all the planners and show the average number n of calls to the SMT solver (subtable n - Calls to the SMT solver); the number of variables (subtable $|\mathcal{X} \cup \mathcal{A}^{\prec} \cup \mathcal{X}'|$) and assertions (subtable $|\mathcal{T}(\mathcal{X} \cup \mathcal{A}^{\prec} \cup \mathcal{X}')|$) of the encoding when a solution is found.

Looking at the performance results, the first observation is that PATTY_F has by far the best results: on all the domains, PATTY_F has the highest coverage, meaning that it can solve the highest number of problems in each domain. Further, PATTY_F has higher coverage on 6 out of the 10 domains where PATTY_O did not solve all problems. Then, by comparing PATTY_F vs PATTY_H, and PATTY_H vs PATTY_G/PATTY_O, the second observation is that the simplification of the pattern plays a major role in PATTY_F positive performance.

Examining the subtable with the number n of calls made to the SMT solver, we observe that PATTY_O and PATTY_G yield identical values, while the count for PATTY_H is never greater than that of PATTY_F. No strict relationship emerges when comparing PATTY_O/PATTY_G with PATTY_H and PATTY_F. These results align with the theoretical findings. As expected, the total number of variables and the total number of assertions utilized in the final call to the SMT solver is typically lower for PATTY_F than for PATTY_O/PATTY_G/PATTY_H, even on domains in which PATTY_F needs more iterations/calls to the SMT solver. The number of variables and assertions used by PATTY_O is always greater than those that are used by PATTY_G, as PATTY_O uses state variables representing intermediate states, along with the corresponding assertions. Conversely, the average

number of variables per assertion is consistently higher for PATTY$_G$ than for PATTY$_O$ being 16.9 and 7.3 respectively. Despite these differences, PATTY$_O$ and PATTY$_G$ exhibit very similar performance.

Table 5.2 compares PATTY$_F$ with the planners that performed better than PATTY$_O$ on at least one domain. Thus, it includes the search-based planners ENHSP Scala et al. [2016b], METRICFF Hoffmann [2003] and NUMERIC-FASTDOWNWARD (NFD) Kuroiwa et al. [2022], and does not include the symbolic planners SPRINGROLL Scala et al. [2016d] and OMTPLAN Leofante et al. [2020]. We remind that NFD participated and ranked first in the last IPC. The planner ENHSP has been run three times using the `sat-hadd`, `sat-aibr` and `sat-hmrphj` settings, and for each domain the best result is reported. Considering all the planners in Table 5.2, PATTY$_F$ has the highest coverage on 14 domains, compared to the 9 by ENHSP, and 1 for both FF and NFD. Considering the timings, PATTY$_F$/ENHSP/FF/NFD have the lowest values on 11/5/2/1 domains, and can solve 290/247/143/144 of the 19×20 problems we consider, respectively.

Finally, we did some experiments with a 30-minute time-limit, obtaining the same overall picture.

Chapter 6

Discussion: SPP in Applications and Hybrid Planning

In this section, we discuss how the SPP approach presented in this thesis can provide beneficial results when applied to an application domain and to the hybrid flavour of planning. The application domain considered is the *In-Station Train Dispatching Problem*, concerning of planning the movements of trains inside a railway station. The work presented in this chapter is just a discussion on the benefits that the SPP approach could bring in this flavour and application domain. The candidate is currently investigating this approaches and theoretical and empirical analyses are still ongoing. Still, we wanted to highlight some preliminary formulation and considerations.

6.1 In-Station Train Dispatching

To characterize the *In-Station Train Dispatching* (INSTRADI) problem, we first present the objects characterizing the domain (i.e., stations, trains and nominal timetable), then what is a (valid) station state and how commands can be used to evolve the state of the station. Next, we present the concept of dispatchment, i.e. a timed sequence of commands' sets, denoting (valid) transitions between (valid) station states. Afterward, we will introduce the notion of a forecast, i.e., an estimator of the arrival of the trains at the station, their running and stop times. Employing the forecast, we will

define the concept of a forecasted dispatchment, i.e., a dispatchment that follows the timings estimated in the forecast. Finally, we will discuss when a valid and forecasted dispatchment can be considered a solution to the INSTRADI problem. For each of these concepts, we will firstly provide an intuitive description followed by a more formal characterization, helpful in guaranteeing the correctness and completeness of the approach.

6.1.1 Stations, Trains and Nominal Timetable

Station A station, as exemplified in Figure 6.1, is composed of signals, track circuits, platforms and switches. A *signal* is a visual display device that gives instructions to the driver of a train on whether to stop or proceed in its path. A driver can stop in the station only in the proximity of a signal. Signals are physically placed near track circuits and are oriented with respect to them. Figure 6.1 shows a station with 17 signals, the orientation of the signal is depicted by the direction of the red and green colors. *Track circuits* are the smallest logical units forming rails. Usually, track circuits are hundreds of meters long and allow for the detection of a train moving on top of them.[1] In Figure 6.1, track circuits are represented by segments ended by two perpendicular small lines, see, e.g., the track circuit 101 in the bottom-left part of the Figure. A *platform* is a set of track circuits in which trains can stop to embark/disembark passengers or load/unload goods. A platform is delimited by two signals, called *left* and *right*, both oriented towards each other, instructing the driver when to start/leave from the platform when departing towards the right or the left of the platform (looking at the station schema like the one in Figure 6.1), respectively.[2] In Figure 6.1, the platform II is composed of the track circuits 114, 115, 116 and is delimited by signals 22 and 42. *Switches* are mechanical components that allow trains to deviate their movement towards another track circuit. In the bottom-left of Figure 6.1, for example, the switch d1 allows moving from the track circuit 102 to either the track circuits 111 or 103. A switch has two states: *through* and *diverge*.

[1] The name “circuit" comes from the fact that each rail is connected to a pole of a battery and a relay. When a train moves on top of a track circuit, its axles close the circuit and make current flow through the relay, signalling the presence of the train.

[2] Of course, for terminal platforms, the signal corresponding to the blocked direction is always red.

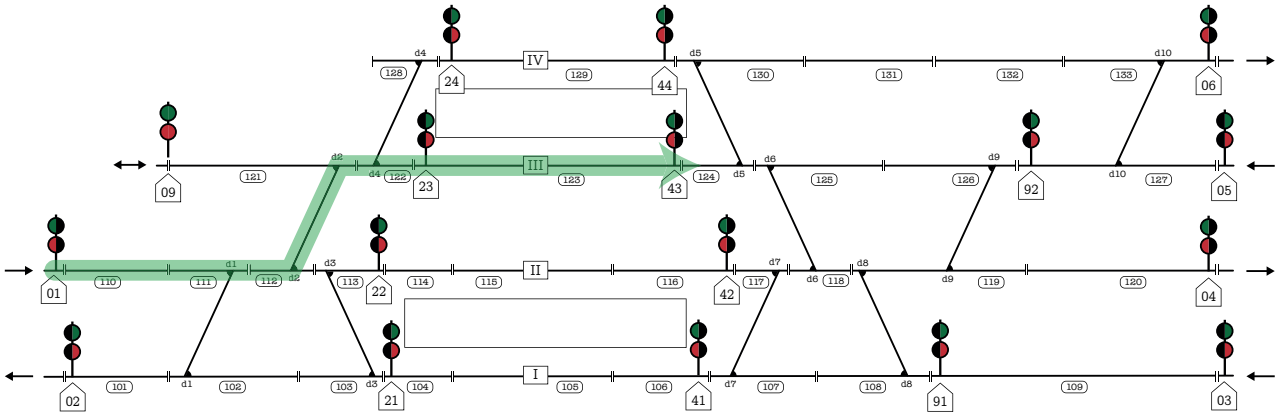

Fig. 6.1 A schema of a railway station with 4 platforms, 4 entry points and 4 exit points. Signals are oriented based on the direction of the red and green colors. The route 01-43 is highlighted in green.

When a train is arriving from 101 and moves in 102, the train will move to 103 if the switch is in the *through* state or to 111 if the switch is in the *diverge* state. Switches are directional and in the figure, the direction of each switch is given by their associated angle, and the movement associated with the *diverge* state can happen only when it is acute regarding the direction of the train movement. For instance, if a train is arriving from 103 it cannot move via d1 from 102 to 111 but it can only proceed to 101.

How trains enter and exit, in and from a station, is dictated by the possible entry and exit points of the station. *Entry and exit points* are signals that delimit the station and control the flow inbound and outbound of the station. Figure 6.1 shows 4 entry points (01, 03, 05, 09) and 4 exit points (02, 04, 06, 09).[3] Since in 09 a train can both enter and exit, the signal is oriented both ways. A *route* is a sequence of track circuits connecting two different signals having the same orientation. In Figure 6.1, the route 01-43 consists of the track circuits 110, 111, 112, 121, 122, 123 and connects the signal 01 with the signal 43. The route implicitly defines the states of the switches it traverses. For example, for the route 01-43 to be run, the switches d1, d2, d4 must be in the state *through*, *diverge* and *through*, respectively. A route can be used to move a train inside a platform and, for such reason, it must contain all the track circuits of the platform. For example, in Figure

[3]Note that, on our representation of a station in Figure 6.1, entry and exit signals are named 0X, platform's signals are always named 2X and 4X where X is the cardinal number of the platform, and intermediate signals, i.e. the remaining signals, are named 9X.

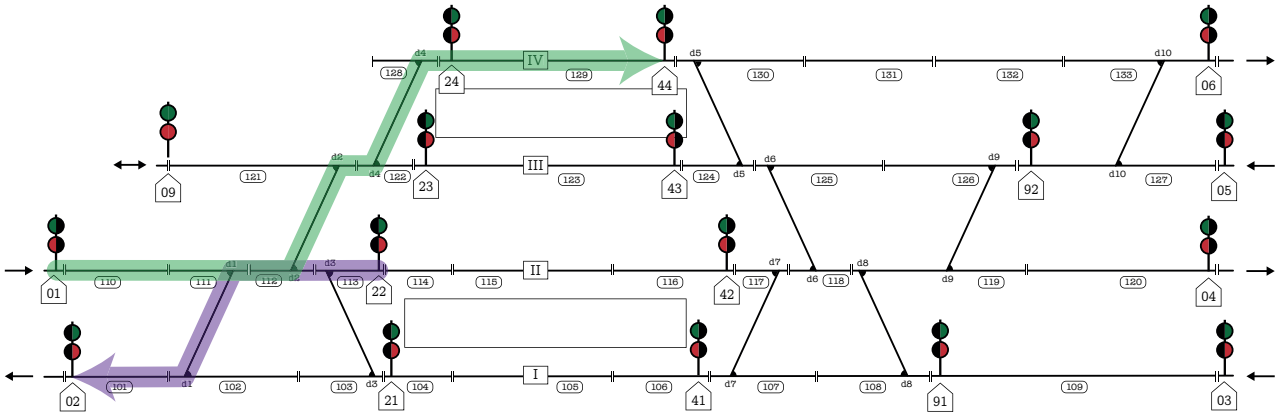

Fig. 6.2 An example of two incompatible routes 01-44 and 22-02 which share the track circuits 112 and 111.

6.2 the route 01-44 is used to move a train on a platform IV and thus the route includes a track segment 129. Of course, the description of a station is expected to represent its layout faithfully, i.e., to represent (at the given level of abstraction) a “real station". For instance, we assume that a station – when seen as a graph whose track circuits are the edges– is planar. A more concrete assumption that we make is (i) that in a track circuit there is at most one red/green signal for each given direction, or (ii) that two different track circuits are connected by at most one switch (as a consequence, for instance, it is not possible to have a train starting from a circuit, then switching from another circuit and then again back to the starting one). All these (and many others) assumptions are left implicit and assumed to hold thanks to the explicit specification of the possible routes each train can take to move inside the station. When multiple trains are inside the station, to guarantee the safety of the movements of each train, railway experts define a table of pairwise *incompatible* routes. The incompatibility between two routes can arise from a share of track circuits, i.e., route 22-02 and route 01-44 in Figure 6.2 which share track circuits 111 and 112, or can be declared incompatible by domain experts, due to the sharing of switches that must be configured in conflicting states (i.e. *through* and *diverge*) for the movement of trains in the two routes to happen simultaneously.

Trains The trains which move inside a railway station are usually associated with an ID, which identifies their trip through several stations, and an objective. For example, in Italy, the ID 8620 identifies a train moving every day from

Roma Termini to *Milano Centrale* transiting from *Camogli* and stopping at *Genova Brignole*. A train can have one of four objectives:

1. *origin* if the modelled station is the first in the train's trip and thus, in the modelled station, the train will start from a platform and leave the station through an exit point (e.g. 8620 in *Roma Termini*),
2. *destination* if the modelled station is the last station on its trip, thus entering the station from an entry point and ending its trip stopping on a platform (e.g. 8620 in *Milano Centrale*),
3. *stop* if the modelled station is an intermediate step on its trip, and it is required to stop at a platform (with a sidewalk able to contain the full length of the train) to let passengers embark/disembark (e.g. 8620 in *Genova Brignole*), and
4. *transit* if the train will pass through the modelled station, but it will not stop at a platform (e.g. 8620 in *Camogli*).

Nominal Timetable A *nominal timetable* specifies when each train is expected to pass, arrive and/or depart from a platform. We suppose to have this information in real time, and thus the timetable indicates the number of seconds, relative to now, in which the train should arrive or depart. If the train is (or should have) already arrived or departed, we indicate the corresponding time with a negative number.

6.1.2 States and Commands

After having described the physical and virtual components of a station, we now focus on describing the movements of trains inside a station. For example, let's take the red train stopped on the platform II in Figure 6.3. The train is *locking* the track circuits 114, 115 and 116 which compose platform II (i.e., signalling they are reserved for that train). Before the signal 22, (i.e., $\mathrm{left}(\mathsf{II})$) is given the green, authorizing the train to move away from the platform II, the train must *reserve* a route, signalling the desired path of track circuits to reach a destination signal. To reserve a route, all the track circuits of the route and the ones denoted by the incompatibilities must be free. This reservation

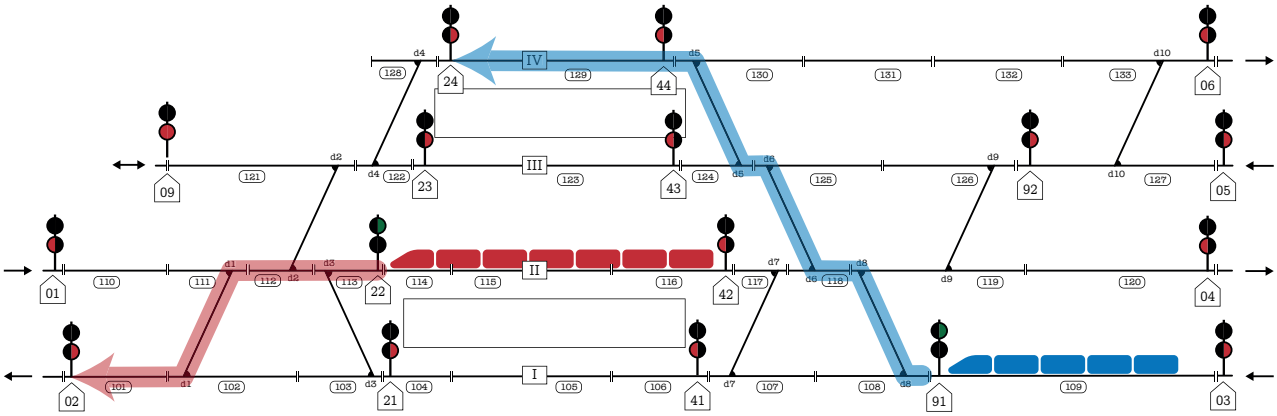

Fig. 6.3 An example of the movement of two trains, red and blue. The train red is stopping on the platform II with its locomotive directed towards the signal 22. The route 22-02 allows the train to exit the station from the exit point 02. Train blue has completed running through 03-91. Route 91-24 allows reaching platform IV.

automatically sets the switches in the correct orientation to perform the movement and locks all the track circuits of the route, avoiding any other train reserving an incompatible route. After reserving the route, the train can *move* through the route. While moving, the train frees the track circuits it passes with its rear axles. When this happens, the track circuits are *unlocked*, becoming available again, thus allowing other trains to reserve other routes containing them. For example, in Figure 6.3, if the red needs to leave the station from signal 02, we have to use the route 22-02. Firstly, we have to assure that no other train is locking track circuits 113, 112, 111, 102, and 101 and none of the other track circuits of routes deemed incompatible with 22-02. and then we can reserve the route and lock all its track circuits, positioning the switches d3, d2 and d1 to *through*, *through* and *diverge*, respectively, to allow for the desired movement. When the train moves through the route, after having completed it with its rear axles, it starts to unlock the track circuits, starting from 110.

For simplicity, we don't model routes outside entry and exit points. When a train has completed a route which leads to an exit point, we simply unlock the last track circuit of the route and signal that the train has left the station. Similarly, when instead arrives from an entry point, we start reserving the first route of the station, not caring about the track circuits locked outside the station.

When a train has completed running a route and has reached an intermediate signals inside the station (e.g., 91 in Figure 6.3) it can perform an *overlap* between two routes, meaning that, for a certain amount of time, the train is on both routes at the same time. For example, let's take the train blue in Figure 6.3. The train has completed the route 03-91 and, since it needs to reach the platform IV, it will first reserve and then move into route 91-24 and, for a certain amount of time, overlapping both routes. To ensure safety, while the overlap is performed, the last track circuit of the originating route is kept locked.

Station State A *station state* denotes, at any given time and for each train, the track circuits it has locked, the routes it has reserved and/or occupied, and which platform it is possibly occupying, or it has possibly occupied.

Commands After presenting the definition of a (valid) station state, we now introduce *commands*. A command is an instruction provided to the physical and virtual components of the station and to the trains, which can change the station state in which they are applied. The commands can be of various types: they can (i) lock or unlock track circuits, (ii) reserve/release routes to/from trains, (iii) instruct a train to move into a route, (iv) signal that a route is no longer ran by a train, (v) signal that a train has started occupying a platform and stopping on it and (vi) signal that the train has completed stopping and can free the platform. A command is atomic on a specific track circuit/route/platform, but, in normal operation, they are provided to the system in sets, called *commands' sets*, to govern the multiple track circuits/routes/platforms involved in the movement of a train.

6.1.3 Forecast

The simulation of the actual timings of the movements of trains inside the station would require the characterization of the length of track circuits, platforms and trains, as well as a complex definition of the variation of the speed of the trains and how they accelerate and decelerate inside the station. Moreover, to model the time required for the embarkation/disembarkation

of passengers, one would need to keep track of the number of tickets sold and the number of passengers inside the train. To understand when a train actually arrives at the station, which could be ahead or behind than what is scheduled in the timetable, we would also require having an idea of where the train is outside the station. This could become very complex to manage and to deal with. Instead, in this chapter, we suppose to be provided with a *forecast system* able to estimate, for each train, the running times of track circuits and routes, the time it takes to overlap between two routes, the stopping time at platforms and the estimated arrival of trains at entry points. With such a tool, we do not need to deal with lengths, speeds, passengers nor number of tickets, but we can extract, directly from the forecast system, the necessary times. Of course, this complexity is now moved to the forecast system, but, having it in a separate module, allows us to deal with the granularity and the closeness to the reality as a different and separate research topic. In fact, one could find the times of the forecast system simply by a statistical analysis of the timings of the past (e.g., a train always takes $45s$ to run through 22-02) or perhaps clustering it based on information not directly modelled in our work like weather (e.g., when it rains it takes $60s$ to run through 22-02) or seasonality (e.g, in the summer it takes 1 minute more to embark/disembark all passengers). The system could employ simple statistical methods or very cutting-edge Machine Learning technologies. The interested reader can find a description of such forecast systems in Boleto et al. [2021]. Thus, in our system, we take this forecast for each single train as an input and are not interested in how the times are actually computed.

6.1.4 The Dispatchment

After having defined what are a station state and a command's set, we can now define a *dispatchment* as a sequence of command's set. Given an initial station state, applying the dispatchment consists in sequentially applying the command's set of the dispatchment starting from the initial station state. The dispatchment's validity thus depends on the validity of the commands' sets and the validity of all the station states produced by applying the commands' sets sequentially. In the previous section, we saw how a forecast system is used in our model to manage the timings of the movements of the trains

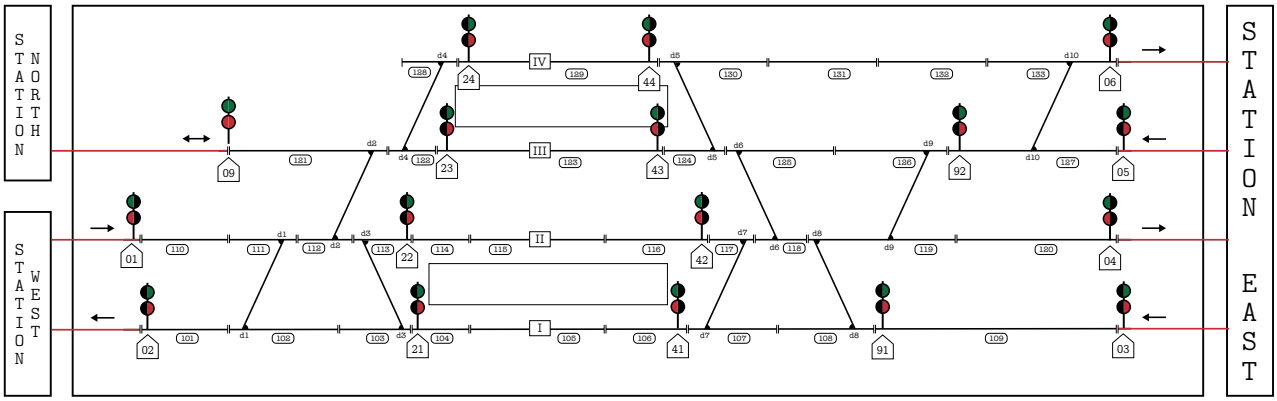

Fig. 6.4 Connections of the modelled station with other stations. Red lines represent uninterrupted and switch-free tracks outside the station, directly connecting the entry/exit points of the modelled station with entry/exit points of the other stations.

inside the station. The concept of validity of a dispatchment, in fact, only checks that the commands and the associated states of the dispatchment are valid, but there is no check on the timings in which these commands are executed. We thus introduce the concept of a *forecasted* dispatchment, to denote a dispatchment which respects the timings between the actions as provided by the forecast.

Exit order The station's entry/exit points are connected to other stations' entry/exit points via track lines called *line points*. Figure 6.4 shows an example of the connections of the modelled station with other stations. Red lines represent uninterrupted and switch-free tracks connecting to the other station. As it can be noted, between the modelled station and *Station North*, only one line exists, called *simple line*, and so trains must proceed in an alternate fashion between the two stations.[4] Between *Station West* and the modelled station, two tracks exist, and thus it is a called *double line*, but an imposed direction of travel must be respected for each track, disallowing trains to run in parallel towards the same direction[5]. Between the modelled station and *Station East* instead, four tracks exist (or two double lines), two

[4]This is common between rural stations which have few trains per hour.

[5]In Italy, for example, according to Rete Ferroviaria Italiana (RFI) [2023], out of 12.205 km of electrified line points, 4.547 km are simple line points and 7.658 km are double line points.

for each direction, allowing trains to move from or to the station in parallel.[6] These track lines are uninterrupted and do not present switches between the entry/exit points of the stations, and for this reason, the only overtake between trains can happen inside the stations. For this reason, based on the type of the train, (e.g., high-speed, regional, freight, etc.) an order must always be respected, avoiding any fast-moving train lagging behind a slower one, and thus imposing that, for example, a high-speed train always leave the station before any other slower (e.g., regional, freight) train.

In-Station Train Dispatching We are now in the position to define the *In-Station Train Dispatching* (INSTRADI) problem. We are provided with (i) a description of the station with its physical and virtual components, (ii) the list of trains which are inside the station or that will shortly arrive, (iii) a *nominal timetable*, that indicates when trains arrive and depart from a platform in the station, (iv) the current state of the station, denoting the trains inside the station, the routes they reserve, the track circuits they lock and platforms they are occupying or have occupied, (v) a forecast system, and (vi) an exit order for the trains. We are tasked to find a dispatchment, which is valid w.r.t. the initial state and forecasted by the forecast, which guarantees that

1. if a train has to stop on a platform, it is never the case that the train leaves the platform before the time specified in the nominal timetable,
2. the train respects its objectives, i.e.
   (a) if the train is of type *stop*, *transit* or *origin*, it must have run through the station and exited it,
   (b) if the train is of type *destination*, the dispatchment must end with the train on the platform,
   (c) if the train is of type *stop*, the train must have stopped on the platform,
3. each pair of trains exiting the station respects the provided exit order.

[6] This case is more of an exception to the rule. Usually, multiple double lines, spanning only a few kilometres, are in place only between stations of the same large city, to better manage the flow of traffic in an urban setting.

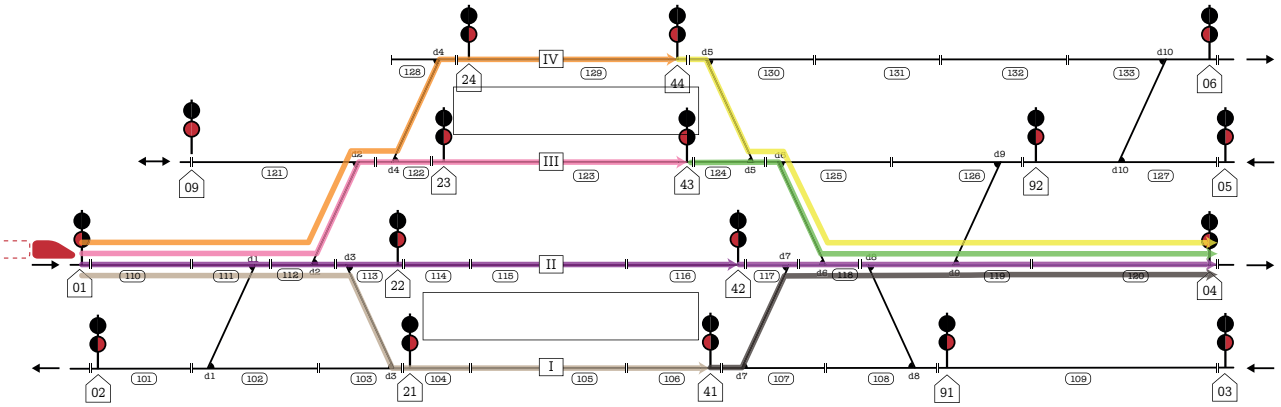

Fig. 6.5 All the routes that connect the entry point 01 with the exit point 04. They are 01-41, 01-42, 01-43, 01-44, 41-04, 42-04, 43-04, 44-05.

6.1.5 SPP for the INSTRADI Problem

Tackling the INSTRADI problem via planning approaches has already been proved very effective in one of the candidate's work [Cardellini et al., 2021a,b,c] where hybrid planning – which we will explore later in the chapter – was used to model the problem. In that work, we demonstrated that planning could be very beneficial in reducing train delays on the nominal timetable. The approach was prior to the introduction of SPP and was focused mainly on modelling and the introduction of domain-dependant approaches to simplify the problem, while the job of finding of a plan was given to the state-of-the-art solver ENHSP.

In this section, we discuss the possible benefits of using the SPP approach in the INSTRADI application. The first reason of why the SPP approach can be very beneficial has already been demonstrated in the Chapters 2 through 5, where the solver PATTY, based on the SPP approach, was able to outperform all the state-of-the-art solver, including the ENHSP solver. This would greatly increase the solving capability, resulting in the possibility to be able to plan the movements of trains inside bigger stations with a larger number of trains.

The second reason instead lies in the use of patterns. As it can be seen by the description of the INSTRADI problem in the previous sections – and especially by Fig. 6.5 – inside a railway station, the sequence of the actions a single train has to make – e.g., for a transit train, entering the station, running through a route, stopping at a platform, departing from a platform and exiting the station – is fixed and can be known a priori. In fact, if we look at all these

possible actions for a train, we can easily see that there is already a partial order between all the actions concerning a single train. Let's consider, for simplicity, a transit train like the one depicted in Figure 6.5, which needs to move from the entry point 01 to the exit point 04. We can infer some simple rules on the order of the actions.

1. The entrance of the train from the entry-point 01 must happen before any other action.

2. The movements from the entry-point 01 to any of the platforms I, II, III, IV, are all mutually exclusive with each other, since the train cannot move on two routes at the same time and can only happen after the train has entered the station.

3. Stopping at any platform can happen only after the movement from the entry-point to the signal of the station.

4. Departing from any platform can happen only after the train has stopped at that platform.

5. Moving from any platform to the exit point 04 can only happen after the train has departed the station and all the actions that perform this movement are in mutex with each other.

6. Exiting the station from 04 is the last action performed by the train.

Looking at this example, we can thus see, that, for a single train, the order between the actions can already be known a priori and the purpose of the planner would be restricted to choose only a correct path for the train.

When dealing with multiple trains instead, we need to consider the temporal aspect. Given two trains which need to move inside the railway network, like the ones in Fig. 6.2, both trains might need to move on the same track circuit, route, or platform. Thus, we need to consider the interleaving of the two actions that make the two trains use the same resource, and put one before the other. If the two trains are scheduled in the station – by the forecast – at times distant between each other, the tie can be easily broken by putting first the action of the train that arrives first in the station. In the case where the trains are scheduled very close to each other, instead, we can break the tie according to some heuristic, that would tell us which train should pass

first[7] or we could employ a non-simple pattern which contains a repetition of the same action. To illustrate the last case, consider two actions a and b. The pattern $\prec = a; b; a$ covers both plans $\pi_1 = a; b$ and $\pi_2 = b; a$ where the two actions appear in either order.

Following the considerations made up until now, we can see that, the order of the actions can be already defined a priori, lightening the burden of the solver, which now only needs to find the correct path for each train and the correct temporal interleave of the actions to guarantee the safety of the movements. This is why we believe the SPP approach would be very beneficial in solving the INSTRADI problem.

6.2 Hybrid Planning

The nature of real-world applications usually requires the ability to reason in terms of continuous changes of numeric variables, and where the environment in which the agents act can respond to the agents' actions. In automated planning, this necessity led to the design of hybrid planning, expressible with PDDL+ [Fox and Long, 2006], that introduces the notions of processes and events to represent continuous change on numeric variables and effects which are not under the control of the planner. Hybrid planning has already proved very effective in solving complex real-world problems such as Traffic Control [Vallati et al., 2016], Train Dispatching [Cardellini et al., 2021c], Unmanned Aerial Vehicle Control Kiam et al. [2020] and Pharmacokinetic Optimization [Alaboud and Coles, 2019]. Hybrid planning tasks are notoriously difficult to cope with and a well-established approach to reason upon hybrid planning problems is through discretisation [Penna et al., 2012; Percassi et al., 2023a; Cardellini et al., 2024c], which allows breaking down complexity by assuming the time is discrete, and so are the actual numeric changes. This enables the reuse of well-established general search techniques based on forward state-based exploration to address hybrid planning problems.

[7]Note that in this scenarios – called *conflicts* in the railway jargon – this tie breaking approach is what human train dispatchers currently perform manually, choosing between the two trains based on the knowledge of the network, the traffic, and the delays of the two trains.

6.2.1 Formalism

Let $\delta \in \mathbb{Q}_{\geq 0}$ be a *discretisation step*. A *Discrete Hybrid (DH) planning task*, expressible with PDDL+, is a tuple

$$\Pi_\delta = \langle V_b, V_n, I, G, A, E, P_\delta \rangle.$$

The sets V_b and V_n are sets of propositional and numeric variables with domains in $\{\top, \bot\}$ and $\mathbb{Q}$, respectively. A *state* s is a total assignment to the variables in V_b and V_n to their respective domains. The *initial condition* I is a state. A *propositional condition* is an expression of the form $v = \top$ or $v = \bot$ with $v \in V_b$. A *numeric expression* φ is a formula over the variables in V_n and coefficients in $\mathbb{Q}$, in which variables can appear summed, subtracted, multiplied and divided between each other. A *numeric condition* is an expression of the form $\varphi \trianglerighteq 0$, with φ being a numeric expression and $\trianglerighteq \in \{=, >, \geq\}$. The *goal* G is a set of propositional and numeric conditions. The sets of *actions* A, *events* E and *processes* P_δ are sets of happenings. A *happening* h is a tuple of the form $\langle \mathrm{pre}(h), \mathrm{eff}(h) \rangle$ where the precondition $\mathrm{pre}(h)$ is a set of propositional and numeric conditions and the effect $\mathrm{eff}(h)$ is a set of propositional and numeric assignments. A *propositional assignment* is an expression of either $v := \top$ or $v := \bot$ with $v \in V_b$. A *numeric assignment* is an expression of the form $x := \varphi$ with $x \in V_n$ and φ a numeric expression. Even if, syntactically, the actions A, the events E and the processes P_δ are all sets of happenings, semantically they are quite different. Actions prescribe *may transitions*, meaning that, even if they are applicable, the planner could decide not to apply them. Events and processes instead prescribe *must transitions*, meaning that, if their preconditions are satisfied, they must be immediately applied by the planner. Events model *one-time* change in the propositional and numeric variables, while instead processes model a flow of change in the numeric variables. For this reason, we will assume in the following that processes have, in their effects, only numeric effects in the form $x := x + \varphi_1 \cdot \delta$, with φ_1 a numeric expression denoting the discrete change of x.

Let s be a state, v be a variable in $V_b \cup V_n$ and φ a numeric expression, we denote with $s(v)$ the value assumed by the variable v in the state s, and with

$s(\varphi)$ the value obtained by substituting in φ all the variables $x \in V_n$ with $s(x)$. A set of propositional or numerical conditions Ψ is satisfied in a state s, written as $s \models \Psi$, if for each $v = \top$ or $w = \bot$ in Ψ, we have $s(v) = \top$ and $s(w) = \bot$ and for each $\varphi \trianglerighteq 0$ in Ψ, we have $s(\varphi) \trianglerighteq 0$. A happening h is applicable in a state s if $s \models \mathrm{pre}(h)$. Applying a happening h to the state s results in the state $s' = \mathrm{res}(h, s)$ in which, (i) if $v := \top$ or $w := \bot$ is in $\mathrm{eff}(h)$, then $s'(v) = \top$ and $s'(w) = \bot$, (ii) if $x := \varphi$ is in $\mathrm{eff}(h)$, then $s'(x) = s(\varphi)$, and (iii) $s'(v) = s(v)$ otherwise. Applying a sequence of happenings $\langle h_1, \ldots, h_k \rangle$ to a state s results in the state $\mathrm{res}(\langle h_1, \ldots, h_k \rangle, s) = \mathrm{res}(h_k, \mathrm{res}(h_{k-1}, \mathrm{res}(\ldots, \mathrm{res}(h_1, s))))$.

We indicate with $E(s) \subseteq E$ the set of events which are triggered at a state s, i.e., $E(s) = \{e \in E \mid s \models \mathrm{pre}(e)\}$. In this chapter, we follow the semantic of events specified in Fox et al. [2005] for guaranteeing determinism of events. In particular, we impose that (i) each pair of events in $E(s)$ are not in mutex with each other (i.e., applying them in any order doesn't change the outcome), and (ii) that after applying all the events in $E(s)$ it is no longer possible to apply any event in $E(s)$ again, thus avoiding infinite repetition of events. We denote with $\mathrm{res}(E(s), s)$ the result of applying all the events in $E(s)$ in an arbitrary sequence.

Similarly to $E(s)$, we define $P_\delta(s)$ as the set of processes which are applicable in the state s. For this reason, given a state s, with applicable processes $P_\delta(s)$ we can compute the state s' resulting from the application of all the applicable processes as $s' = \mathrm{res}(P_\delta(s), s)$ such that $s'(v) = s(v)$ for each $v \in V_b$ (since propositional variables can't continuously change) and

$$s'(x) = s(x) + \sum_{\varphi \in \Phi_\delta(x,s)} s(\varphi) \quad \text{for each } x \in V_n,$$

where $\Phi_\delta(x, s) = \{\varphi_1 \cdot \delta \mid x := x + \varphi_1 \cdot \delta \in \mathrm{eff}(p), p \in P_\delta(s)\}$ is the set of discrete changes to x applicable from s.

DH Plan Validity. Let δ be a discretisation step and Π_δ a DH planning task. A *DH plan* π of Π_δ is a sequence of length n of timestamped actions where $\mathrm{action}_\pi(k) \in A$ is the k-th action of the plan which is applied at the timestamp $\mathrm{time}_\pi(k) \in \{i \cdot \delta \mid i \in \mathbb{N}\}$, with $k \in \{0, \ldots, n-1\}$. The plan is coupled with a value $M_\pi \in \mathbb{Q}_{\geq 0}$ which represents the *make span* of the plan,

i.e., when the last action or event triggers or process terminates, in general $M_\pi \geq \text{time}_\pi(n-1)$. Since the plan only contains actions, to validate it, we need to emulate the behaviour of events and processes to understand how the state of variables changed over time. For this reason, we project the plan into a discrete *history* in which time is discretized according to δ. A history $\mathcal{H} = \langle \mathcal{S}_1, \dots \mathcal{S}_m \rangle$ is an ordered list of *situations*. A situation $\mathcal{S}_i$ is a tuple $\langle t_i, c_i, s_i, a_i \rangle$ where $t_i \in \mathbb{Q}_{\geq 0}$ is the time of $\mathcal{S}_i$, $c_i \in \mathbb{N}$ is a counter, keeping track of the number of actions projected until $\mathcal{S}_i$, s_i is the state of the situation $\mathcal{S}_i$, and $a_i \in A \cup \{\epsilon\}$ is the action which is applied at $\mathcal{S}_i$, with ϵ be a no-op action with empty preconditions and effects.

Let Π_δ be a DH planning task, I the initial condition of Π_δ, $\delta \in \mathbb{Q}_{\geq 0}$ a discretisation step and π a discrete plan for Π_δ with make span M_π. We say that a history $\mathcal{H}^\pi_\delta = \langle \mathcal{S}_1, \dots, \mathcal{S}_m \rangle$, with $m \geq n$, is a discrete projection of π with discretisation step δ if

$A0$ $\mathcal{S}_1 = \langle t_1, c_1, s_1, a_1 \rangle$ with $t_1 = 0$, $c_1 = 0$ and $s_1 = I$,

and for each two contiguous situations $\mathcal{S}_i = \langle t_i, c_i, s_i, a_i \rangle$ and $\mathcal{S}_{i+1} = \langle t_{i+1}, c_{i+1}, s_{i+1}, a_{i+1} \rangle$ with $i \in \{1, \dots, m-1\}$

$A1$ if $E(s_i) \neq \emptyset$ then $a_i = \epsilon$, $s_{i+1} = \text{res}(E(s_i), s_i)$, $c_{i+1} = c_i$ and $t_{i+1} = t_i$,

$A2$ if $E(s_i) = \emptyset$, and $\text{time}_\pi(c_i) = t_i$ then $a_i = \text{action}_\pi(c_i)$, $s_{i+1} = \text{res}(a_i, s_i)$, $t_{i+1} = t_i$ and $c_{i+1} = c_i + 1$,

$A3$ if $E(s_i) = \emptyset$, $\text{time}_\pi(c_i) \neq t_i$ and $t_i < M_\pi$ then $a_i = \epsilon$, $s_{i+1} = \text{res}(P_\delta(s_i), s_i)$, $t_{i+1} = t_i + \delta$ and $c_{i+1} = c_i$.

We build the projection in a forward fashion. Rule $A0$ describes the initial situation at time zero, in which no action is yet projected, and the state is equal to the initial condition. After the initialisation, one of three things can happen, either events trigger, an action is applied, or processes are applied, bringing the time forward. Rule $A1$ states that if events can be triggered, they *must* be triggered, keeping the time still. If applying the events causes other events to become applicable, $A1$ is applied at all the following situations until the set is empty. Rule $A2$ states that if no event is applicable and the next action in the plan (based on the counter c_i) happens at the current time t_i, we apply it. If after applying the action, other events become applicable, rule $A1$ keeps expanding until no event can trigger. Finally, when all events have

triggered and all the actions at that time (if any) have been applied, we can apply the processes with rule $A3$, moving the time forward of one step δ.

Let δ be a discretisation step, π be a discrete plan of a DH planning task Π_δ, and let $\mathcal{H}_\delta^\pi = \langle \mathcal{S}_1, \ldots, \mathcal{S}_m \rangle$ be a discrete projection of π with discretisation step δ. We say that π is valid w.r.t. the discretisation step δ iff

- for each $\mathcal{S}_i = \langle t_i, c_i, s_i, a_i \rangle$ with $i \in \{1, \ldots, m-1\}$ we have $s_i \models \mathrm{pre}(a_i)$, and
- in $\mathcal{S}_m = \langle t_m, c_m, s_m, a_m \rangle$ we have $s_m \models G$.

6.2.2 SPP in Hybrid Planning

Hybrid Planning represents the most challenging flavour of planning that we covered in this thesis. In fact, there are mainly two difficulties of hybrid planning:

1. Respecting the *must semantic* [Cashmore et al., 2016]: i.e., if an event or process its applicable, their effects *must* be applied, which is the opposite of the *may semantic*, where if an action is applicable, its effects *may* be applied, depending on whether the actions is applied or not.
2. The discrete effects of action and events, and, most importantly, the continuous effects of processes can be defined through non-linear functions, which make difficult to compute the value resulting in the application of the effect, to compute heuristics or to determine the next state.

As a result of these complicances, there exists only few planners able to deal with hybrid planning problems in their full specification, i.e., ENHSP [Scala et al., 2016c], UPMURPHI [Penna et al., 2012], DINO Piotrowski et al. [2016], dREACH [Bryce et al., 2015], SMTPLAN+ Cashmore et al. [2016]. Out of these solvers, only dREACH and SMTPLAN+ are PAS based and only SMTPLAN+ can deal fully with the must semantic and the non-linear effects, while dREACH considers only hybrid planning tasks without events.

Fig. 6.6 shows a graphical representation of how the SMTPLAN+ planner solves a hybrid planning problem using a PAS approach. Having fixed a bound n on the number of steps, each gay box represents a time point in the plan,

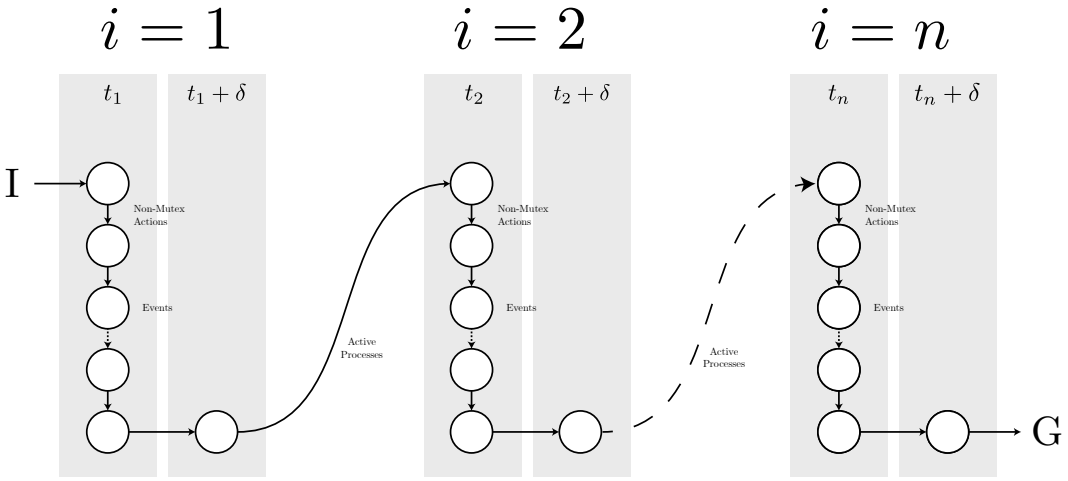

Fig. 6.6 Figure taken from Cashmore et al. [2016] showing how the SMTPLAN+ solver encodes a hybrid planning task into a PAS encoding.

with $t_i < t_i + \delta \leq t_{i+1}$ for $i \in [1, n)$. For each step i, there are two time points t_i and $t_i + \delta$. Every circle is a state. The states in t_i are all the intermediate states between application of the actions and events which are applied in t_i. In $t_i + \delta$ are reflected all the effects of the process active in t_i from the last state after all actions and events are applied in t_i. As for the standard PAS approach [Rintanen et al., 2006], for each step i only non-mutex actions are allowed to be applied at the same step, thus representing the first transition between the states at the time t_i, then, all the possible events which are triggered by the application of the actions are applied. Following the semantics, the same event cannot be triggered more than one time at the same time, thus, after at most $|E|$ different states, the resulting state is reflected in $t_i + \delta$ and we can proceed to apply all possible processes and their (possible) non-linear effects, reaching the state in t_{i+1}. By checking if the first state at time t_1 coincided with the initial condition I and the last state at time $t_n + \delta$ satisfies the state G, we can search for a plan with bound n, increasing n upon unsatisfiability.

Just by looking at Fig. 6.6, it can be noted how the SPP approach can be greatly beneficial also in the case of Hybrid Planning. In fact, the arrows that connect I to G provide a total order between actions, events, and processes. For this reason, a pattern for a hybrid planning problem, is a finite sequence of elements in $A \cup E \cup P_\delta$. Moreover, by the semantic, we know that at each time point t_i (i) events can happen at most one time, and (ii) all the applied events should not be in mutex, since their triggering order should not matter.

For this reason, in the pattern we can interleave each pair of actions a_i, a_{i+1} in A with the events in $|E|$, in any order, which could be triggered by the application of a_i, imposing some constraints to avoid mutex events being applied at the same time. As standard for the SPP approach, using the pattern can greatly reduce the bound n required to find a solution. We plan to explore how this approach performs compared to SMTPLAN+ in future work.

Chapter 7

Conclusions, Future Work and Algorethic

In this thesis, we have seen how the SPP approach can be very beneficial in many flavours of planning. We started analysing Classical and Numerical Planning in Chapter 2, then considered the case with Conditional Effects in Chapter 3 and Temporal Planning in Chapter 4. Then, focusing on the Numeric Planning flavour, we discussed in Chapter 5 how a symbolic-based search strategy can improve the performances of the SPP approach, by changing the pattern at some intermediate points in the search of a plan. Finally, in Chapter 6, we discussed how the SPP approach can be beneficial also on application domains, like the In-Station Train Dispatching Problem and on the higher flavour of planning, i.e., Hybrid Planning.

7.1 Future Work

We now recap each chapter's conclusion, and discuss on what could be the next steps, together with the ideas that the candidate hopes to be able to tackle in the upcoming years after his PhD.

1. In the conclusions of Chapter 2, we discussed how, in the modern editions of the International Planning Competition and in the recent literature, much focus has been given to trying to solve planning prob-

lems through *lifted* planners. The use of a lifted representation, instead of a grounded one, allows for planners to tackle planning problems with thousands of variables and actions in a more compact way, thus accelerating the planning procedure. Instrumenting this thesis solver's PATTY as a lifted planner would be very beneficial in solving all those problems, very popular in the realm of transportation and logistics, where there is the need to plan the movements of thousands of agents and objects.

2. In Chapter 3, we saw how computing the transitive closure for each action, compactly representing all the states reachable by multiple applications of the same action, can be very beneficial in solving planning problems with CEs. We saw in the experimental analysis that computing the transitive closure becomes expensive when the number of variables involved increase. In the conclusions of that chapter, we discussed how, in the past three decades, several approaches have been presented in the literature to reduce the cost of computation of the transitive closure, either by parallelizing the computation or by subdividing the problem into smaller sub-problems. Analysing these approaches and comparing them, together with analysing how the special case of classical planning with CEs could be exploited in the computation of the transitive closure, could be very beneficial and allow solving even more problems.

3. In Chapter 4, we discussed how the SPP approach can be beneficial in the case of Numeric Temporal Planning. We remember that in Temporal Planning, an action's preconditions can be specified to hold either at the start, at the end or throughout the action, while effects could be applied only at the start or at the end. A rather new formalism of Temporal Planning has lately emerged from the literature, called *Temporal Planning with Intermediate Conditions and Effects* (ICEs) [Valentini et al., 2020]. In this formalism, conditions and effects are indeed intermediate, meaning that conditions can be configured to hold at some specific interval throughout the action and effects can be scheduled to be applied at some specific time after or before the action has started or ended. This approach has attracted a lot of attention lately due to its capacity to better model real-world domain applications. In the case of Temporal

Planning with SPP, we saw that a pattern is a sequence of snap actions, either starting or ending a durative action. In the case of Temporal Planning with ICEs, the pattern could become a finite sequence of *events*, which would signal either the start/end of an action, the start/end of the interval in which a condition must hold, or the application of an effect. Looking at how the SPP approach would be beneficial could help better solve real-world scenarios.

4. In Chapter 5 we saw how, in the SPP approach, the use of the same static pattern, computed from the initial condition, could be detrimental in some domains where the order between actions is not fixed and could change at some point during the plan. Thus, to improve the SPP approach, we presented a technique to recompute the pattern during the search phase, by searching for some intermediate states in which at least one of the subgoals was satisfied, and computing the pattern again from that intermediate state. In that approach, it was paramount to iteratively find intermediate states which always satisfied an additional subgoal more than the previous intermediate state. The approach thus greatly depends on the subgoals of the planning task, and how states that satisfy them are positioned in the state-space. Moreover, the approach could be nullified by the presence of a single (sub-)goal. Subgoaling has been extensively studied in the literature, also for numeric planning (see, e.g., [Scala et al., 2016a, 2020]), and the discussed limitations are well-known in the literature. For this reason, another alternative to subgoaling was introduced. *Landmarks* [Helmert and Domshlak, 2009], roughly speaking, landmarks are facts – i.e., assignments to variables – that must be true at some point in every valid solution plan. Unfortunately, computing landmarks is a PSPACE-complete task on its own. Even if intractable, many planners employ relaxed computations of landmarks to compute heuristics [Helmert and Domshlak, 2009; Kuroiwa et al., 2022] and to improve the search of a plan. In our approach, using landmarks could amount allow finding intermediate states earlier and recompute the pattern from those intermediate states, even in the presence of a single subgoal.

5. Chapter 6 contains a discussion on how to improve, through the use of SPP, application domains like the In-Station Train Dispatching Problem and the Hybrid Planning flavour. These are only discussions, but the candidate is currently working on consolidating the claims made in the chapter.

In this thesis, we hope to have given the idea that the SPP approach can be a new, interesting and powerful technique to solve planning problems for real-world applications.

7.2 Algorethic of Planning

Motives. Since I[1] believe that the SPP approach presented in this thesis could be beneficial in several real-world applications, here I want to give a small contribution to the public discussion that, in the past years, has emerged regarding the ethics of Artificial Intelligence (AI). This contribution does not pretend to be a new contribution to the ethic's literature, which I leave to ethicists and philosophers to do, but rather wants to be a space in which I reflect on potential harm in the use of planning – which has been improved by the SPP approach – in real-world applications that are nowadays touching, directly or indirectly, the every-day life of every human being. In this section, I will collect the thoughts that other people, more expert and deep than myself, have produced in the last years. The motivation for this section lies in my belief that, even as scientists – even when working in theoretical domains thus not actually implementing our ideas in applications – we have a responsibility in at least get informed on the problems and risks that our technologies pose and thus trying to guide our work to be used only in the applications that we deem correct and secure.

> "Scientists and those working in the digital world should continue to promote such research, engaging in a noble competition to combat the wrongful use of newly available technology. I therefore appeal to computer engineers to feel personally responsible for

[1]The thoughts and considerations presented in this section are to be considered the work of the doctoral candidate and no one else.

> building the future. It is their task to undertake, with our support, an ethical development of algorithms – an *algorethic* –, and in this way, to help create a new ethics for our time" (Francis [2019])

The term *algorethic* [Benanti, 2018, 2023] is meant to identify a new ethics discipline concerning the ethical development of algorithms of AI, and is opposed to *algocracy*, i.e., the idea that algorithms must rule free.

Definition of AI. During my PhD – which started in November 2021 and ended in November 2024 – AI saw a deep surge in attention from the public, and in particular from the media. This surge was originated by the launch, in November 2022 of the software called *ChatGPT*, the first chatbot based on new AI technologies called Generative Adversarial Networks (GAN) [Goodfellow et al., 2014] and Large Language Models (LLM) [Vaswani et al., 2017], which were actually published in the literature several years before. This new software shook the world and all media outlets started covering the topic, sometimes with very exaggerated tones, contributing to grow even more what it is now called the AI *spring* [Manyika and Bughin, 2019]. This excitement led to new products based on LLMs being produced almost daily and the public being exposed constantly to stories and examples of these products. This exposure has produced the effect that, nowadays, the term AI has become mostly a synonym for LLMs. However, as researchers, especially in the planning community, we know that AI is instead much more broad. If we read the definition of AI in Article 3 of the AI *act* of the European Union, i.e., the law which regulates the use of AI in the European Union, which should enter into force in February 2025, we read that the definition of AI is

> "a machine-based system that is designed to operate with varying levels of autonomy and that may exhibit adaptiveness after deployment, and that, for explicit or implicit objectives, infers, from the input it receives, how to generate outputs such as predictions, content, recommendations, or decisions that can influence physical or virtual environments." (Council of European Union [2024a])

and in Recital 12 we read

> "A key characteristic of AI systems is their capability to infer. [...] The techniques that enable inference while building an AI system include machine learning approaches that learn from data how to achieve certain objectives, and logic- and knowledge-based approaches that infer from encoded knowledge or symbolic representation of the task to be solved." (Council of European Union [2024b])

This definition and the recital denotes actually a much broader categorization of AI based on the type of inference (also known as *reasoning*): either the inference (i) is *inductive*, i.e., machine learning approaches, like LLMs, that learn from data, or (ii) *deductive*, i.e., logic- and knowledge-based approaches. The topic of this thesis, planning, falls indeed in the deductive reasoning categorization.

Two kinds of AI to emulate the brain. In his seminal book, *Thinking, fast and slow* [Kahneman, 2011], Kahneman describes how the human brain's mode of thinking can be categorized into two systems: System 1 and System 2. System 1 is fast, automatic, and intuitive, handling routine tasks and reactions with little effort. It's quick but prone to errors and biases, relying on instincts and mental shortcuts. In contrast, System 2 is slow, deliberate, and logical, requiring focused attention and analytical thinking. This system steps in for complex or unfamiliar problems, offering more accurate, rational decision-making but at a mental cost. While System 1 often dominates, System 2 helps correct errors, balancing fast intuition with careful thought. It can be noted how, in our definition of AI, System 1 can be mapped into inductive reasoning – i.e., machine learning – while System 2, can be mapped into deductive reasoning – i.e, logic- and knowledge-based approaches, like planning. The inductive and deductive approaches, emulating System 1 and System 2, have always been developed in parallel, with few attempts to join the two approaches together, and presently, there is not a singular system able to emulate together System 1 and System 2.

The System 2, or deductive reasoning, actually was the first branch of AI that sprung interest and awe outside the community [Buchanan, 2005] with the development of knowledge-based *expert systems* in the 1960s and early

1970s, and more famously in 1997, where the expert system of IBM, Deep Blue, was able to win in a game of chess against the master Garry Kasparov. In 2010, came the advent of social media and search engines, together with an increase in the computing power of machines. The inductive reasoning approaches, like machine learning, deep learning, GANs and LLMs– which were at first limited by the amount of data and computing power required to *train* them – thus started to become feasible approaches, and rapidly surged the interest of researchers and the public. As for every innovation, the surge of work and interest in a new technology, creates the illusion that that particular technology can solve any problem that we present to it, only to fail in their endeavour. This happened to expert systems in the late 80s and 90s and is happening now with the plethora of systems based on LLMs and GANs. Thus, when systems do not live up to the enthusiasm, solutions start to become old and disappointment certainly follows. When this happened to expert systems in the 80s, it was called the AI *winter*, which led John Haugeland to coin the expression *Good Old-Fashioned Artificial Intelligence* (GOFAI), to indicate the deductive reasoning approaches, stating the idea that these approaches were destined to disappear, since they are unable to bring useful solution to the market [Haugeland, 1989].

LLMs cannot do everything. The idea that a new technology is the ultimate answer for every problem is nowadays gaining even more attraction for the case of LLMs, where the public, the media, some AI companies and, unfortunately, some AI researchers, are screaming how *Artificial General Intelligence* (AGI) – i.e., one artificial intelligence system which can perform every possible task provided, thus combining both System 1 and System 2 – can be achieved through the use of LLMs and its realisation is near or will be here in n weeks or months.

Fortunately, this wave of claims of the closeness to build an AGI is contrasted by few researchers which actually try to dismantle these claims with science. The work of Rao Kambhampati's Team [Kambhampati, 2024; Kambhampati et al., 2024; Valmeekam et al., 2024] is showing that the argument of this thesis, planning, is actually one of the biggest examples of why LLMs are not at all near to be considered AGIs. In their work, Kambhampati et al. empirically show how state-of-the-art LLMs-based chatbots, like ChatGPT, do

not manifest real planning abilities, but mostly use their great capacity to retrieve information learned from its training data to provide plans only for those planning tasks which have already been solved and which solution was inside the training data. In fact, they made an experiment using the most popular classical planning problem, *Blocksworld*. In Blocksworld, a set of blocks are placed either on a table or on top of other blocks and the goal is to arrange some of these blocks in a stack in a particular order either by stacking them on top of each other, or unstacking them. On this domain, the latest version of ChatGPT was able to solve 34.6% of the planning tasks provided. When the name of the actions were changed to words with different meanings – e.g, *stack* becomes *attack* and *unstack* becomes feast – the solved planning problems by ChatGPT dropped to 0.2%. Since a planner's ability to find a plan does not depend on the name of the action, it is clear that the abilities of ChatGPT on planning was only due to the presence, in the training set, of the solutions of the planning task.

Thus, planning is currently one of the champions of the deductive branch of AI, showing that LLMs are actually very great at emulating System 1, but not so much when it comes to System 2. As stated by Kahneman, System 2 helps correct errors, balancing fast intuition with careful thought. With their inability to emulate System 2, thus LLM systems like ChatGPT do not have the correct tool to autoregulate, being critic of their output and thus produce solutions which are transparent, inclusive, responsible, impartial, trustable, and secure[2].

There are thus three possibilities regarding these limitations of LLMs, either

1. System 2 emulation, and thus planning, is indeed not an interesting problem for the public, as claimed by Haugeland, and thus LLMs will continue to be the most prominent paradigm for AI,

2. if it is indeed an interesting problem, then either LLMs will fail to overcome their limitation, hitting a wall at these kinds of problems, leading again to a AI winter, or

[2]These are the principles discussed in the AI Rome Call for Ethics, which we will discuss further ahead

3. a significant number of resources will be put into creating new technologies – beyond LLMs– which could emulate System 2 and thus planning.

I argue that we are heading in the direction pointed by the 3rd item, and thus is of paramount importance to pose some ethical questions also to System 2 technologies, like planning, before the hype on these technologies causes – as already happened for LLMs– a gold rush which gives small attention to ethical considerations.

Planning in a belligerant world. In 1968, the catholic Pope Paul VI established January 1st as the "World Day of Peace". In the midst of the Cold War, the Pope wanted to dedicate a day to reflect and pray for peace, each year characterized by a message by the pontiff, reflecting on a particular theme. For the 58th World Day of Peace, held on the 1st of January 2024, Pope Francis dedicated the day to the theme "Artificial Intelligence and Peace" [Francis, 2024b]. The catholic Pope – which in the collective imaginary is at the antipodes of the topics of technology and engineering – felt the urgency to send a message to the civil society, the international community and researchers to warn against the risks that AI pose to the world's peace. This is indeed remarkable and worthy of note, and should be taken seriously by researchers. Currently, in 2024, there are 42 countries in a state of war [World Population Review, 2024]. While the media is stressing the conflict in Ukraine and in the Holy Land, which had the most estimated casualties in 2024 – respectively 49,881 and 22,386 – there are other countries which get less coverage, currently facing a Civil War – like Myanmar, Sudan, Ethiopia, and Syria – or victim of terrorism – like Nigeria, Burkina Faso, Mali and the D.R. of Congo – or facing a drug war – like Mexico.

AI systems are already being used in warfare in various conflicts, and the military sector is where AI is becoming more advanced, thanks to high funding. In this domain, planning can be used to select the best strategies for invasions, which targets should be eliminated first, thus seeking the best plan that minimizes losses on one side and maximizes them on the other. Using AI systems for warfare, completely removes the human conscience from the decision. By abstracting human-life into variables, probabilities and metrics,

we create a barrier between us and the suffering of human beings, and thus the loss of human-life becomes just a number. In the message mentioned above, Pope Francis wrote:

> "In these days, as we look at the world around us, there can be no escaping serious ethical questions related to the armaments sector. The ability to conduct military operations through remote control systems has led to a lessened perception of the devastation caused by those weapon systems and the burden of responsibility for their use, resulting in an even more cold and detached approach to the immense tragedy of war. Research on emerging technologies in the area of so-called Lethal Autonomous Weapon Systems, including the weaponization of artificial intelligence, is a cause for grave ethical concern. Autonomous weapon systems can never be morally responsible subjects. The unique human capacity for moral judgment and ethical decision-making is more than a complex collection of algorithms, and that capacity cannot be reduced to programming a machine, which, as "intelligent" as it may be, remains a machine. For this reason, it is imperative to ensure adequate, meaningful and consistent human oversight of weapon systems." (Francis [2024b])

Planning in a lonely world. In 2020, the entire world experienced a common fragility. The Covid-19 pandemic united us in the idea that we are all equally fragile, regardless of our social status, education and economic resources. The pandemic made us face our limits. We learned to think more about other people, to take care of the people we loved, who we could no longer see. Unfortunately, after the pandemic, we quickly forgot the lessons we had learned, and went in an entirely different direction. The abrupt return to everyday life made us quickly forget the lessons we have learnt. The decline, that we faced even before the pandemic, quickly reprised, even more steeply, bringing us to an even more individualistic world. The competition between people, the structure of the cities, of the work environment, of buildings, fashion and lifestyle, the use of social networks are rapidly feeding the increase in the city of lonely people, as if loneliness was a direct cause of the

modern world. This has increased the discrepancy with the peripheries, not only geographically, in the sense of suburbs, but also the "existential peripheries", which are present even in the city centre, in the houses of lonely old people, in nursing homes, in the migrant families where parents have to work multiple jobs, in jails. Loneliness is a risk factor for mortality [Holt-Lunstad et al., 2015]. The 21st Surgeon General of the United States, Vivek Murthy, in a report on "Healing Effects of Social Connection and Community" wrote in 2023

> "Such a world, where we recognize that relationships are just as essential to our well-being as the air we breathe and the food we eat, is a world where everyone is healthier, physically and mentally. It is a world where we respect and value one another, where we look out for one another, and where we create opportunities to uplift one another." (Office of the Surgeon General [2023])

For "existential peripheries", AI techniques must consider social relationships as a positive factor and not something to minimize. The tendency nowadays is to build solutions that make our life faster, easier and where everything can be delivered without leaving our homes. Planning technologies, must not always consider the shorter plan, which could also minimise interactions, as the only valid solution, but must consider the social aspect of the plan, strengthening the possibility of encounters and relationships between actors. The definition of planning, as agents which act in the environment, which we used also in this thesis, should not make the planning task designers forget that agents are still humans, and thus interactions between the human agents must be encouraged and not diminished.

Concerning "geographical peripheries", AI systems used, e.g., in urban planning and in the management of traffic [Vallati et al., 2016; Cardellini et al., 2024a] must consider how the urban network could present skewed topologies, which could amount in more traffic being routed in the outskirts of the cities, with an increase in pollution and health issues in the population.

> "The use of autonomous and fixed metrics to distribute traffic can lead to certain urban areas receiving prolonged heavy traf-

> fic, with a significant detrimental impact on quality of life, property valuation, and increased health risks due to pollution and noise. Addressing this issue requires ensuring that the benefits and burdens of vehicles are distributed equitably and do not impact communities disproportionately." (Guo et al. [2024])

Planning in a burning world. Climate change is one of the most pressing issues facing our world today. Over the last 50 years, global temperatures have risen significantly, with the Earth's average surface temperature increasing by about 0.9 °C to 1.2 °C (1.6 °F to 2.2 °F) since the late 19th century. These significance shifts in global temperatures are primarily driven by human activities, such as burning fossil fuels, deforestation, and industrial processes.

AI is unfortunately contributing to this phenomenon. Training LLMs, like ChatGPT, on huge amounts of data, requires a lot of power and water to cool down the machines. A recent survey by the Washington Post showed that to train GPT-3, Microsoft employed 700,000 litres of water, while Meta used 22 million litres while training the open-source LLaMa 3. A simple query run by ChatGPT consumes "0.14 kilowatt-hours (kWh) of electricity, equal to powering 14 LED light bulbs for 1 hour". A simple query run by ChatGPT "once weekly for a year by 1 out of 10 working Americans requires 121,517 megawatt-hours (MWh), equal to the electricity consumed by all D.C. households for 20 days". In July, Google released its most recent environmental report, showing its carbon emission footprint rose by 48 percent, largely due to AI and data centres. It also replenished only 18 percent of the water it consumed [Verma and Tan, 2024].

As explained in previous paragraphs, despite being empirically proven, the fact that "LLMs cannot plan" [Valmeekam et al., 2024] is an idea that most researchers and, all the AI companies, do not share. Since their invention, LLMs have manifested what are called *emergent abilities*, meaning abilities for which LLMs are not trained, but that are learnt from the data. Another word used in this context is *zero-shot*, referring to a model's ability to perform a task or make predictions on data it hasn't seen before—without any task-specific training. Suppose a LLM model trained on all the translations from the Bible, the most translated document in the world, on some particular task. As a

byproduct, even if not trained specifically for that, the model would also be able to translate other phrases, not present in the Bible, between two of the languages in which the Bible is translated, even if nobody on earth knows how to translate directly between these two languages. This byproduct is called zero-shot learning. The mainstream idea is that, given massive data to train on, the LLMs systems will, at some point, manifest on its own the ability to plan, and they will be able to perform planning as a zero-shot task. In trying to achieve these abilities, AI companies will increase their energy- and water-consumption even more, thus contributing even more to climate change. Before employing this humongous amount of energies, we firstly have to ask ourselves, as researchers, if these abilities can actually emerge.

It is not all bad. In the last three paragraphs I have presented the risks that AI, and especially planning, can have in a belligerent, lonely and burning world. While these problems concern me, AI has proven in the last years to be an incredible technology which could be very helpful in solving many problems that the world is facing. In the application of planning proposed in this thesis, the In-Station Train Dispatching Problem, the use of AI could greatly benefit the life of passengers, where, in a world of 8 billion people, railway traffic is becoming more and more congested, and delays are nowadays very common. In Medicine, thanks to AI, new, break-through discoveries can be made in the cure for cancer or other illnesses. AI can help alleviate the burden of very fatiguing jobs, and make work even more productive. In agriculture and farming, the use of AI could reduce waste and help in producing even more food with less water and energy consumption.

> "After all, we cannot doubt that the advent of artificial intelligence represents a true cognitive-industrial revolution, which will contribute to the creation of a new social system characterised by complex epochal transformations. For example, artificial intelligence could enable a democratization of access to knowledge, the exponential advancement of scientific research and the possibility of giving demanding and arduous work to machines. Yet at the same time, it could bring with it a greater injustice between advanced and developing nations or between dominant

> and oppressed social classes, raising the dangerous possibility that a "throwaway culture" be preferred to a "culture of encounter". (Francis [2024a])

What has been done? In recent years, several policies and initiatives have been put in place to try to regulate AI so that its great power can actually benefit, rather than harm, society. I want to highlight two: the *EU* AI *act* and the *Rome Call for* AI *Ethics*.

The AI Act, which we briefly covered in previous paragraphs, is a European Union Law which has been presented in February 2024 and will be put into force in 2025. Its main contribution is the categorization of AI systems into four levels of risk: (i) *Unacceptable Risk*: AI that poses serious threats to safety or fundamental rights and are thus banned, (ii) *High-Risk*: Includes applications in critical areas like healthcare, law enforcement, and employment, (iii) *Limited Risk*: AI systems that require transparency measures, such as chatbots, where users must be informed they are interacting with, and (iv) *Minimal Risk*, such as video games or spam filters, which have few or no additional regulatory requirements. Unacceptable Risk includes AI for social scoring, certain types of surveillance, and subliminal manipulation. High-risk AI systems must undergo rigorous assessments before deployment. Developers must document the AI 's design, training data, and testing results. AI systems must also provide explanations of their decisions, ensuring that they are transparent and accountable. The Act mandates human oversight, especially in high-stakes decisions (e.g., healthcare diagnostics, judicial decisions). Human operators must have the ability to intervene or override AI decisions in certain contexts. In my opinion, the great force of this act stands in the mandatory steps that will be required before deploying AI technologies in High-Risk areas, forcing researchers and stakeholders to stop and think about the consequences of their work. Moreover, the act will produce benchmarks and test suites to evaluate the standards of the products, giving to consumer certifications of the goodness of the software they are using.

The Rome Call for AI Ethics is an initiative launched by the Vatican in 2020 to promote ethical standards in artificial intelligence. Spearheaded by the Pontifical Academy for Life, leaded by Mons. Vincenzo Paglia, it brings

together religious leaders, tech companies, and governments to advocate for a human-centered approach to AI development. The idea for this Rome Call was born from a meeting between Microsoft President Brad Smith and Mons. Paglia, proposed by the former, to discuss the rapid growth that AI was experiencing and the desperate need for guidance on how to design it ethically [Paglia, 2024]. This meeting spurred an interest in other companies, like IBM. As stated earlier, the fact that big companies decided to turn to a religious institution, like the Vatican, to guide them towards an ethical way for AI, must be taken seriously, and unfortunately means that no other actor in the world, including universities and research institutions, is making enough efforts in considering the ethical implication of AI and is so much concerned with the impacts on human life.

The key principles of the Rome call are

1. Transparency: AI systems should be designed in a way that makes them understandable and transparent to the people who use them. This includes making AI decisions explainable to ensure public trust.

2. Inclusion: The initiative advocates for ensuring that AI technology is accessible and beneficial to all, regardless of socioeconomic status, geographic location, or background.

3. Accountability: All stakeholders, including governments, companies, and developers, are called to work collaboratively in the ethical governance of AI. This includes ensuring accountability for the consequences of AI deployment.

4. Responsibility: Organizations and developers are encouraged to take responsibility for their AI systems, especially in high-stakes areas like healthcare, justice, and labor. They should strive to prevent harm and protect vulnerable populations.

5. Impartiality: AI should be developed and used without bias, fostering fairness and avoiding discrimination. This principle is especially relevant to ensure that AI does not reinforce or exacerbate existing social inequalities.

6. Security and Privacy: The Rome Call stresses the importance of securing AI systems to protect personal data and privacy, which is seen as essential to respect individual rights and maintain public trust.

What can we do as researchers? As stated in the motives, I believe that researchers, even if theoretical, have the responsibilities to feel these problems burning. As engineers and scientists, we are used to feeling the urgency to do things, and, unfortunately, to rapidly dismiss problems when it appears that nothing can be done, or the problem seems too huge to handle. For building an algorethic, an ethic for algorithms, we must start not by *doing* but by *caring*. Caring about the problems, as the three I have mentioned before, is what starts the internal movements that make us ask questions and search for answers. To help improve the world with our technologies, we need to actually live in the world, with its complexity and its difficulties. We must therefore always start thinking about the consequences of our work, siding with the marginalized – the poor, the elderly, the sick, and children – and making them central to our technologies, rather than treating them as special cases.

> "In the quest for normative models that can provide ethical guidance to developers of digital technologies, it is indispensable to identify the human values that should undergird the efforts of societies to formulate, adopt and enforce much-needed regulatory frameworks. The work of drafting ethical guidelines for producing forms of artificial intelligence can hardly prescind from the consideration of deeper issues regarding the meaning of human existence, the protection of fundamental human rights and the pursuit of justice and peace. This process of ethical and juridical discernment can prove a precious opportunity for shared reflection on the role that technology should play in our individual and communal lives, and how its use can contribute to the creation of a more equitable and humane world. For this reason, in debates about the regulation of artificial intelligence, the voices of all stakeholders should be taken into account, including the

poor, the powerless and others who often go unheard in global decision-making processes." (Francis [2024b])

References

Alaboud, F. K. and Coles, A. (2019). Personalized Medication and Activity Planning in PDDL+. In Benton, J., Lipovetzky, N., Onaindia, E., Smith, D. E., and Srivastava, S., editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, pages 492–500. AAAI Press.

Bacchus, F. (2001). The AIPS '00 Planning Competition. *AI Mag.*, 22(3):47–56.

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*, pages 865–871. IEEE Computer Society.

Barrett, C., Fontaine, P., and Tinelli, C. (2016). The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org. Accessed: 2024-01-06.

Benanti, P. (2018). *Oracoli: tra algoretica e algocrazia*. Collassi. Luca Sossella editore, Rome.

Benanti, P. (2023). The urgency of an algorethics. *Discov. Artif. Intell.*, 3(1).

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.

Bercher, P., Haslum, P., and Muise, C. (2024). 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*, pages 7941–7950. ijcai.org.

Bofill, M., Espasa, J., and Villaret, M. (2016). The RANTANPLAN planner: system description. *Knowl. Eng. Rev.*, 31(5):452–464.

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

Boleto, G., Oneto, L., Cardellini, M., Maratea, M., Vallati, M., Canepa, R., and Anguita, D. (2021). In-Station Train Movements Prediction: from Shallow to Deep Multi Scale Models. In *29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2021, Online event (Bruges, Belgium), October 6-8, 2021*.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33.

Brand, S., Bäck, T., and Laarman, A. (2023). A Decision Diagram Operation for Reachability. In Chechik, M., Katoen, J., and Leucker, M., editors, *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, volume 14000 of *Lecture Notes in Computer Science*, pages 514–532. Springer.

Bryant, R. E. (1985). Symbolic manipulation of Boolean functions using a graphical representation. In Ofek, H. and O'Neill, L. A., editors, *Proceedings of the 22nd ACM/IEEE conference on Design automation, DAC 1985, Las Vegas, Nevada, USA, 1985*, pages 688–694. ACM.

Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691.

Bryant, R. E. (1992). Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318.

Bryce, D., Gao, S., Musliner, D. J., and Goldman, R. P. (2015). Smt-based nonlinear PDDL+ planning. In Bonet, B. and Koenig, S., editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 3247–3253. AAAI Press.

Buchanan, B. G. (2005). A (very) brief history of artificial intelligence. *AI Mag.*, 26(4):53–60.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. (1992). Symbolic Model Checking: 10ˆ20 States and Beyond. *Inf. Comput.*, 98(2):142–170.

Bylander, T. (1994). The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.*, 69(1-2):165–204.

Cabodi, G., Camurati, P., Lavagno, L., and Quer, S. (1997). Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits. In Yoffa, E. J., Micheli, G. D., and Rabaey, J. M., editors, *Proceedings of the 34st Conference on Design Automation, Anaheim, California, USA, Anaheim Convention Center, June 9-13, 1997*, pages 728–733. ACM Press.

Cardellini, M., Dodaro, C., Maratea, M., and Vallati, M. (2024a). Optimising dynamic traffic distribution for urban networks with answer set programming. *Theory and Practice of Logic Programming*, page 1–19.

Cardellini, M. and Giunchiglia, E. (2025). Temporal Numeric Planning With Patterns. In *Thirty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2025, February 25 - March 4, 2025, Philadelphia, Pennsylvania, USA*. AAAI Press.

Cardellini, M., Giunchiglia, E., and Maratea, M. (2024b). Symbolic Numeric Planning with Patterns. In Wooldridge, M. J., Dy, J. G., and Natarajan, S., editors, *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*, pages 20070–20077. AAAI Press.

Cardellini, M., Maratea, M., Percassi, F., Scala, E., and Vallati, M. (2024c). Taming discretised PDDL+ through multiple discretisations. In Bernardini, S. and Muise, C., editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, pages 59–67. AAAI Press.

Cardellini, M., Maratea, M., Vallati, M., Boleto, G., and Oneto, L. (2021a). A Planning-based Approach for In-Station Train Dispatching. In Ma, H. and Serina, I., editors, *Proceedings of the Fourteenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021*, pages 156–158. AAAI Press.

Cardellini, M., Maratea, M., Vallati, M., Boleto, G., and Oneto, L. (2021b). An Efficient Hybrid Planning Framework for In-Station Train Dispatching. In Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M. A., editors, *Computational Science - ICCS 2021 - 21st International Conference, Krakow, Poland, June 16-18, 2021, Proceedings, Part I*, volume 12742 of *Lecture Notes in Computer Science*, pages 168–182. Springer.

Cardellini, M., Maratea, M., Vallati, M., Boleto, G., and Oneto, L. (2021c). In-Station Train Dispatching: A PDDL+ Planning Approach. In Biundo, S., Do, M., Goldman, R., Katz, M., Yang, Q., and Zhuo, H. H., editors, *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, pages 450–458. 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., editors, *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, pages 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.

Ciardo, G., Lüttgen, G., and Siminiceanu, R. (2001). Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation. In Margaria, T. and Yi, W., editors, *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2031 of *Lecture Notes in Computer Science*, pages 328–342. Springer.

Cimatti, A., Griggio, A., Schaafsma, B. J., and Sebastiani, R. (2013). The mathsat5 SMT solver. In Piterman, N. and Smolka, S. A., editors, *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*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer.

Clarke, E. M., McMillan, K. L., Campos, S. V. A., and Hartonas-Garmhausen, V. (1996). Symbolic Model Checking. In Alur, R. and Henzinger, T. A., editors, *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, volume 1102 of *Lecture Notes in Computer Science*, pages 419–427. Springer.

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.

Corrêa, A. B., Frances, G., Hecher, M., Longo, D. M., and Seipp, J. (2023). Levitron: Combining Ground and Lifted Planning. *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Corrêa, A. B., Francès, G., Hecher, M., Longo, D. M., and Seipp, J. (2023). Scorpion Maidu: Width Search in the Scorpion Planning System. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Council of European Union (2024a). Article 3: Definitions | EU Artificial Intelligence Act. https://artificialintelligenceact.eu/article/3/. [Accessed 01-11-2024].

Council of European Union (2024b). Recital 12 | EU Artificial Intelligence Act. https://artificialintelligenceact.eu/recital/12/. [Accessed 01-11-2024].

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*, pages 1852–1859.

Daniele, M., Traverso, P., and Vardi, M. Y. (1999). Strong Cyclic Planning Revisited. In Biundo, S. and Fox, M., editors, *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10,*

1999, Proceedings, volume 1809 of *Lecture Notes in Computer Science*, pages 35–48. Springer.

de Moura, L. M. and Bjørner, N. S. (2008). Z3: An Efficient SMT Solver. In Ramakrishnan, C. R. and Rehof, J., editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer.

Drexler, D. (2023). Vanir: Learning and Executing Width-based Hierarchical Policies. *Tenth International Planning Competition (IPC-10) Learning Track: Planner Abstracts*.

Edelkamp, S. and Reffel, F. (1998). Obdds in heuristic search. In Herzog, O. and Günter, A., editors, *KI-98: Advances in Artificial Intelligence, 22nd Annual German Conference on Artificial Intelligence, Bremen, Germany, September 15-17, 1998, Proceedings*, volume 1504 of *Lecture Notes in Computer Science*, pages 81–92. 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*, pages 49–64. Springer.

Fikes, R. and Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4):189–208.

Fox, M., Howey, R., and Long, D. (2005). Validating Plans in the Context of Processes and Exogenous Events. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 1151–1156. AAAI Press / The MIT Press.

Fox, M. and Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20:61–124.

Fox, M. and Long, D. (2006). Modelling Mixed Discrete-Continuous Domains for Planning. *J. Artif. Intell. Res.*, 27:235–297.

Francis (2019). Address to Participants in the Congress on "Child Dignity in the Digital World". https://www.vatican.va/content/francesco/en/speeches/2019/november/documents/papa-francesco_20191114_convegno-child%20dignity.html. Accessed: 2024-11-01.

Francis (2024a). Address Of Pope Francis at the G7 Session on Artificial Intelligence. https://www.vatican.va/content/francesco/en/speeches/2024/june/documents/20240614-g7-intelligenza-artificiale.html. Accessed: 2024-11-01.

Francis (2024b). Artificial Intelligence and Peace. https://www.vatican.va/content/francesco/en/messages/peace/documents/20231208-messaggio-57giornatamondiale-pace2024.html. Accessed: 2024-11-01.

Gazen, B. C. and Knoblock, C. A. (1997). Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In Steel, S. and Alami, R., editors, *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, volume 1348 of *Lecture Notes in Computer Science*, pages 221–233. Springer.

Gebser, M., Kaminski, R., König, A., and Schaub, T. (2011). Advances in *gringo* Series 3. In Delgrande, J. P. and Faber, W., editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer.

Geldenhuys, J. and Valmari, A. (2001). Techniques for Smaller Intermediary BDDs. In Larsen, K. G. and Nielsen, M., editors, *CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings*, volume 2154 of *Lecture Notes in Computer Science*, pages 233–247. Springer.

Gelfond, M. and Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Gener. Comput.*, 9(3/4):365–386.

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*, pages 226–229. AAAI.

Gerevini, A. E., Percassi, F., and Scala, E. (2024). An Effective Polynomial Technique for Compiling Conditional Effects Away. In Wooldridge, M. J., Dy, J. G., and Natarajan, S., editors, *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*, pages 20104–20112. AAAI Press.

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.

Gigante, N. and Scala, E. (2023). On the Compilability of Bounded Numeric Planning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 5341–5349. ijcai.org.

Giunchiglia, E. and Maratea, M. (2007). Planning as satisfiability with preferences. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 987–992. AAAI Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial networks. *CoRR*, abs/1406.2661.

Guo, R., Vallati, M., Wang, Y., Zhang, H., Chen, Y., and Wang, F. (2024). Sustainability opportunities and ethical challenges of ai-enabled connected autonomous vehicles routing in urban areas. *IEEE Trans. Intell. Veh.*, 9(1):55–58.

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.

Haugeland, J. (1989). *Artificial intelligence - the very idea*. MIT Press.

Helmert, M. (2002). Decidability and Undecidability Results for Planning with Numerical State Variables. In Ghallab, M., Hertzberg, J., and Traverso, P., editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 44–53. AAAI.

Helmert, M. and Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A., Howe, A. E., Cesta, A., and Refanidis, I., editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

Hoffmann, J. (2003). The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Artif. Intell. Res.*, 20:291–341.

Höller, D. and Behnke, G. (2022). Encoding Lifted Classical Planning in Propositional Logic. In Kumar, A., Thiébaux, S., Varakantham, P., and Yeoh, W., editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pages 134–144. AAAI Press.

Holt-Lunstad, J., Smith, T. B., Baker, M., Harris, T., and Stephenson, D. (2015). Loneliness and social isolation as risk factors for mortality: a meta-analytic review. *Perspectives on psychological science*, 10(2):227–237.

Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York.

Kambhampati, S. (2024). Can large language models reason and plan? *CoRR*, abs/2403.04121.

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., and Murthy, A. (2024). Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Katz, M. (2019). Red-Black Heuristics for Planning Tasks with Conditional Effects. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 7619–7626. AAAI Press.

Kautz, H. A., McAllester, D. A., and Selman, B. (1996). Encoding Plans in Propositional Logic. In Aiello, L. C., Doyle, J., and Shapiro, S. C., editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 374–384. Morgan Kaufmann.

Kautz, H. A. and Selman, B. (1992). Planning as Satisfiability. In Neumann, B., editor, *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, pages 359–363. John Wiley and Sons.

Kautz, H. A. and Selman, B. (1996). Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In Clancey, W. J. and Weld, D. S., editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, pages 1194–1201. AAAI Press / The MIT Press.

Kiam, J. J., Scala, E., Jávega, M. R., and Schulte, A. (2020). An AI-Based Planning Framework for HAPS in a Time-Varying Environment. In Beck, J. C., Buffet, O., Hoffmann, J., Karpas, E., and Sohrabi, S., editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 412–420. AAAI Press.

Kissmann, P. and Hoffmann, J. (2013). What's in It for My BDD? On Causal Graphs and Variable Orders in Planning. In Borrajo, D., Kambhampati, S., Oddi, A., and Fratini, S., editors, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI.

Kissmann, P. and Hoffmann, J. (2014). BDD Ordering Heuristics for Classical Planning. *J. Artif. Intell. Res.*, 51:779–804.

Kuroiwa, R., Shleyfman, A., and Beck, J. C. (2022). LM-Cut Heuristics for Optimal Linear Numeric Planning. In Kumar, A., Thiébaux, S., Varakantham, P., and Yeoh, W., editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pages 203–212. AAAI Press.

Lauer, P., Torralba, Á., Fiser, D., Höller, D., Wichlacz, J., and Hoffmann, J. (2021). Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In Zhou, Z., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4119–4126. ijcai.org.

Leofante, F., Giunchiglia, E., Ábrahám, E., and Tacchella, A. (2020). Optimal Planning Modulo Theories. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4128–4134. ijcai.org.

Lin, B., Touati, H. J., and Newton, A. R. (1990). Don't Care Minimization of Multi-Level Sequential Logic Networks. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1990, Santa Clara, CA, USA, November 11-15, 1990. Digest of Technical Papers*, pages 414–417. IEEE Computer Society.

Manyika, J. and Bughin, J. (2019). The Coming AI Spring. https://www.project-syndicate.org/commentary/artificial-intelligence-spring-is-coming-by-james-manyika-and-jacques-bughin-2019-10. [Accessed 01-11-2024].

Matsunaga, Y., McGeer, P. C., and Brayton, R. K. (1993). On Computing the Transitive Closure of a State Transition Relation. In Dunlop, A. E., editor, *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pages 260–265. ACM Press.

McCarthy, J. and Hayes, P. (1969). Some Philosophical Problems From the Standpoint of Artificial Intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.

McDermott, D. V. (2000). The 1998 AI Planning Systems Competition. *AI Mag.*, 21(2):35–55.

McGeer, P. C. (1989). *On the interaction of functional and timing behaviour of combinational logic circuits*. University of California, Berkeley.

Muise, C. J., McIlraith, S. A., and Beck, J. C. (2012). Improved Non-Deterministic Planning by Exploiting State Relevance. In McCluskey, L.,

Williams, B. C., Silva, J. R., and Bonet, B., editors, *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Nebel, B. (2000). On the Compilability and Expressive Power of Propositional Planning Formalisms. *J. Artif. Intell. Res.*, 12:271–315.

Office of the Surgeon General (2023). Our epidemic of loneliness and isolation: The us surgeon general's advisory on the healing effects of social connection and community [internet].

Paglia, V. (2024). *L'algoritmo della vita: etica e intelligenza artificiale*. Piemme, Milano, i edizione edition.

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*, pages 12095–12102. AAAI Press.

Panjkovic, S. and Micheli, A. (2024). Abstract Action Scheduling for Optimal Temporal Planning via OMT. In Wooldridge, M. J., Dy, J. G., and Natarajan, S., editors, *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*, pages 20222–20229. AAAI Press.

Penna, G. D., Magazzeni, D., and Mercorio, F. (2012). A universal planning system for hybrid domains. *Appl. Intell.*, 36(4):932–959.

Percassi, F., Scala, E., and Vallati, M. (2023a). A Practical Approach to Discretised PDDL+ Problems by Translation to Numeric Planning. *J. Artif. Intell. Res.*, 76:115–162.

Percassi, F., Scala, E., and Vallati, M. (2023b). A Practical Approach to Discretised PDDL+ Problems by Translation to Numeric Planning. *J. Artif. Intell. Res.*, 76:115–162.

Piotrowski, W. M., Fox, M., Long, D., Magazzeni, D., and Mercorio, F. (2016). Heuristic planning for PDDL+ domains. In Kambhampati, S., editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3213–3219. IJCAI/AAAI Press.

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

Rete Ferroviaria Italiana (RFI) (2023). La Rete Oggi. Accessed: March 18, 2024.

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*, pages 280–287. AAAI.

Rintanen, J. (2011). Heuristics for Planning with SAT and Expressive Action Definitions. In Bacchus, F., Domshlak, C., Edelkamp, S., and Helmert, M., editors, *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*. 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*, pages 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*, pages 743–749. ijcai.org.

Rintanen, J., Heljanko, K., and Niemelä, I. (2006). Planning as satisfiability: parallel plans and algorithms for plan search. *Artif. Intell.*, 170(12-13):1031–1080.

Röger, G., Pommerening, F., and Helmert, M. (2014). Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting. In Schaub, T., Friedrich, G., and O'Sullivan, B., editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 765–770. IOS Press.

Russell, S. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

Scala, E. and Gerevini, A. E. (2020). An Introduction to Numeric Planning - Representation and Search Algorithms. In *ICAPS 2020 - Summer School*.

Scala, E., Haslum, P., and Thiébaux, S. (2016a). Heuristics for Numeric Planning via Subgoaling. In Kambhampati, S., editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3228–3234. IJCAI/AAAI Press.

Scala, E., Haslum, P., Thiébaux, S., and Ramírez, M. (2016b). Interval-Based Relaxation for General Numeric Planning. In Kaminka, G. A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F., and van Harmelen, F., editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 655–663. IOS Press.

Scala, E., Haslum, P., Thiébaux, S., and Ramírez, M. (2016c). Interval-Based Relaxation for General Numeric Planning. In Kaminka, G. A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F., and van Harmelen, F., editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 655–663. IOS Press.

Scala, E., Ramírez, M., Haslum, P., and Thiébaux, S. (2016d). Numeric Planning with Disjunctive Global Constraints via SMT. In Coles, A. J., Coles, A., Edelkamp, S., Magazzeni, D., and Sanner, S., editors, *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, pages 276–284. AAAI Press.

Scala, E., Saetti, A., Serina, I., and Gerevini, A. E. (2020). Search-Guidance Mechanisms for Numeric Planning Through Subgoaling Relaxation. In Beck, J. C., Buffet, O., Hoffmann, J., Karpas, E., and Sohrabi, S., editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 226–234. AAAI Press.

Scala, E. and Vallati, M. (2021). Effective grounding for hybrid planning problems represented in PDDL+. *Knowl. Eng. Rev.*, 36:e9.

Shin, J. and Davis, E. (2004). Continuous Time in a SAT-Based Planner. In McGuinness, D. L. and Ferguson, G., editors, *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*, pages 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.

Sipser, M. (1997). *Introduction to the theory of computation*. PWS Publishing Company.

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.

Taitler, A., Alford, R., Espasa, J., Behnke, G., Fiser, D., Gimelfarb, M., Pommerening, F., Sanner, S., Scala, E., Schreiber, D., Segovia-Aguas, J., and Seipp, J. (2024). The 2023 International Planning Competition. *AI Mag.*, 45(2):280–296.

Valentini, A., Micheli, A., and Cimatti, A. (2020). Temporal planning with intermediate conditions and effects. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9975–9982. AAAI Press.

Vallati, M., Magazzeni, D., Schutter, B. D., Chrpa, L., and McCluskey, T. L. (2016). Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. In Schuurmans, D. and Wellman, M. P., editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3188–3194. AAAI Press.

Valmeekam, K., Stechly, K., and Kambhampati, S. (2024). Llms still can't plan; can lrms? A preliminary evaluation of openai's o1 on planbench. *CoRR*, abs/2409.13373.

van Dijk, T., Meijer, J., and van de Pol, J. (2019). Multi-core on-the-fly saturation. In Vojnar, T. and Zhang, L., editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 58–75. Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Verma, P. and Tan, S. (2024). A bottle of water per email: the hidden environmental costs of using AI chatbots. https://www.washingtonpost.com/technology/2024/09/18/energy-ai-use-electricity-water-data-centers/. [Accessed 03-11-2024].

Wehrle, M. and Rintanen, J. (2007). Planning as Satisfiability with Relaxed ∃-Step Plans. In Orgun, M. A. and Thornton, J., editors, *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, volume 4830 of *Lecture Notes in Computer Science*, pages 244–253. Springer.

World Population Review (2024). Countries Currently at War. https://worldpopulationreview.com/country-rankings/countries-currently-at-war. [Accessed 02-11-2024].

Yoon, S. W., Fern, A., and Givan, R. (2007). FF-Replan: A Baseline for Probabilistic Planning. In Boddy, M. S., Fox, M., and Thiébaux, S., editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, page 352. AAAI.

Acknowledgements

Ringrazio,

- il Prof. Enrico Giunchiglia, per il suo impegno a tramandarmi la passione per la ricerca e per la bellezza della matematica, per i suoi aforismi e per guidarmi ogni giorno a essere un ricercatore migliore;
- il Prof. Marco Maratea, per i primi anni del mio percorso accademico e del mio dottorato, per la sua guida incessante e il pensare sempre a ciò che è migliore per me, per avermi fatto appassionare al mondo della ricerca;
- il Prof. Mauro Vallati, per la sua grande conoscenza e capacità tecnica negli ambiti applicativi, per la sua ospitalità ad Huddersfield, per la sua generosa compagnia alle conferenze;
- i revisori Dott. Andrea Micheli e Dott. Nicola Gigante, per i loro utili commenti, l'apprezzamento del mio lavoro e per la loro ricerca, che ha motivato molte sezioni di questa tesi;
- i miei genitori, per avermi dato fiducia, per aver creduto in me, per avermi cresciuto in una casa felice e per avermi donato la curiosità;
- il mio amore Emanuela, per il suo grande amore nei miei confronti e per la altrettanto grande pazienza nel sopportare le mie distrazioni, per la tranquillità e la serenità che ha portato nella mia vita e nel mio futuro;
- la Comunità di Sant'Egidio tutta, per avermi dato le parole che, in piccolissima parte, ho cercato di trasmettere in Sezione 7.2 e per avermi dato un orientamento nella vita;

- i ragazzi della Scuola della Pace: Besi, Hongbo, Albi, Gabri, Leon, Vilson, Michi ed Entoni, per avermi insegnato cosa vuol dire mettersi dalla parte dei piccoli;
- i nostri amici del Giro: Lancinè, Aldo, Abdel, Mousta e Doina, per il vostro grande affetto e per la speranza che trasmettete;
- i miei amici di sempre: Davide, Elena e Picci, per la strada che abbiamo fatto insieme e che continueremo a fare.