Journal of Artificial Intelligence Research 76 (2023) 115-162 Submitted 05/2022; published 01/2023

A Practical Approach to Discretised PDDL+ Problems by Translation to Numeric Planning

Francesco Percassi F.PERCASSI@HUD.AC.UK
School of Computing and Engineering
University of Huddersfield, UK

Enrico Scala ENRICO.SCALA@UNIBS.IT
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Brescia, Italy

Mauro Vallati M.VALLATI@HUD.AC.UK
School of Computing and Engineering
University of Huddersfield, UK

Abstract

PDDL+ models are advanced models of hybrid systems and the resulting problems are notoriously difficult for planning engines to cope with. An additional limiting factor for the exploitation of PDDL+ approaches in real-world applications is the restricted number of domain-independent planning engines that can reason upon those models.

With the aim of deepening the understanding of PDDL+ models, in this work, we study a novel mapping between a time discretisation of PDDL+ and numeric planning as for PDDL2.1 (level 2). The proposed mapping not only clarifies the relationship between these two formalisms but also enables the use of a wider pool of engines, thus fostering the use of hybrid planning in real-world applications. Our experimental analysis shows the usefulness of the proposed translation and demonstrates the potential of the approach for improving the solvability of complex PDDL+ instances.

1. Introduction

The availability of domain-independent planning engines is fostering the use of automated planning in a wide range of applications. This is despite the complexity issues inherent in plan generation, which are exacerbated by the separation of planning logic from domain knowledge (McCluskey & Porteous, 1997). A major advantage of the separation of planning logic from domain knowledge lies in the fact that, given a standard language to be used for input and output, the two components can be interchanged in a modular way, without affecting the other component, and with no negative repercussions on the overall application framework where the planning system is usually embedded (McCluskey, Vaquero, & Vallati, 2017; Vallati, Chrpa, McCluskey, & Hutter, 2021).

This modular approach promotes the use of reformulation techniques, which can automatically re-formulate or re-represent the domain knowledge and/or the problem instance in order to increase the effectiveness of the planning logic component and increase the scope of problems solved. The general idea is to develop reformulation techniques that are agnostic about the domain knowledge and the planning logic and use them to form a wrapper around the planning engine, improving its performance for the domain in which it is applied. The transformation is then reversed after a solution has been found, such that the solution is rephrased in the original formulation language. A well-known class of reformulation approaches aims at translating a model from the original input language to a different one. The idea is usually to remove the use of some poorly supported features of the language (Helmert, 2009; Percassi & Gerevini, 2019; Bonassi, Gerevini, Percassi, & Scala, 2021) or to re-represent the problem in a less expressive language. The latter strategy has the advantage of increasing the number of planning engines that are able to reason upon the planning problem, and leverage existing robust technologies devised for solving more restricted cases. Well-known examples of this approach include the translation of conformant planning problems into classical problems (Taig & Brafman, 2013; Grastien & Scala, 2020), the re-representation of uncertainty in conformant planning problems (Palacios & Geffner, 2009), and the translation of complex temporal aspects in PDDL2.1 (Cooper, Maris, & Régnier, 2010).

In this paper, we introduce a reformulation approach for translating discretised PDDL+ planning problem instances into PDDL2.1 instances. PDDL+ models are advanced models of hybrid systems and the resulting problems are notoriously difficult to cope with. Further, a limited number of planning engines are able to parse PDDL+ models, therefore translating into PDDL2.1 can significantly extend the pool of planning engines that can be used to solve a given instance.

More precisely we study two translations. The first translation leads to a numeric planning problem that is exponentially larger than the input PDDL+ but preserves the number of discrete transitions. The second one keeps the resulting formulation polynomial but requires also a polynomial number of additional transitions to generate a solution. We start off by revisiting the formalisation provided by Shin and Davis (2005), which lets us formalise the problem without the need to depend on the hybrid automaton interpretation proposed by Fox and Long (2003), and also crisply formalises the connection between the continuous-time representation, and its discretisation. We formally present the two translations and show how these can be extended to encode a cascade of events emulated by actions. Further, we introduce two approaches for optimising the operationality of the translated models; this makes the resulting PDDL2.1 models more effective. Indeed, our optimisations let us reduce the number of necessary checks along with a plan whilst preserving soundness and completeness with a pre-processing phase that looks at the structure of our problem instance and applies the optimisations only when it is possible. We validate the resulting formulations against a set of challenging benchmark domains, including real-world applications, and well-known planning engines, and we assess the impact of the introduced optimisations. Our results indicate that the proposed translations can unlock the use of PDDL2.1 planning engines for tackling hybrid PDDL+ problems, with the clear advantage of expanding the number of approaches that can be used to solve a problem.

This paper significantly extends our conference paper on the topic (Percassi, Scala, & Vallati, 2021). This extended version advances our previous work along several dimensions. Firstly, we revise the formalisation of PDDL+ and its discretisation that are not fully spelt out in the conference paper. We enrich our discussion with examples and more formal definitions. This reformulation does not change the meaning of our previous formalisation but makes things more precise and less ambiguous. Secondly, we provide full proofs for all the translations that we have presented and also enrich the discussion with a concrete full example that is an extension of the well-known car domain by Fox and Long (2006).

To these proofs, we also add new theorems on the size of our translated output. Thirdly, we formalise and study theoretically two optimisations that we apply to our translation schemata. Such optimisations are meant to reduce the length of the plans obtained on the translated output. Last but not least, we substantially extend our experimental analysis to evaluate our contribution, pros and cons, in greater detail.

This paper is organised as follows. Section 2 formalises the problem of PDDL+. The proposed translations are then presented in detail in Section 3. Section 4 focuses on different ways for optimising the models generated by the introduced translations. An extensive experimental analysis is provided in Section 5. Related works are discussed in Section 6. Finally, Section 7 frames the scope of the proposed approach, highlighting its advantages and discussing some limitations of the general discretisation-based approach, and conclusions are given in Section 8.

2. Problem Formalisation

In this section we formalise the problem of PDDL+ (Fox & Long, 2006) without durative actions, thus focusing on the core features of this formalism, and the problem of numeric planning as the one that can be specified in PDDL2.1, Level 2 (Fox & Long, 2003), hereinafter simply referred to as PDDL2.1. We first describe the syntax of our problems and then detail the semantics. Our discussion follows the formalisation and terminology provided by Shin and Davis (2005) in a way that is instrumental for our work.

Let F be a set of Boolean variables, and X be a set of numeric variables taking values from $\mathbb{R}_\bot = \mathbb{R} \cup \{\bot\}$; $\bot$ represents an undefined numeric value. A PDDL+ state s is a full assignment to all variables in F and in X. Given a state s and a variable $v \in F \cup X$, we make use of $s[v]$ to indicate the value assumed by v in s. This value can either be a numeric value or a Boolean value depending on whether v is in X or in F, respectively. A numeric expression is defined inductively as follows: $x \in X$, and $k \in \mathbb{Q}$ are numeric expressions; let ξ and ξ' be two numeric expressions, $\xi \; op \; \xi'$ with $op \in \{+, -, \times, /\}$ is a numeric expression. For instance, $x+5+y$ is a numeric expression while $x++5$ is not. Let ξ be a numeric expression, we denote as $s[\xi]$ the evaluation of ξ in s. A division between two numeric expressions ξ and ξ', i.e., ξ/ξ', is undefined in s if $s[\xi'] = 0$, i.e., division by zero is undefined. Moreover, let x be a variable involved in an expression ξ. If x is undefined in state s, then ξ is undefined too.

Boolean and numeric assignments both have the form $\langle f := b \rangle$. For Boolean assignments, f is a Boolean variable and $b \in \mathbb{B} = \{\top, \bot\}$. For numeric assignments, $\langle \{asgn, inc, dec\}, x, \xi \rangle$, where $\{asgn, inc, dec\}$ are the contractions of the keywords *assign*, *increase* and *decrease*, respectively, x is a numeric variable and ξ is a numeric expression, respectively. A Boolean condition has the form $\langle f = b \rangle$, where f is a Boolean variable and $b \in \mathbb{B}$. A numeric condition has the form $\langle \xi \bowtie 0 \rangle$ where ξ is a numeric expression and $\bowtie \in \{\leq, <, =, >, \geq\}$. We detail our problems using formulas expressed in Negation Normal Form (NNF). The terms of our formulas contain both propositional and numeric conditions.

Definition 1 (PDDL+ Problem). *A* PDDL+ planning problem Π *is a tuple* $\langle F, X, I, G, A, E, P \rangle$ *where:*

- *F and X are sets of Boolean and numeric variables, respectively;*

- *I is the description of the initial state expressed as a full assignment to all variables in X and F;*
- *G is the description of the goal expressed as a formula;*
- *A and E are the sets of actions and events, respectively. Actions and events are pairs $\langle p, e \rangle$ where p is a formula and e is a set of conditional effects of the form $c \triangleright e$ where (i) c is a formula and (ii) e is a set of Boolean or numeric assignments;*
- *P is a set of processes. A process is a pair $\langle p, e' \rangle$ where p is a formula and e' is a set of numeric continuous effects expressed as pairs $\langle x, \xi \rangle$ where $x \in X$ and ξ is a numeric expression.*[1]

Let $a = \langle p, e \rangle$ be an action or an event or a process, we use *pre*(a) to denote the precondition p of a, and *eff*(a) the effect e of a. In what follows we generally use the symbols a, ρ and ε for denoting an action, a process, and an event, respectively. Given a PDDL+ problem Π we denote the whole state space as $states(\Pi) = \{\mathbb{B}^{|F|} \times \mathbb{R}_{\perp}^{|X|}\} \cup \{undefined\}$, where *undefined* denotes a special state that represents the notion of inconsistency within our semantic (this aspect will be dealt with in detail later). Note that, although X takes values in $\mathbb{R}$, it is not possible to use real values in PDDL+ in the problem specification.

For ease of exposition and following PDDL common practice, in our examples and encodings we will, when convenient, represent the state and the effects using a set-theoretic representation. Boolean conditions and assignments having the extended form $\langle f = \top \rangle$ ($\langle f := \top \rangle$) and $\langle f = \bot \rangle$ ($\langle f := \bot \rangle$) are shortened, by abuse of notation, to f and $\neg f$, and a conditional effect where the left-hand side is always satisfied i.e., $\top \triangleright e$, is rewritten as e. For example, formula $\langle a = \top \rangle \wedge \langle b = \bot \rangle$ is contracted into $a \wedge \neg b$, and an assignment having the form $\{\langle a := \top \rangle, \langle b := \bot \rangle\}$ is contracted into $\{a, \neg b\}$. In the set-theoretic representation, a state is expressed as a pair encompassing a set of facts and a set of assignments to numeric variables: if a fact is not mentioned in the state, the Boolean variable associated with that fact is false (closed world assumption); similarly, if a numeric variable is not given any assignment in the state, such a variable is undefined. For example, a state $s = \{\langle a = \top \rangle, \langle b = \bot \rangle\}$, where a and b are Boolean variables, can be expressed more succinctly by $s = \{a\}$.

A plan for a PDDL+ problem is an ordered set of timed actions plus a time envelope, organised formally as the following.

Definition 2 (PDDL+ Plan)**.** *A PDDL+ plan π_t for a PDDL+ problem Π is a pair $\langle \pi, \langle t_s, t_e \rangle \rangle$ where $\pi = \langle \langle a_0, t_0 \rangle, \langle a_1, t_1 \rangle, ..., \langle a_{n-1}, t_{n-1} \rangle \rangle$ is a finite sequence of pairs such that for each $i \in [0..n-1]$*[2] *we have that $a_i \in A$ and $t_s \leq t_i \leq t_e$; $\langle t_s, t_e \rangle$, with $t_s, t_e \in \mathbb{Q}$ and $t_s \leq t_e$, is the envelope within which π is performed. π_t is a well-formed plan if $\forall\, i, j \in [0..n-1]$ and $i < j$, then $t_i \leq t_j$ holds.*

Hereinafter, we consider just well-formed plans. Let $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ be a PDDL+ plan and let $\langle a_i, t_i \rangle$ and $\langle a_j, t_j \rangle$ be two pairs appearing in π. We write $\langle a_i, t_i \rangle \prec \langle a_j, t_j \rangle$ ($a_i \prec a_j$)

1. We will see that under continuous time interpretation, ξ denotes the net derivative of x.
2. $[1..n]$ is the integer interval $\{1, ..., n\}$.

iff $\langle a_i, t_i \rangle$ (a_i) appears strictly before $\langle a_j, t_j \rangle$ (a_j) in π. Furthermore, given a sequence of objects A we denote its set representation as $set(A)$, e.g., $set(\pi)$ is the set of all time-stamped actions in π.

Having formally defined the PDDL+ problem, we now see how a PDDL2.1 one is defined.

Definition 3 (PDDL2.1 Problem). *A* PDDL2.1 *planning problem* Π *is a tuple* $\langle F, X, I, G, A, c \rangle$ *where all elements are as for* PDDL+*, yet there are neither processes nor events, and* c *is a function that associates to each action a rational non-negative cost.*

We clarify that our definition of PDDL2.1 slightly differs from that provided by Fox and Long (2003) in which action costs are not included.

For a PDDL2.1 problem Π, the state space is defined as $states(\Pi) = \{\mathbb{B}^{|F|} \times \mathbb{Q}_?^{|X|}\} \cup \{undefined\}$.

Definition 4 (PDDL2.1 Plan). *A* PDDL2.1 plan *for a* PDDL2.1 *problem* Π *is simply a finite sequence of actions from* A.

Again, let π be a PDDL2.1 plan and let a_i and a_j be two actions appearing in π. We write $a_i \prec a_j$ iff a_i appears strictly before a_j in π.

Before going into the formal details of the semantics of PDDL+ and PDDL2.1, let us briefly explain intuitively what these problems actually entail. Roughly speaking, a PDDL+ problem consists in finding a number of actions along a potentially infinite timeline, whilst conforming to a number of processes and events that may change the state of the world in a continuous or instantaneous manner as time goes by. A solution for a PDDL+ problem is such iff all actions are applicable when they are scheduled and the goal is met at time t_e. In technical terms, PDDL+ prescribes events and processes to be interpreted as *must* discrete and continuous transitions along a potentially infinite timeline, while actions are *may* transitions. A PDDL2.1 problem is a variant where there is no explicit management of time and exogenous changes, and we only seek a sequence of actions that starts from some initial state and yields a state satisfying the goal.

We start with the simpler semantics of PDDL2.1 and then delve into the details of PDDL+ by stressing, in particular, its temporal aspect.

The satisfiability of a formula follows the usual semantics of propositional logic extended with numeric conditions. In particular, let $\langle \xi \bowtie 0 \rangle$ be a numeric condition, $s \models \langle \xi \bowtie 0 \rangle$ iff $s[\xi] \bowtie 0$ evaluates to true. A formula containing a numeric expression that is undefined in the state s is treated as unsatisfiable in s by default reasoning. For instance, if we have $\alpha = \langle x \geq 0 \rangle \vee \langle x \leq 10 \rangle$ with x undefined in s, then α is not satified in s; similarly, let $\beta = \langle x \geq 0 \rangle \vee \langle y = 1 \rangle$ we have that $s \models \beta$ iff $s[y] = 1$.

The application of a in s yields the state $s' = \Gamma(s, a)$ in a way that, for each $v \in F \cup X$, the following holds:

$$\Gamma(s,a) = \begin{cases} s'[v] = b & \text{if } \exists\, \alpha \triangleright \{..., \langle v := b \rangle\} \in \mathit{eff}(a) \text{ and } s \models \alpha, \text{ where } b \in \mathbb{B}; \\ s'[v] = s[\xi] & \text{if } \exists\, \alpha \triangleright \{..., \langle asgn, v, \xi \rangle\} \in \mathit{eff}(a) \text{ and } s \models \alpha; \\ s'[v] = s[v] + s[\xi] & \text{if } \exists\, \alpha \triangleright \{..., \langle inc, v, \xi \rangle\} \in \mathit{eff}(a) \text{ and } s \models \alpha; \\ s'[v] = s[v] - s[\xi] & \text{if } \exists\, \alpha \triangleright \{..., \langle dec, v, \xi \rangle\} \in \mathit{eff}(a) \text{ and } s \models \alpha; \\ \mathit{undefined} & \text{if there exists at least a pair of conflicting assignments in } \mathit{eff}(a); \\ s'[v] = s[v] & \text{otherwise.} \end{cases}$$

Case (i) of $\Gamma(s,a)$ deals with the propositional assignment, (ii) the simple numeric assignment, (iii) and (iv) the additive numeric assignments. Case (v) handles the possibility that two contradicting effects exist, either Boolean or numeric. For a given state s and an action a, two Boolean effects $\alpha \triangleright \{\ldots, \langle f := \top\rangle\}, \beta \triangleright \{\ldots, \langle f := \bot\rangle\} \in \mathit{eff}(a)$, where α and β are two formulae and $f \in F$, are conflicting if $s \models \alpha \wedge \beta$, and two numeric effects $\alpha \triangleright \{\ldots, \langle op, x, \xi\rangle\}, \beta \triangleright \{\ldots, \langle op', x, \xi'\rangle\} \in \mathit{eff}(a)$ with $x \in X$ and $op, op' \in \{asgn, inc, dec\}$, are conflicting if $s \models \alpha \wedge \beta$; in other words, two numeric effects are conflicting if they contextually affect the same variable in different ways. Finally, case (vi) is the frame-axiom, so the value of v persists if no effect of a affects v.

Let $\langle a_0, \ldots, a_{n-1}\rangle$ be a sequence of actions, we say that it is applicable in s iff $s \models \mathit{pre}(a_0)$, $\Gamma(s, a_0) \models \mathit{pre}(a_1)$, ..., $\Gamma(\ldots\ \Gamma(s, a_0)), a_{n-2}) \models \mathit{pre}(a_{n-1})$. We use $\gamma(s, \cdot)$ for the state resulting by applying either an action, i.e., $\gamma(s, a)$, or a sequence of (applicable) actions, i.e., $\gamma(s, \langle a_0, \ldots, a_{n-1}\rangle)$, in state s. Since actions and events have the same structure, we use the same notation for denoting the state resulting from the application of an event, i.e., $\gamma(s, \varepsilon)$, or a sequence of (applicable) events, i.e., $\gamma(s, \langle \varepsilon_0, \ldots, \varepsilon_{n-1}\rangle)$.

Definition 5 (Valid and Optimal pddl2.1 Plan)**.** *Let* $\Pi = \langle F, X, I, G, A, c\rangle$ *be a* pddl2.1 *problem, a plan* $\pi = \langle a_0, \ldots, a_{n-1}\rangle$ *is said to be* valid plan for Π *if it is applicable and* $\gamma(I, \pi) \models G$*. The plan* π *is said to be* optimal *if among all valid plans for* Π*, it is the one that minimises the total cost,* $\pi = \underset{\pi' \text{ valid for } \Pi}{\operatorname{argmin}} \sum_{a \in set(\pi')} c(a)$.

The semantics of pddl+ can be specified through the notion of time points, intervals, and histories over intervals. We use these to characterise the projection of a plan and conclude with the formalisation of the validity of a pddl+ plan which we interpret both on a continuous and a discrete timeline. We start with the continuous version and then introduce its discretisation. Note that all our translations that we present from Section 3 are all assuming a discrete model.

Definition 6 (Time Point)**.** *A* time point T *is a pair* $\langle t, n\rangle$ *where* $t \in \mathbb{R}$ *and* $n \in \mathbb{N}$.

Given a time point $T = \langle t, n\rangle$, in the rest of the paper we will refer to t as the clock and n as the step of T.

Time points over $\mathbb{R} \times \mathbb{N}$ are ordered lexicographically, i.e., let $\langle t_1, n_1\rangle$ and $\langle t_2, n_2\rangle$ be two time points, $\langle t_1, n_1\rangle < \langle t_2, n_2\rangle$ iff either $t_1 < t_2$ or $t_1 = t_2$ and $n_1 < n_2$. Let T_1 and T_2 be two time points, a closed (open) time interval $I = [T_1, T_2]$ ((T_1, T_2)) is the non-empty set $I = \{T \mid T_1 \leq T \leq T_2\}$ ($\{T \mid T_1 < T < T_2\}$).

Intuitively, a time point allows us to order situations along time and impose an ordering, by using a natural number n, among two different situations whenever such situations share the same clock. This representation idealises the execution of actions and events to be truly instantaneous as long as an order is imposed among transitions sharing the same clock as done in the work of Shin and Davis (2005). Note that pddl+ (Fox & Long, 2006) supports such a temporal model for events only; mutexes actions instead are still required to be temporally separated by at least a small quantity of time ϵ (ϵ-separation requirement).[3]

3. For a deeper analysis on the temporal implications of ϵ vs non ϵ-separation requirement in the context of temporal pddl2.1, the interested reader can look at the paper by Gigante, Micheli, Montanari, and Scala (2022).

Definition 7 (History)**.** *A* history H for a PDDL+ problem Π over $I = [T_s, T_e]$ *maps each time point in I into a situation. A "situation at time point T" is the tuple $H(T) = \langle H_A(T), H_s(T) \rangle$, where $H_A(T)$ is a sequence of actions executed at time point T and $H_s(T)$ is a state, i.e., the assignment to all variables in X and F at time point T.*

An event ε is active (triggered) in a state s iff $s \models pre(\varepsilon)$. Given a history H over $I = [T_s, T_e]$ and a time point $T \in I$, multiple events can be triggered.

Hereinafter, given a set S, we denote with $seq(S)$ any possible sequencing of its elements.

Definition 8 (Triggered Events)**.** *Let H be a history for a* PDDL+ *problem* Π *over I and let $T \in I$. The* sequence of events triggered at time point T *is defined as* $E_{trigg}(T) = seq(\{\varepsilon \in E \mid H_s(T) \models pre(\varepsilon)\})$.

A time point is said to be significant if something meaningful happens, such as the execution of an action, the triggering of a sequence of events, or a change in the set of active processes.

Definition 9 (Significant Time Point)**.** $T = \langle t, n \rangle$ *is a* significant time point *(hereinafter STP) of a history H for* Π *over $I = [T_s = \langle t_s, n_s \rangle, T_e = \langle t_e, n_e \rangle]$, iff $T \in I$ and at least one of the following holds:*

1. *$T = T_s$ or $T = T_e$; the time points associated with the extremes of the time envelope are always STPs;*
2. *$H_A(T) \neq \langle \rangle$;*
3. *$E_{trigg}(T) \neq \langle \rangle$, i.e., there exists at least an event $\varepsilon \in E$ such that $H_s(T) \models pre(\varepsilon)$;*
4. *there has been a discrete change just before; formally it has to hold that $n > 0$ and there exists a $T' = \langle t, n-1 \rangle$ such that $H_A(T') \neq \langle \rangle$ or $E_{trigg}(T') \neq \langle \rangle$;*
5. *a process has started (stopped) in T; formally $H_A(T) = \langle \rangle$, $E_{trigg}(T) = \langle \rangle$ and there exists $\rho \in P$ for which $H_s(T) \models pre(\rho)$ ($H_s(T) \not\models pre(\rho)$), there exists $T' < T$ such that $T' \in I$ and for each T'' such that $T' \leq T'' < T$ then $H_s(T'') \not\models pre(\rho)$ ($H_s(T'') \models pre(\rho)$).*

We remark that Point 5 of the previous definition, i.e. the condition that determines the starting (stopping) of a process, is to be interpreted as a high-level definition and not an operational one. Specifically, in the case of continuous semantics discussed therein, identifying the exact point where a context switch occurs can be pointless even in simple cases. For example, consider the case in which the precondition of a process requires $\langle x > 0 \rangle$ to be activated and x is linearly incremented starting from 0. For handling cases like this, it is necessary to specialise the provided condition by considering the case in which the context switch involves a closed or open condition. Specifically, in the second case, for providing an operational definition it is necessary to exploit the concept of mathematical limit. As this work focuses primarily on discrete semantics, such a level of detail is outside its scope.

Within an interval, we can also identify a number of sub-intervals where nothing meaningful happens. This is the notion of a monotonous interval. Formally:

Definition 10 (Monotonous interval). *A history H of Π over $I = [T_s = \langle t_s, n_s\rangle, T_e = \langle t_e, n_e\rangle]$ is* monotonous over $I_t = (t_1, t_2) \subseteq (t_s, t_e)$*, where $t_1, t_2 \in \mathbb{R}$, if for each $t \in I_t$ and for any given n, $\langle t, n\rangle$ is not a STP of H.*

Note that, for each $\langle t, n\rangle$ such that $t \in (t_1, t_2)$, the set of active processes does not change and the sequence of triggered events is empty. We denote the set of active processes over I_t as the context $C(I_t) = \{\rho \in P \mid H_S(\langle t_1, n_1\rangle) \models pre(\rho)\}$, where n_1 is a sufficiently large natural number beyond which the state is stable, i.e., for each step $n \geq n_1$ then $H_A(\langle t_1, n\rangle) = \langle\rangle$ and $E_{trigg}(\langle t_1, n\rangle) = \langle\rangle$. A context switch happens when the set of active processes changes.

In order to guarantee that there is only a unique and finite set of significant time points given a history, it is necessary to impose some restrictions on its structure. In what follows we will go over the issues caused by events and context switches, and conclude with a number of assumptions that are needed to have a meaningful and manageable notion of plan projection, which is what we use to infer whether a plan is valid or not. Similar restrictions have been used also in other works on PDDL+ (Fox, Howey, & Long, 2005; Shin & Davis, 2005; Cashmore, Magazzeni, & Zehtabi, 2020).

We start off this discussion with an example showing how the general formulation of PDDL+ can induce non-deterministic spontaneous transitions.

Example 2.1 (Non-deterministic Events). *Let $\varepsilon_1 = \langle\langle x = 0\rangle, \{\langle asgn, y, 1\rangle, \langle asgn, x, 1\rangle\}\rangle$ and $\varepsilon_2 = \langle\langle x = 0\rangle, \{\langle asgn, y, -1\rangle, \langle asgn, x, 1\rangle\}\rangle$ be two events and let s be a state such that $s \models \langle x = 0\rangle$. The set of triggered events in s is $\{\varepsilon_1, \varepsilon_2\}$, since $s \models pre(\varepsilon_1) \wedge pre(\varepsilon_2)$. Such a set can be sequenced into $\langle\varepsilon_1, \varepsilon_2\rangle$ or $\langle\varepsilon_2, \varepsilon_1\rangle$ and, depending on the chosen ordering, two different outcomes can be obtained, i.e., $s' = \gamma(s, \langle\varepsilon_1, \varepsilon_2\rangle) = \gamma(\gamma(s, \varepsilon_1), \varepsilon_2) \models \langle y = 1\rangle$ and $s'' = \gamma(s, \langle\varepsilon_1, \varepsilon_2\rangle) = \gamma(\gamma(s, \varepsilon_2), \varepsilon_1) \models \langle y = -1\rangle$. A problem including these events is said to be* non-deterministic *as different sequences of triggered events can produce different resulting states.*

To avoid this possibility, Fox and Long (2006) provided the definition of an event-deterministic PDDL+ problem. This definition requires that in any state where more than one event is triggered, the outcome resulting from their sequential execution is independent of their ordering.

Definition 11 (Event-deterministic PDDL+ Problem (Fox & Long, 2006)). *A PDDL+ problem Π is said to be* event-deterministic *if, for each state where two events ε_1 and ε_2 are triggered, then the transition sequences induced by $\langle\varepsilon_1, \varepsilon_2\rangle$ and $\langle\varepsilon_2, \varepsilon_1\rangle$ are both valid and reach the same state. In this case, ε_1 and ε_2 are said to commute.*

Determining whether a PDDL+ problem is event-deterministic or not may be expensive, but it is possible to establish a sufficient criterion that ensures that this always holds. For example, two non-mutex events always commute. Indeed, if every pair of events that are triggered in any state commute, then the problem is event-deterministic. In the following, we assume that PDDL+ problems are event-deterministic, which implies the guarantee that any order of evaluation chosen from the triggered events in a state leads to the same state.

Another source of complexity comes with a potentially infinite cascade of events. A cascade of events occurs when a state $H_S(\langle t, n\rangle)$ triggers at least an event, i.e., $E_{trigg}(\langle t, n\rangle) =$

$\langle\varepsilon_1\rangle$, such that the resulting state obtained by the application of ε, i.e., $H_s(\langle t, n+1\rangle) = \gamma(H_s(\langle t, n\rangle), \langle\varepsilon_1\rangle)$, triggers again at least an event, i.e., $H_s(\langle t, n+1\rangle) \models pre(\varepsilon_2)$.

Hereinafter, given two sequences of elements V and V', we denote as $V \oplus V'$ the sequential merging of V and V'. For example, given $V = \langle 1, 2\rangle$ and $V' = \langle 2, 3\rangle$, then $V \oplus V' = \langle\langle 1, 2\rangle, \langle 2, 3\rangle\rangle$. The $\oplus$ operator is associative and the identity element is the empty sequence, so $V \oplus \langle\rangle = V$.

Definition 12 (Cascade of Events)**.** *Let* Π *be a* PDDL+ *problem,* H *be a history for* Π *over* I *and let* $T = \langle t, n\rangle \in I$*. The* cascade of events *triggered in* T *is defined recursively as a sequence of sequences as follows:*

$$ES(T) = \begin{cases} \langle\rangle & \text{if } E_{trigg}(T) = \langle\rangle \\ E_{trigg}(\langle t, n\rangle) \oplus ES(\langle t, n+1\rangle) & \text{otherwise} \end{cases}$$

To give an intuition of the definition, look at the following small example.

Example 2.2 (Cascade of Events)**.** *Let* Π *be a* PDDL+ *problem encompassing three events,* $E = \{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$*, and* $\varepsilon_1 = \langle\langle x = 0\rangle, \{\langle asgn, x, 1\rangle\}\rangle$*,* $\varepsilon_2 = \langle\langle x = 1\rangle, \{\langle asgn, x, 2\rangle\}\rangle$ *and* $\varepsilon_3 = \langle\langle x = 2\rangle, \{\langle asgn, x, 3\rangle\}\rangle$*. Let* H *be a history for* Π *over* I *and let* $T = \langle t, n\rangle \in I$ *be a STP such that* $H_s(\langle t, n\rangle) \models \langle x = 0\rangle$*. Then, by iterating recursively over the states obtained executing the events, we get that:*

- $H_s(\langle t, n\rangle) \models \langle x = 0\rangle \models pre(\varepsilon_1) \wedge \neg pre(\varepsilon_2) \wedge \neg pre(\varepsilon_3)$*, then* $E_{trigg}(\langle t, n\rangle) = \langle\varepsilon_1\rangle$*;*
- $H_s(\langle t, n+1)\rangle = \gamma(H_s(\langle t, n\rangle), E_{trigg}(\langle t, n\rangle) = \langle\varepsilon_1\rangle) \models \langle x = 1\rangle \models \neg pre(\varepsilon_1) \wedge pre(\varepsilon_2) \wedge \neg pre(\varepsilon_3)$*, then* $E_{trigg}(\langle t, n+1\rangle) = \langle\varepsilon_2\rangle$*;*
- $H_s(\langle t, n+2)\rangle = \gamma(H_s(\langle t, n+1\rangle), E_{trigg}(\langle t, n+1\rangle) = \langle\varepsilon_2\rangle) \models \langle x = 2\rangle \models \neg pre(\varepsilon_1) \wedge \neg pre(\varepsilon_2) \wedge \neg pre(\varepsilon_3)$*, then* $E_{trigg}(\langle t, n+1\rangle) = \langle\varepsilon_3\rangle$*;*
- $H_s(\langle t, n+3)\rangle = \gamma(H_s(\langle t, n+2\rangle), E_{trigg}(\langle t, n+2\rangle) = \langle\varepsilon_3\rangle) \models \langle x = 3\rangle \models \neg pre(\varepsilon_1) \wedge \neg pre(\varepsilon_2) \wedge \neg pre(\varepsilon_3)$*, then* $E_{trigg}(\langle t, n+3\rangle) = \langle\rangle$*.*

The cascade of events triggered in T *is defined as* $ES(T) = E_{trigg}(\langle t, n\rangle) \oplus ES(\langle t, n+1\rangle) = E_{trigg}(\langle t, n\rangle) \oplus E_{trigg}(\langle t, n+1\rangle) \oplus ES(\langle t, n+2\rangle) = E_{trigg}(\langle t, n\rangle) \oplus E_{trigg}(\langle t, n+1\rangle) \oplus E_{trigg}(\langle t, n+2\rangle) \oplus ES(\langle t, n+3\rangle) = \langle\varepsilon_1\rangle \oplus \langle\varepsilon_2\rangle \oplus \langle\varepsilon_3\rangle \oplus \langle\rangle = \langle\langle\varepsilon_1\rangle, \langle\varepsilon_2\rangle, \langle\varepsilon_3\rangle\rangle$*.*

As hinted at above, Definition 12 allows the characterisation of an infinite cascade of events. Fox et al. (2005) discusses limitations to be imposed on the structure of events, in addition to those described above concerning the non-determinism, in order to prevent such an infinite cascade of events from actually happening. The rule is that every event has to be *self-deactivating*. An event is self-deactivating if it falsifies its preconditions when it is applied, i.e., for each state $s \models pre(\varepsilon)$ then $\gamma(s, \varepsilon) \models \neg pre(\varepsilon)$. This is necessary to prevent a triggered event from continuing to trigger indefinitely. It should be noted that there can also be more complex cyclic-triggering situations involving more than a single event. For this reason, it is assumed that an event is triggered at most once in a timestamp (Fox et al., 2005).

Our assumptions construct on the above notions and can be then summarised as follows:

Assumption 1 (Event-determinism)**.** *The* PDDL+ *problem* Π *has to be event-deterministic (see Definition 11).*

Assumption 2 (Finite Complexity)**.** *The* PDDL+ *problem* Π *has to induce a finite number of spontaneous changes over an interval* I_t*. This finiteness is imposed both on the number of events that can be triggered, and the number of times the starting and the stopping of a process causes a change of context.*

It is easy to see that the above assumptions lead us to get a unique and finite number of STPs over an interval I_t.

We are now ready to define the projection of a PDDL+ plan. Intuitively, we define the plan projection around a number of rules that specify how history evolves over time. The first (second) rule states that if an action (event) is executed (triggered) in an STP $T = \langle t, n \rangle$, then there necessarily exists a successor of T, i.e., $T' = \langle t, n+1 \rangle$ having the same clock t and the step increased by one unit, i.e., $n+1$; the successor state associated to T' is calculated by simply applying the discrete effects of the actions (events). The third rule is used to enforce how actions of a PDDL+ plan π are projected over a history, preserving their original ordering in case they share the same timestamp in π. Then, the fourth rule is used to enforce how a numeric variable changes continuously over time according to the active processes in those "monotonous" temporal intervals in which "nothing happens" (there is no action/event executed/triggered and there is no process which starts/ends). The following definition formalises this intuition, assuming $\Pi = \langle F, X, I, G, A, E, P \rangle$, and $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ implicit.

Definition 13 (PDDL+ Plan Projection)**.** *Let* H *be a history for* Π *over* I*, let* I *be an initial state and let* π_t *be a* PDDL+ *plan for* Π*. We say that* H *is* a projection of π_t which starts in I *iff* H *induces a finite sequence of STPs* $T_H = \langle T_0 = \langle t_s, 0 \rangle, ..., T_m = \langle t_e, n_m \rangle \rangle$ *such that* H *is defined over* $I = [T_0, T_m]$ *with* $H_s(T_0) = I$*,* $H_A(T_m) = \langle \rangle$*,* $E_{trigg}(T_m) = \langle \rangle$ *and, for all* $i \in [0..m]$*, the following rules hold:*

R1 $E_{trigg}(T_i) \neq \langle \rangle$ *iff* $H_s(T_{i+1}) = \gamma(H_s(T_i), E_{trigg}(T_i))$*,* $H_A(T_i) = \langle \rangle$*,* $t_{i+1} = t_i$ *and* $n_{i+1} = n_i + 1$*;*

R2 $H_A(T_i) \neq \langle \rangle$ *iff* $H_s(T_{i+1}) = \gamma(H_s(T_i), H_A(T_i))$*,* $E_{trigg}(T_i) = \langle \rangle$*,* $t_{i+1} = t_i$ *and* $n_{i+1} = n_i + 1$*;*

R3 *for each* $\langle a_i, t_i \rangle, \langle a_j, t_j \rangle$ *in* π*, with* $i < j$ *and* $t_i = t_j$ *there exists* T_k, T_z *in* T_H *such that* a_i *in* $H_A(T_k)$ *and* a_j *in* $H_A(T_z)$ *where* $t_k = t_z = t_i$ *and* $n_k < n_z$*;*

R4 H *is monotonous over* $I_t = (t_i, t_{i+1})$*, with* $t_i < t_{i+1}$ *and for each* $x \in X$*, we have that:*

- $H_s(\langle t, 0 \rangle)[x]$ *is continuous and differentiable over* I_t*;*
- $\forall t \in I_t$ *it holds that*

$$\frac{dH_s(\langle t, 0 \rangle)[x]}{dt} = \sum_{\substack{\langle x', \xi \rangle \in e(\rho);\, x' = x \\ \rho \in C(I_t)}} H_s(\langle t, 0 \rangle)[\xi]$$

– $H_s(\langle t_{i+1}, 0\rangle)[x] = \lim_{t \to t_{i+1}} H_s(\langle t, 0\rangle)[x]$, *and values of unaffected variables persist up to* t_{i+1} *(frame-axiom).*

To better understand Definitions 6–13, the example of Figure 1 shows how a numeric variable, in this case, the velocity of a car, changes continuously over time according to a PDDL+ plan. It is remarkable how the PDDL+ problems can induce dynamics on numeric variables having piece-wise defined dynamics with points of discontinuity, as shown for instance in Figure 1. This is why monotonous intervals must necessarily be defined as open intervals and the numeric variable must be continuous and time-differentiable only within them (R4 of Definition 13).

Finally, we remark that, as R3 is defined, for each $T \in I$ then $|H_A(T)| \leq 1$.

Definition 14 (Valid PDDL+ Plan)**.** *Let* Π *be a* PDDL+ *problem. Let* π_t *be a* PDDL+ *plan for* Π *and let* H *be the plan projection of* π_t*;* π_t *is said to be* a valid plan for Π *iff* $H_s(T_m) \models G$ *and the sequence of actions* $H_A(T)$ *is applicable in* $H_s(T)$ *for each* T *in* T_H*.*

Having clarified the continuous semantics for PDDL+, we proceed by adapting the proposed definitions to the discrete case. In the following, as a convention, we distinguish a history generated according to continuous and discrete semantics using H and H, respectively. Furthermore, we use $\delta \in \mathbb{Q}_{>0}$ (where $\mathbb{Q}_{>0} = \{q \in \mathbb{Q} \mid q > 0\}$) as discretisation step.

The following definition of the discrete STP inherits Conditions 1–4 from Definition 9, while Condition 5 is reshaped for detecting when a process has started or stopped in the discrete setting.

Definition 15 (Discrete Significant Time Point)**.** *Let* $\delta \in \mathbb{Q}_{>0}$*.* $T = \langle t, n\rangle$ *is a* STP *of a history* H *for* Π *over* $I = [T_s, T_e]$*, iff* $T \in I$ *and at least one Condition of Definition 9 holds, with Condition 5 reshaped as follows:*

5 a process has started (stopped) in T*; formally* $\mathsf{H}_A(T) = \langle\rangle$ *and* $E_{trigg}(T) = \langle\rangle$ *and there exists* $\rho \in P$ *such that* $\mathsf{H}_s(T) \models pre(\rho)$ *(*$\mathsf{H}_s(T) \not\models pre(\rho)$*) and there exists a* $T' = \langle t', n'\rangle \in I$ *with* $t' = t - \delta$ *such that* $\mathsf{H}_A(T) = \langle\rangle$ *and* $E_{trigg}(T) = \langle\rangle$ *and* $\mathsf{H}_s(T) \not\models pre(\rho)$ *(*$\mathsf{H}_s(T) \models pre(\rho)$*)*

Let ξ be the numeric expression that denotes the contribution to the time derivative of some numeric variable, and let $\delta \in \mathbb{Q}_{>0}$, $\Delta(\xi, \delta) = \xi \cdot \delta$ represents the discretisation of ξ according to δ. For example, let $\langle x, 1.5 \cdot y\rangle$ ($\dot{x} = 1.5 \cdot y$) and $\delta = 2$ be a continuous effect and a discretisation parameter, the discretised expression is $\Delta(1.5 \cdot y, \delta) = 3 \cdot y$.

The following definition of the discrete PDDL+ plan projection inherits Rules 1–3 from Definition 13, while Rule 4 is reshaped to enforce how numeric variables change when time advances by a discrete quantity.

Definition 16 (Discrete PDDL+ Plan Projection)**.** *Let* $\delta \in \mathbb{Q}_{>0}$*, let* H *be a history for* Π *over* I*, let* I *be an initial state and let* π_t *be a* PDDL+ *plan for* Π*. We say that* H *is a discrete projection of* π_t *which starts in* I *iff* H *induces the STPs* $T_{\mathsf{H}} = \langle T_0 = \langle t_s, 0\rangle, ..., T_m = \langle t_e, n_m\rangle\rangle$ *where either* $t_{i+1} = t_i + \delta$ *or* $t_{i+1} = t_i$*, and all rules as for Definition 13 apply, except for R4 that becomes:*

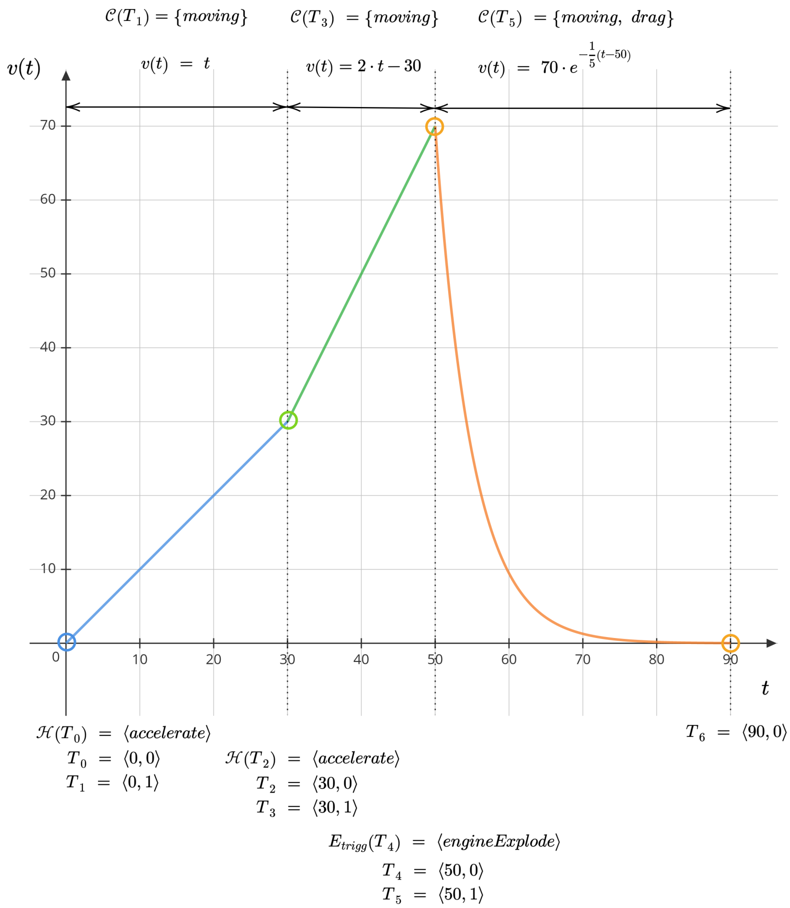

Figure 1: History H of the plan $\pi_t = \langle \pi = \langle\langle accelerate, 0\rangle, \langle accelerate, 30\rangle\rangle, \langle t_s = 0, t_e = 90\rangle\rangle$ for a variant of the Car domain (Fox & Long, 2006). The history generates 7 STPs, i.e., $T_H = \langle T_s = T_0, T_1, T_2, T_3, T_4, T_5, T_e = T_6\rangle$, yielding a piece-wise function consisting of three different monotonous intervals, i.e., $(0, 30)$, $(30, 50)$ and $(50, 90)$. The execution of the action *accelerate* in $T_0 = \langle 0, 0\rangle$ activates the process *moving* in T_1 ($\mathcal{C}(T_1) = \{moving\}$) and then the car increases its speed linearly in $(0, 30)$ (according to $\dot{v} = a = 1$ an consequently to $v(t) = t$). In $(30, 0)$, after the second acceleration is performed in $T_2 = \langle 30, 0\rangle$, the car increases its speed linearly but with a greater slope (according to $\dot{v} = a = 2$ and consequently to $v(t) = 2 \cdot t - 30$). In $T_4 = \langle 50, 0\rangle$ an event is triggered, i.e., the event which models the explosion of the engine when a critical velocity of 70 is reached. The event resets the acceleration activating the process *drag* in $T_5 = \langle 50, 1\rangle$ ($\mathcal{C}(T_5) = \{moving, drag\}$) and deactivating the effect of *moving* on v, which in turns makes the speed decrease exponentially in the interval $(50, 90)$ due to the drag ($\dot{v} = -\frac{1}{5} \cdot v$ and then $v(t) = 70 \cdot e^{-\frac{1}{5}(t-50)}$).

R4 *for each pair of contiguous STPs* $T_i = \langle t_i, n_i \rangle$, $T_{i+1} = \langle t_{i+1}, 0 \rangle$ *such that* $t_{i+1} = t_i + \delta$, *the value of each numeric variable* $x \in X$ *is updated as:*

$$\mathcal{H}_S(T_{i+1})[x] = \mathcal{H}_S(T_i)[x] + \sum_{\substack{\langle x', \xi \rangle \in e(\rho);\, x' = x \\ \rho \in P \text{ such that } \mathcal{H}_S(T_i) \models pre(\rho)}} \mathcal{H}_S(T_i)[\Delta(\xi, \delta)]$$

and values of unaffected variables remain unchanged (frame-axiom).

Note that plans featuring actions at non-discrete time points do not admit any projection, and are therefore ill-defined under discrete interpretation.[4] Furthermore, we remark that the discretisation step $\delta \in \mathbb{Q}_{>0}$ is a given parameter (see discussion in Section 7).

Definition 17 (Discrete Valid PDDL+ Plan)**.** *Let* π_t *be a* PDDL+ *plan and let* $\mathcal{H}$ *be the plan discrete projection of* π_t *for* $\delta \in \mathbb{Q}_{>0}$; π_t *is said to be* a valid plan for Π under δ discretisation *iff* $\mathcal{H}_S(T_m) \models G$ *and the sequence of actions* $\mathcal{H}_A(T)$ *is applicable in* $\mathcal{H}_S(T)$ *for each* T *in* $T_{\mathcal{H}}$.

2.1 Example: The Overtaking-Car Problem

In order to explain our approach, in the remaining of this paper we will make use of a variant of the well-known Linear-Car PDDL+ problem (Fox & Long, 2006), namely Overtaking-Car.

Example 2.3 (Overtaking-Car Problem)**.** *The* Overtaking-Car *domain models a number of cars that can move across two lanes, a fast and a slow lane. The position of each car changes over time when the speed is different from 0; the speed in turn changes for the effect of the acceleration. Each car needs to reach a given distance from the origin while avoiding collisions with other cars; to do so a car can switch from one lane to the other. Let cars denote a finite set of vehicles, a* PDDL+ *planning problem* $\Pi = \langle F, X, I, G, A, E, P \rangle$ *of* Overtaking-Car *can be defined as the following:*

- $F = \{crashed\} \cup \bigcup_{car \in cars} \{ot_{car}, eng\text{-}on_{car}\}$, *where*
  - *crashed models whether a collision among any vehicle has occurred or not. When the variable is set to true, the goal cannot be achieved anymore;*
  - ot_{car} *models whether a "car" is or not in the fast lane. If* ot_{car} *holds the car is overtaking; otherwise it means that the car is moving on the slow lane;*
  - $eng\text{-}on_{car}$ *models whether the engine is on or off;*
- $X = \bigcup_{car \in cars} \{d_{car}, v_{car}, a_{car}\}$, *where the variables* d_{car}, v_{car} *and* a_{car} *express the distance from the origin, the velocity and the acceleration of "car", respectively;*

4. We remark that the plan discrete projection does not model any transient state between two consecutive discrete time points.

- $I = \bigcup_{car \in cars} \{\langle d_{car} := D^{I}_{car}\rangle, \langle v_{car} := 0\rangle, \langle a_{car} := 0\rangle\}$, *where* $D^{I}_{car} \in \mathbb{Q}$*; for each* $car \in cars$*, the speed and the distance are specified in the initial state; by closed world assumption the initial state sets to off all engines' cars, constrains all cars to be positioned in the slow lane and crashed is set to false;*

- $G = \neg crashed \wedge \bigwedge_{car \in cars} \langle v_{car} = 0\rangle \wedge \langle d_{car} = D^{G}_{car}\rangle \wedge \neg eng\text{-}on_{car}$, *where* $D^{G}_{car} \in \mathbb{Q}$*;* G *requires that (i) all cars have stopped (ii) no collisions have occurred (iii) all cars are at a given distance* D^{G}_{car} *from the origin;*

- $A = \bigcup_{car \in cars} \{accelerate_{car}, decelerate_{car}, overtake_{car}, return_{car}, start_{car}, stop_{car}\}$ *where, for each* $car \in cars$*, such actions are defined as:*

  - $accelerate_{car} = \langle eng\text{-}on_{car} \wedge \langle a_{car} < A_{max}\rangle, \{\langle increase, a_{car}, 1\rangle\}\rangle$;
  - $decelerate_{car} = \langle eng\text{-}on_{car} \wedge \langle a_{car} > A_{min}\rangle, \{\langle decrease, a_{car}, 1\rangle\}\rangle$;
  - $overtake_{car} = \langle eng\text{-}on_{car} \wedge \langle v_{car} > 0\rangle \wedge \neg ot_{car}, \{ot_{car}\}\rangle$;
  - $return_{car} = \langle eng\text{-}on_{car} \wedge \langle v_{car} > 0\rangle \wedge ot_{car}, \{\neg ot_{car}\}\rangle$;
  - $start_{car} = \langle \neg eng\text{-}on_{car}, \{eng\text{-}on_{car}\}\rangle$;
  - $stop_{car} = \langle eng\text{-}on \wedge \langle v_{car} = 0\rangle \wedge \langle a_{car} = 0\rangle, \{\neg eng\text{-}on_{car}\}\rangle$.

  The actions $overtake_{car}$ *and* $return_{car}$*, are used to model the movement of "car" from the slow lane to the fast lane and vice versa, respectively.*

- $E = \bigcup_{\substack{car, car' \in cars \\ car \neq car'}} \{crash\text{-}fast_{car,car'}, crash\text{-}slow_{car,car'}\}$*; these events model the possible collisions that can occur between two moving cars, i.e., car and car′, if those are moving in the same lane at a distance less than a critical distance* $D_T \in \mathbb{Q}_{>0}$*. They are defined as follows:*

  - $crash\text{-}fast_{car,car'} = \langle\langle d_{car} - d_{car'} < D_T\rangle \wedge \langle d_{car} - d_{car'} \geq 0\rangle \wedge ot_{car} \wedge ot_{car'} \wedge \neg crashed, \{crashed\}\rangle$;
  - $crash\text{-}slow_{car,car'} = \langle\langle d_{car} - d_{car'} < D_T\rangle \wedge \langle d_{car} - d_{car'} \geq 0\rangle \wedge \neg ot_{car} \wedge \neg ot_{car'} \wedge \neg crashed, \{crashed\}\rangle$;

- $P = \bigcup_{car \in cars} \{moving_{car}\}$ *where, for each* $car \in cars$*, such processes are defined as:*

  - $moving_{car} = \langle eng\text{-}on_{car}, \{\langle v_{car}, a_{car}\rangle, \langle d_{car}, v_{car}\rangle\}\rangle$*, this process models the dynamics of the car: the time derivative of the displacement is the speed (i.e.,* $\dot{d}_{car} = v_{car}$*), whereas the time derivative of the speed is the acceleration (i.e.,* $\dot{v}_{car} = a_{car}$*);*

The numeric constants involved in the definition of I *and* G *are specified such that:*

- *for each pair of cars, i.e.,* $car, car' \in cars$ *with* $car \neq car'$*, then* $|D^{I}_{car} - D^{I}_{car'}| < D_T$*; this constraint prevents a collision event from being triggered in the initial state*

- *for each car* $\in$ *cars then* $D^{G}_{car} > D^{I}_{car}$*;*

- *for each pair of cars, i.e., car,* car' $\in$ *cars with car* $\neq$ car' *and then* $D^{G}_{car} < D^{G}_{car'}$ ($D^{G}_{car} > D^{G}_{car'}$)*; this implies that to achieve the goal, it is necessary to use the fast lane;*

- *the maximum acceleration and deceleration are bounded by two quantities, i.e.,* A_{max} *and* A_{min}*, respectively.*

In Overtaking-Car, the actions to be sought and scheduled are the accelerations and decelerations to reach the goal, while the actions to handle overtaking are used to avoid collisions. Accelerate and decelerate actions have an impact on the speed and therefore on the actual positions of the cars. Such aspects are directly modelled through processes. Each process indeed models how the distance from the origin is affected; this is given as a function of speed that in turn is controlled by the current acceleration.

3. Translating PDDL+ to PDDL2.1

pddl+ problems differ from pddl2.1 ones for the presence of processes and events. In this and the following sections, we devise translations that transform all processes and events into regular actions enriched with additional predicates so that every valid plan that is found in the process and event-free translation, retains its validity on the discretised version of the pddl+ problem it has been generated from. We do so by explicitly formulating a simulation action that lets the planning engine wait and observe the state of the world for a given amount of time. Instead, when the planner picks some action, time does not flow; rather the state is instantaneously modified by the planning engine.

To make this operational, there are a number of challenges to pursue: (i) we need to capture what processes are active in a given state so that the time-discretised continuous update of the state consistently reflects what the dynamical specification of the system prescribes; (ii) we need to take care of the potentially complex cascade of events that may be triggered for each encountered state.

We proceed in a modular fashion. In what follows we firstly present a solution to point (i) with a (straightforward) exponential encoding, and then with a more sophisticated polynomial translation. Both are guaranteed to work for event-free pddl+ problems. Then we face point (ii) by showing how also events can be translated into actions with conditional effects with a translation step that is modular to how we tackle point (i).

3.1 Exponential Translation

Given an event-free pddl+ problem $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$, we define a discrete context $\mathcal{C}$, hereinafter simply referred to as context, to be a non-empty subset of processes, and denote with $\mathcal{P}^{+}(P)$ the set of non-empty subsets of P, that is the set of all possible contexts.

For an event-free pddl+ problem Π, the exponential translation generates a pddl2.1 problem $\Pi_{\text{EXP}} = \langle F, X, I, G, A \cup \{SIM\}, c\rangle$, discretised in δ. Π_{EXP} is almost identical to Π but for the absence of processes and the presence of the special action *SIM* playing the role of the simulator, i.e., what changes when time goes forward. *SIM* is defined as follows:

$$\begin{aligned}
pre(SIM) &= \top \\
eff(SIM) &= \bigcup_{C \in P^{+}(P)} \{contpre(C) \triangleright conteff(C)\}
\end{aligned}$$

where

$$\begin{aligned}
contpre(C) &= \bigwedge_{\rho \in P \setminus C} \neg pre(\rho) \wedge \bigwedge_{\rho \in P \cap C} pre(\rho) \\
conteff(C) &= \bigcup_{x \in X} \{\langle inc, x, \sum_{\langle x', \xi \rangle \in e(\rho);\ \rho \in C;\ x'=x} \Delta(\xi, \delta) \rangle\}
\end{aligned}$$

Intuitively, the action *SIM* organises all possible contexts within a unique action, delegating to each conditional effect (i) the conditions under which a context is triggered and (ii) the consequences that such a context has on the state after some time δ has passed. Point (i) is formalised by conjoining two conjunctions: the first ensures that no other process of some other context has its precondition satisfied ($\bigwedge_{\rho \in P \setminus C} \neg pre(\rho)$); the second ensures that all the preconditions of a given context are satisfied ($\bigwedge_{\rho \in P \cap C} pre(\rho)$). Let x be some numeric variable of our problem, point (ii) is obtained by summing the contribution of each process within the context.

Ultimately, we reflect the effect of the time-passing action on the overall makespan of the plan directly in the cost function of our problem, differentiating therefore whether the planning engine chooses some action, or lets time go for some δ time: $c(a) = 0$ if $a \in A$ while $c(a) = \delta$ if $a = SIM$.

Example 3.1 (EXP Translation - Continuing on Example 2.3)**.** *Let* $\Pi = \langle F, X, I, G, A, E, P \rangle$ *be an* OVERTAKING-CAR *planning instance for the domain reported in Example 2.3 in which* $cars = \{car1, car2\}$*. In this example, we compile* Π *omitting the events (therefore as if it were* $E = \emptyset$*); events will be dealt with later in the paper. The components of the tuple* Π *are therefore:*

$$\begin{aligned}
F =& \{eng\text{-}on_{car1}, eng\text{-}on_{car2}, ot_{car1}, ot_{car2}, crashed\} \\
X =& \{d_{car1}, d_{car2}, v_{car1}, v_{car2}, a_{car1}, a_{car2}\}; \\
I =& \{\langle d_{car1} := D^{I}_{car1} \rangle, \langle d_{car2} := D^{I}_{car2} \rangle, \langle v_{car1} := 0 \rangle, \langle v_{car2} := 0 \rangle, \langle a_{car1} := 0 \rangle, \langle a_{car2} := 0 \rangle\}; \\
G =& \neg crashed \wedge \langle d_{car1} = D^{G}_{car1} \rangle \wedge \langle d_{car2} = D^{G}_{car2} \rangle \wedge \langle v_{car1} = 0 \rangle \wedge \langle v_{car2} = 0 \rangle \wedge \neg eng\text{-}on_{car1} \wedge \\
& \neg eng\text{-}on_{car2}; \\
A =& \{accelerate_{car1}, accelerate_{car2}, overtake_{car1}, overtake_{car2}, return_{car1}, return_{car2}, \\
& start_{car1}, start_{car2}, stop_{car1}, stop_{car2}\}; \\
E =& \emptyset; \\
P =& \{moving_{car1}, moving_{car2}\}.
\end{aligned}$$

The EXP *reformulation of* Π *discretised in* δ *is* $\Pi_{\text{EXP}} = \langle F, X, I, G, A \cup \{SIM\}, c \rangle$ *where* *SIM is the simulation action as for* EXP *specification. Let* W_C *be the single conditional effect* *of SIM such that $C \in \mathcal{P}^+(P)$ is a context, and let "m1" and "m2" be short for processes "moving$_{car1}$" and "moving$_{car2}$". The SIM action is as follows:*

$$pre(SIM) = \top$$
$$eff(SIM) = \{W_{\{m1\}}, W_{\{m2\}}, W_{\{m1;m2\}}\}$$

where

$$W_{\{m1\}} = eng\text{-}on_{car_1} \wedge \neg eng\text{-}on_{car_2} \triangleright \{\langle inc, d_{car1}, v_{car1} \cdot \delta\rangle, \langle inc, v_{car1}, a_{car1} \cdot \delta\rangle\}$$
$$W_{\{m2\}} = \neg eng\text{-}on_{car_1} \wedge eng\text{-}on_{car_2} \triangleright \{\langle inc, d_{car2}, v_{car2} \cdot \delta\rangle, \langle inc, v_{car2}, a_{car2} \cdot \delta\rangle\}$$
$$W_{\{m1;m2\}} = eng\text{-}on_{car_1} \wedge eng\text{-}on_{car_2} \triangleright \{\langle inc, d_{car1}, v_{car1} \cdot \delta\rangle, \langle inc, v_{car1}, a_{car1} \cdot \delta\rangle, \langle inc, d_{car2}, v_{car2} \cdot \delta\rangle, \langle inc, v_{car2}, a_{car2} \cdot \delta\rangle\}.$$

In the following lemma, we show the soundness and completeness properties of the EXP translation for a PDDL+ event-free problem. By completeness and soundness, we mean that if Π admits a valid solution then the corresponding problem Π_{EXP} admits one solution too, and vice-versa.

Lemma 1 (Soundness and Completeness of EXP for an Event-free PDDL+ Problem)**.** *Let $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$ be a PDDL+ problem, and let $\Pi_{\text{EXP}} = \langle F, X, I, G, A \cup \{SIM\}, c\rangle$ be the PDDL2.1 problem obtained by using the EXP translation discretised in δ. Π admits a solution under δ discretisation iff so does Π_{EXP}.*

Proof. ($\Rightarrow$) Let $\pi_t = \langle \pi, \langle 0, t_e\rangle\rangle$ be a valid solution for Π (assume w.l.o.g. $t_s = 0$) under δ discretisation and let π_{EXP} be a PDDL2.1 plan constructed in such a way that: (i) for each $\langle a, t\rangle$ in π then a' is in π_{EXP} (where a' is the compiled version of a); (ii) for each $\langle a_i, t_i\rangle, \langle a_j, t_j\rangle$ with $a_i \prec a_j$ in π then $a'_i \prec a'_j$ holds in π_{EXP} and (iii) a sequence, possibly empty, of SIM actions has to be placed before each action $a'_i \in \pi_{\text{EXP}}$ and at the end of π_{EXP} according to the following structure

$$\pi_{\text{EXP}} = \langle \langle SIM\rangle \times \frac{t_0}{\delta}, a'_0, \langle SIM\rangle \times \frac{t_1 - t_0}{\delta}, ..., a'_{n-1}, \langle SIM\rangle \times \frac{t_e - t_{n-1}}{\delta}\rangle$$

where $\langle SIM\rangle \times k$, with $k \in \mathbb{N}$, indicates k repetitions of *SIM*.

In order to prove that π_{EXP} is a valid solution for Π_{EXP}, it suffices to show that, let $\tau = \langle \mathcal{H}_s(T_0), ..., \mathcal{H}_s(T_m)\rangle$ be the sequence of states associated to each STP of $\mathcal{H}$, and $\tau' = \langle s_0, ..., s_m\rangle$ be the sequence of states generated by iteratively executing π_{EXP}, $\mathcal{H}_s(T_i)$ and s_i are equivalent (agree on all values for $F \cup X$) for each $i \in [0..m]$. We prove this by induction on τ (τ').

Firstly, we have to show how $n = |\pi|$ and $m = |\tau| - 1$ are related and then that $|\tau| = |\tau'|$ holds. Given π, according to R1 of Definition 16, we have n instantaneous transitions and, according to R4, we have $\frac{t_e - t_s}{\delta} = \frac{t_e}{\delta}$ temporal transitions, i.e., the number of discrete advancements of time. It follows that $m = n + \frac{t_e - t_s}{\delta}$. As shown in π_{EXP} definition, there is an action $a'_i \in \pi_{\text{EXP}}$ for each action $a_i \in \pi$ with $i \in [0..n-1]$, and a *SIM* action for each discrete advancement of time, i.e., $\frac{t_e}{\delta}$. Then $|\pi_{\text{EXP}}| = n + \frac{t_e}{\delta} = m$. Then π_{EXP} induces m transitions from I, i.e., $|\tau'| = m + 1$, and finally we got that $|\tau| = |\tau'|$.

The base case ($i = 0$) trivially proves true as $H_S(T_0) = I$ and $s_0 = I$. For the induction step, we assume truly the statement for some $i < |\tau|$, and prove this for $i+1$ by considering the two types of transitions occurring between two contiguous STPs in H.

Instantaneous transition. Let $T_i = \langle t_i, n_i \rangle$ and $T_{i+1} = \langle t_i, n_i + 1 \rangle$ be two STPs of H. Rule 2 of Definitions 13-16 implies that $H_A = \langle a_i \rangle \neq \langle \rangle$. Since π_t is a valid solution for Π, we know that $H_S(T_i) \models \mathit{pre}(a_i)$ therefore, by inductive hypothesis, $s_i \models \mathit{pre}(a'_i)$. Since a_i and a'_i are the same operator, it is easy to see that the outcomes of the transitions $\gamma(H_S(T_i), a_i)$ and $\gamma(s, a'_i)$ are equivalent.

Temporal transition. Let i be an index such that $T_i = \langle t_i, n_i \rangle$ and $T_{i+1} = \langle t_i + \delta, 0 \rangle$ are two STPs of H.

Note that the *SIM* action features a conditional effect for each possible context $C \in P^+(P)$. By definition of $P^+(P)$, contexts differ from one another for at least one process. It follows that, by definition of $\mathit{contpre}(\cdot)$, for each $C, C' \in P^+$ with $C \neq C'$, $\mathit{contpre}(C)$ and $\mathit{contpre}(C')$ are mutually exclusive. Therefore, for each application of the *SIM* action, at most one conditional effect is activated.

We denote with SIM_i the application of *SIM* in the i-th state of τ'. State s_i induces the context $C(s_i) = \{\rho \in P,\ s_i \models \mathit{pre}(\rho)\}$ and then $\mathit{SIM}_i = \langle \top, \mathit{contpre}(C(s_i)) \triangleright \mathit{conteff}(C(s_i)) \rangle = \langle \mathit{contpre}(C(s_i)), \mathit{conteff}(C(s_i)) \rangle = \langle \top, \mathit{conteff}(C(s_i)) \rangle$; indeed, we can remove each conditional effect that does not hold in s_i. To show that the state produced by the application of SIM_i in s_i is equivalent to that produced according to the semantics of the discrete projection of π_t, when a quantum of δ time passes, it is sufficient to focus on a single variable $x \in X$ that is affected by the process, and then generalise to all the numeric variables.[5] In particular, action SIM_i modifies x according to:

$$\langle inc, x, \sum_{\substack{\langle x', \xi \rangle \in e(\rho);\, x'=x \\ \rho \in C(s_i)}} \Delta(\xi, \delta) \rangle$$

which corresponds to enforcing the transition as follows:

$$s_{i+1}[x] = s_i[x] + \sum_{\substack{\langle x', \xi \rangle \in e(\rho);\, x'=x \\ \rho \in C(s_i)}} s_i[\Delta(\xi, \delta)] \tag{1}$$

According to Rule 4 of Definition 16, each numeric variable $x \in X$ changes over δ by using the active processes in $C(H_S(T_i)) = \{\rho \in P,\ H_S(T_i) \models \mathit{pre}(\rho)\}$ according to:

$$H_S(T_{i+1})[x] = H_S(T_i)[x] + \sum_{\substack{\langle x', \xi \rangle \in e(\rho);\, x'=x \\ \rho \in C(H_S(T_i))}} H_S(T_i)[\Delta(\xi, \delta)] \tag{2}$$

Since $H_S(T_i)$ is equivalent to s_i, by the inductive hypothesis, then the induced contexts are the same, i.e., $C(s_i) = C(H_S(T_i))$. It follows that the right-hand side expressions of Formulae 1-2 are equivalent. Thus, $H_S(T_{i+1})$ and $s_{i+1} = \gamma(s_i, \mathit{SIM})$ are equivalent.

5. Note that, the study of the equivalence of s_{i+1} and $H_S(T_{i+1})$ can be circumscribed to studying the numeric variables X and ignoring the propositional ones, since the P processes of can only affect numeric variables. Same for the conditional effects of *SIM* they are built from.

($\Leftarrow$) Note that every plan solving Π_{EXP} is structured alternating an agent's action and a sequence (possibly empty) of *SIM* actions. Let $\pi_{\text{EXP}} = \langle\langle SIM\rangle^{\frac{t_0}{\delta}}, a'_0, \langle SIM\rangle^{\frac{t_1-t_0}{\delta}}, ..., a'_{n-1}, \langle SIM\rangle^{\frac{t_e-t_{n-1}}{\delta}}\rangle$ be such a plan. We can construct a valid PDDL+ plan $\pi_t = \langle \pi, \langle 0, t_e\rangle\rangle$ as follows: (i) for each action a'_i in π_{EXP} such that $a'_i \neq SIM$ then $\langle a_i, t_i\rangle$ is in π, where t_i is equal to δ multiplied for the occurrences of *SIM* in π_{EXP} before a'_i; (ii) for each a'_i, a'_j such that $a'_i \prec a'_j$ in π_{EXP} then holds $\langle a_i, t_i\rangle \prec \langle a_j, t_j\rangle$ in π and iii) t_e is equal to δ multiplied by the number of *SIM* in π_{EXP}. The plan we get by using this construction is:

$$\pi_t = \langle\langle \pi = \langle a_0, t_0\rangle, \langle a_1, t_1\rangle, ..., \langle a_{n-1}, t_{n-1}\rangle\rangle, \langle 0, t_e\rangle\rangle$$

In order to show the validity of π_t, we reason on the discrete projection H of π_t that, using Definition 16, is determined as follows. Every action $\langle a_i, t_i\rangle$ in π is associated with a STP $T = \langle a_i, n_i\rangle$ such that $\mathsf{H}_A(T) = \langle a_i\rangle$. This implies, by using Rule 1 of Definitions 13-16, the existence of a STP following T, i.e., $T' = \langle t_i, n_i + 1\rangle$, such that $\mathsf{H}_S(T') = \gamma(\mathsf{H}_S(T), a_i)$. Any pair of contiguous actions, having the form $\langle a_i, t_i\rangle$ and $\langle a_{i+1}, t_{i+1}\rangle$, induces $k = \frac{t_{i+1}-t_i}{\delta}$ STPs of the form $\langle t_i + j \cdot \delta, 0\rangle$ with $j \in [1..k]$. The number of STPs so induced equates the number of *SIM* actions between a'_i and a'_{i+1}. Overall H encompasses the following sequence of STPs:

$$T_{\mathsf{H}} = \langle \underbrace{\overbrace{T_0, ..., T_{\frac{t_0}{\delta}}}^{\frac{t_0}{\delta}\text{ STPs}}, T_{\frac{t_0}{\delta}+1}}_{a_0}, ..., \underbrace{\overbrace{T_{i+\frac{t_i}{\delta}}, ..., T_{i+\frac{t_{i+1}}{\delta}}}^{\frac{t_{i+1}-t_i}{\delta}\text{ STPs}}, T_{i+1+\frac{t_{i+1}}{\delta}}}_{a_i}, ..., \underbrace{T_{n-1+\frac{t_{n-1}}{\delta}}, \overbrace{T_{n+\frac{t_{n-1}}{\delta}}}_{a_{n-1}}{}^{\frac{t_e-t_{n-1}}{\delta}\text{ STPs}}, ..., T_{n+\frac{t_e}{\delta}}\rangle$$

We have obtained that the number of STPs of H equates the number of actions of π_{EXP}, i.e., $|T_{\mathsf{H}}| = |\pi| + \frac{t_e}{\delta}$, where $|\pi|$ is the number of non *SIM* action in π_{EXP} and $\frac{t_e}{\delta}$ is the number of *SIM* actions in π_{EXP}; therefore $|T_{\mathsf{H}}| = |\pi_{\text{EXP}}|$. To prove that π_t is a valid plan for Π it suffices to observe that the state where an action from A is applied in H is equivalent to the state where the same action is applied in the sequence of states induced by π_{EXP}. Therefore, because π_{EXP} achieves the goal in Π_{EXP} so does π in Π; this can be proved formally in a way that is very similar to what is done for the opposite direction.

□

3.2 Polynomial Translation

The translation that we present in this section relies on the idea to keep the factored processes-based representation and to reflect this into a number of actions (in the next organised in set A_P), each devoted to model whether one particular process is active, and what its effects on the state of the world are. Instead of evaluating which context holds by exploiting an exponentially large set of possible alternatives, we let the planning engine develop the applications of actions from A_P in depth, i.e., by sequencing their execution one after the other. This sequencing gets activated when the planning engine switches into simulation modality, activated by the so-called action *start* and ends once all actions from A_P are executed.

Since the A_P actions could influence each other by affecting the numeric variables, *start* also makes a full copy of the numeric state values before starting the simulation. It does so by assigning the current value of all variables into a new set of variables, i.e., X^{cp}, before any process starts operating. This lets the planning engine safely evaluate all variables each process effect depends on. In order to achieve this last point, each numeric effect in some process is rewritten before being transformed into an action. The rewriting manipulates each formula in a way to substitute every occurrence of a variable from X with its compilation image in X^{cp}.[6] Let ξ be a formula, we denote with $\sigma(\xi, X^{cp})$ the result of such a rewriting.

This translation trades the exponential blow-up caused by the context-switching operation of EXP, with an increase in the length of the plan. The resulting formulation is sound, complete and, more interestingly, is polynomial on the size of the input. For this reason, we call this translation POLY and detail in what follows its precise definition and functioning.

Let $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$ and $\delta \in \mathbb{Q}_{>0}$ be an event-free PDDL+ problem and a discretisation parameter. POLY generates a new PDDL2.1 problem $\Pi_{\text{POLY}} = \langle F \cup D \cup \{pause\}, X \cup X^{cp}, I, G \wedge \neg pause, A_c \cup A_P \cup \{start, end\}, c\rangle$ such that:

$$X^{cp} = \{x^{copy} \mid x \in X\}$$

$$D = \bigcup_{\substack{ne \in e(\rho)\\ \rho \in P}} \{done_{ne}\}$$

$$A_c = \{\langle pre(a) \wedge \neg pause, eff(a)\rangle \mid a \in A\}$$

$$start = \langle \neg pause, \{pause\} \cup \bigcup_{x \in X} \{\langle asgn, x^{copy}, x\rangle\}\rangle$$

$$end = \langle \bigwedge_{done \in D} done \wedge pause, \{\neg pause\} \cup \bigcup_{done \in D} \{\neg done\}\rangle$$

$$A_P = \bigcup_{\substack{ne:\langle x, \xi\rangle \in e(\rho)\\ \rho \in P}} \{\langle pause \wedge \neg done_{ne}, \{\sigma(pre(\rho), X^{cp}) \triangleright \{\langle inc, x, \Delta(\delta, \sigma(\xi, X^{cp}))\rangle\}\} \cup \{done_{ne}\}\rangle\}$$

As it is possible to observe, at any discrete time step, the planning engine can decide to let time pass by an amount of δ and does so by deciding to execute action *start*. From that moment onward, no action from A_c can be executed, and only when all conditional process effects are applied, the planning engine can come back into actual planning mode ($\neg pause$). A_P encompasses all such processes effects, and delegates to a conditional effect the check and the consequent update of the variables according to their past value ($\sigma(pre(\rho), X^{cp})$ for the precondition of the process, and $\sigma(\xi, X^{cp})$ for the right-hand side of the numeric effect) under the proper Δ discretisation. Note that, action *start* also ensures that all the variables are copied through an assignment operation, which is responsible for iterating over all numeric variables of the problem and updating their value for the next round of simulation (the snippet $\bigcup_{x \in X} \{\langle asgn, x^{copy}, x\rangle\}$).

Similarly to the exponential translation, also in this case we make the planning engine aware of the passage of time through the cost function. As, however, we do not have only one action that reflects such a passage of time, we attribute a non-zero cost only to action *start*. That is: $c(a) = \delta$ if $a = start$, 0 otherwise.

Example 3.2 (POLY Translation - Continuing on Example 2.3)**.** *Let* $\Pi = \langle F, X, I, G, A, E, P\rangle$ *be an* OVERTAKING-CAR *planning instance for the domain reported in Example 2.3 in which* $cars = \{car1, car2\}$*. Again, let us assume for now that* $E = \emptyset$*.*

6. Recall that every numeric expression is a formula.

The process labels are shortened using the same convention provided in Example 3.1. However, if a process admits more than one continuous numeric effect, another convention is introduced. For example, consider the grounded process moving$_{car1}$ *having two numeric continuous effects, i.e.,* $\langle d_{car1}, v_{car1}\rangle$ *and* $\langle v_{car1}, a_{car1}\rangle$*. Such effects are labelled as "*m_{d1}*" and "*m_{v1}*", respectively. That is, "*m_{d1}*" is the numeric continuous effects of moving*$_{car1}$ *affecting* d_1 *whereas "*m_{v1}*" is the numeric continuous effects of moving*$_{car1}$ *affecting* v_1*.*

The polynomial reformulation of Π *discretised in* δ *is* $\Pi_{\text{POLY}} = \langle F \cup D \cup \{pause\}, X \cup X^{cp}, I, G \cup \{\neg pause\}, A_c \cup A_P \cup \{start, end\}\rangle$ *where:*

$$D = \bigcup_{\substack{ne \in e(\;)\\ \in\{m1,m2\}}} \{done_{ne}\} = \{done_{m_{d1}}, done_{m_{v1}}, done_{m_{d2}}, done_{m_{v2}}\}$$

$$X^{cp} = \bigcup_{car \in \{car1,car2\}} \{d^{copy}_{car}, v^{copy}_{car} a^{copy}_{car}\} = \{d^{copy}_{car1}, d^{copy}_{car2}, v^{copy}_{car1}, v^{copy}_{car2}, a^{copy}_{car1}, a^{copy}_{car2}\}$$

$$\begin{aligned}
start =& \langle \neg pause, \{pause\} \cup \{\langle asgn, d^{copy}_{car1}, d_{car1}\rangle, \langle asgn, d^{copy}_{car2}, d_{car2}\rangle, \langle asgn, v^{copy}_{car1}, v_{car1}\rangle,\\
& \langle asgn, v^{copy}_{car2}, v_{car2}\rangle, \langle asgn, a^{copy}_{car1}, a_{car1}\rangle, \langle asgn, a^{copy}_{car2}, a_{car2}\rangle\}\rangle\\
end =& \langle done_{m_{d1}} \wedge done_{m_{v1}} \wedge done_{m_{d2}} \wedge done_{m_{v2}} \wedge pause,\\
& \{\neg done_{m_{d1}} \wedge \neg done_{m_{v1}} \wedge \neg done_{m_{d2}} \wedge \neg done_{m_{v2}} \wedge \neg pause\}\rangle\\
A_P =& \{a_{m_{d1}}, a_{m_{v1}}, a_{m_{d2}}, a_{m_{v2}}\}\\
a_{m_{d1}} =& \langle pause \wedge \neg done_{m_{d1}}, \{engine\text{-}on_{car1} \rhd \{\langle inc, d_{car1}, v^{copy}_{car1}\ \delta\rangle\}\} \cup \{done_{m_{v1}}\}\rangle\\
a_{m_{v1}} =& \langle pause \wedge \neg done_{m_{v1}}, \{engine\text{-}on_{car1} \rhd \{\langle inc, v_{car1}, a^{copy}_{car1}\ \delta\rangle\}\} \cup \{done_{m_{d1}}\}\rangle\\
a_{m_{d2}} =& \langle pause \wedge \neg done_{m_{d2}}, \{engine\text{-}on_{car2} \rhd \{\langle inc, d_{car2}, v^{copy}_{car2}\ \delta\rangle\}\} \cup \{done_{m_{d2}}\}\rangle\\
a_{m_{v2}} =& \langle pause \wedge \neg done_{m_{v2}}, \{engine\text{-}on_{car2} \rhd \{\langle inc, v_{car2}, a^{copy}_{car2}\ \delta\rangle\}\} \cup \{done_{m_{v2}}\}\rangle
\end{aligned}$$

Note that the use of X^{cp} *variables is crucial to make deterministic the outcome of the time-passage simulation sequence, i.e.,* $\langle start, seq(A_P), end\rangle$*, regardless of the chosen ordering in* $seq(A_P)$*.*

In the following lemma, we show that POLY is sound and complete for a PDDL+ event-free problem.

Lemma 2 (Soundness and Completeness of POLY for an Event-free PDDL+ Problem)**.** *Let* $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$ *be a* PDDL+ *problem, and let* $\Pi_{\text{POLY}} = \langle F \cup D \cup \{pause\}, X \cup X^{cp}, I, G \cup \{\neg pause\}, A_c \cup A_P \cup \{start, end\}\rangle$ *be the* PDDL2.1 *problem obtained by using the* POLY *translation discretised in* δ*.* Π *admits a solution under* δ *discretisation iff so does* Π_{POLY}*.*

Proof. ($\Rightarrow$) Let $\pi_t = \langle \pi, \langle 0, t_e\rangle\rangle$ be a valid solution for Π (assume w.l.o.g. $t_s = 0$) under δ discretisation, and let π_{POLY} be a PDDL2.1 plan constructed in such a way that: (i) for each $\langle a, t\rangle$ in π then a' is in π_{POLY} (where a' is the compiled version of a); (ii) for each $\langle a_i, t_i\rangle, \langle a_j, t_j\rangle$ with $a_i \prec a_j$ in π then $a'_i \prec a'_j$ holds in π_{POLY} (iii) a sequence, possibly empty, of sequences having the form $wait = \langle start, seq(A_P), end\rangle$ (where $seq(A_P)$ is any sequencing of all A_P operators) has to be placed before each action a'_i *in* π_{POLY} and at the end of π_{POLY} according to the following structure:

$$\pi_{\text{POLY}} = \langle \langle wait\rangle^{\frac{t_0}{\delta}}, a'_0, \langle wait\rangle^{\frac{t_1 - t_0}{\delta}}, ..., a'_{n-1}, \langle wait\rangle^{\frac{t_e - t_{n-1}}{\delta}} \rangle$$

Much as we do for Lemma 1, we proceed by induction over the states τ and τ', noticing that τ' can be constructed using $wait$ as if it was a single transition, therefore leading us to have $|\tau| = |\tau'|$. Differently from Lemma 1, Π and Π_{POLY} have different variables, i.e., the set of variables of Π_{POLY} contains the set of variables of Π. Hence, given two elements of τ and τ', i.e., s_i and $\mathsf{H}_S(T_i)$, we say that they are equivalent if they match over the variables of Π, i.e., $F \cup X$ and $\mathsf{H}_S(T_i) \models \neg pause$.

The base case ($i = 0$) trivially proves true as $\mathsf{H}_S(T_0) = I \models \neg pause$ and $s_0 = I$. The *instantaneous transition* is similar to what has been discussed in Lemma 1 since a_i and its counterpart a'_i are the same operator except for the $\neg pause$ precondition of a'_i.

The non-trivial aspect to prove is the inductive case when there is a *temporal transition*. Starting from two equivalent states in τ and τ', i.e., $\mathsf{H}_S(T_i)$ and s_i, we need to prove that $\mathsf{H}_S(T_{i+1})$, with $T_i = \langle t_i, n_i \rangle$ and $T_{i+1} = \langle t_i + \delta, 0 \rangle$, is equivalent to $s_{i+1} = \gamma(s_i, wait)$. First, we discuss the applicability of the sequence $wait$. The first action $start$ has $\neg pause$ as a precondition, so, since the inductive hypothesis holds, it is applicable in s_i making $pause$ true; after $start$ has been applied, the only executable operators are those belonging to A_P. Then, any sequencing of A_P generates a state in which $\bigwedge_{done \in D} done$ holds, thus making end applicable, too.

Now we proceed to show that the two compared states are equivalent under variables $F \cup X$. Note that each operator in A_P only modifies the X variables. Thus both variables in X^{cp} and in F persist during and after the application of any sequencing of A_P. The only thing left to prove is to show the equivalence of the states w.r.t. variables from X.

Recall that the $start$ operator makes a copy of all the X variables in the corresponding duplicates in X^{cp}, therefore the set of active processes does not change starting from s_i. Indeed, all compiled processes' preconditions only involve variables from X^{cp}.

Each operator $a = \langle pause \wedge \neg done, \{c \triangleright e, done\}\rangle \in A_P$ has a conditional effect that refers to a process ρ, i.e., $c \triangleright e$ with $c = \sigma(pre(\rho), X^{cp})$ and $e = \langle inc, x, \Delta(\delta, \sigma(\xi, X^{cp})\rangle$, which increases x if a formula defined over X^{cp}, i.e., $\sigma(pre(\rho), X^{cp})$, holds in the state where a is applied. Note moreover that the activation of the conditional effects is independent of the order in which actions from A_P are evaluated. In fact, the preconditions of the conditional effects of A_P are expressed in X^{cp} variables which, as noticed above, remain constant throughout the execution of A_P.

To see why $\mathsf{H}_S(T_{i+1})$ and s_{i+1} are equivalent, note that, given a variable $x \in X$, the effects on x is given by a subset of A_P, i.e., $A_P(x)$, defined as the following:

$$A_P(x) = \bigcup_{\substack{ne:\langle x', \xi\rangle \in e(\rho) \\ x'=x,\ \rho \in P}} \{\langle pause \wedge \neg done_{ne}, \{\sigma(pre(\rho), X^{cp}) \triangleright \{\langle inc, x', \Delta(\sigma(\xi, X^{cp}), \delta)) \rangle\}\} \cup \{done_{ne}\}\rangle\}$$

which can be redefined in $A_P(x, s_i)$ by substituting the right-hand side of the numeric effects with all the variables from s_i:

$$A_P(x, s_i) = \bigcup_{\substack{ne:\langle x', \xi\rangle \in e(\rho) \\ x'=x,\ \rho \in P}} \{\langle pause \wedge \neg done_{ne}, \{\sigma(pre(\rho), X^{cp}) \triangleright \{\langle inc, x', s_i[\Delta(\xi, \delta)] \rangle\}\} \cup \{done_{ne}\}\rangle\}$$

Now, observe that the conditional effects modifying the value of x during *wait* are those created from process ρ where $s_i \models pre(\rho)$, i.e., those processes belonging to the context $C(s_i)$. So we can express $s_{i+1}[x]$ as the following summation:

$$s_{i+1}[x] = \gamma(s_i, wait)[x] = s_i[x] + \sum_{\substack{\langle x', \xi\rangle \in e(\rho);\, x'=x \\ \rho \in C(s_i)}} s_i[\Delta(\xi, \delta)] \tag{3}$$

According to Rule 4 of Definition 16, each numeric variable $x \in X$ changes over δ by the active processes in $C(\mathcal{H}_s(T_i)) = \{\rho \in P,\ \mathcal{H}_s(T_i) \models pre(\rho)\}$:

$$\mathcal{H}_s(T_{i+1})[x] = \mathcal{H}_s(T_i)[x] + \sum_{\substack{\langle x', \xi\rangle \in e(\rho);\, x'=x \\ \rho \in C(\mathcal{H}_s(T_i))}} \mathcal{H}_s(T_i)[\Delta(\xi, \delta))] \tag{4}$$

.

Since $\mathcal{H}_s(T_i)$ is equivalent to s_i, by the inductive hypothesis, then the induced contexts are the same, i.e., $C(s_i) = C(\mathcal{H}_s(T_i))$. It follows that the right-hand side expressions of Formulae 3-4 are equivalent. Thus, $\mathcal{H}_s(T_{i+1})$ and $s_{i+1} = \gamma(s_i, wait)$ are equivalent.

Therefore, by induction, $s_i \equiv \mathcal{H}_s(T_i)$ for all i.

($\Leftarrow$) Note that every plan solving Π_{POLY} is structured alternating an agent's action and a sequence (possibly empty) of *wait* sequences. Let $\pi_{\text{POLY}} = \langle wait^{\frac{t_0}{\delta}}, a'_0, wait^{\frac{t_1-t_0}{\delta}}, ..., a'_{n-1}, wait^{\frac{t_e-t_{n-1}}{\delta}}\rangle$ be such a plan. The mapping from π_{POLY} to $\pi_t = \langle\pi, \langle 0, t_e\rangle\rangle$ is similar to what was done for Lemma 1. All the occurrences of the sequence *wait* in π_{POLY} have to be ignored in building π. Starting from π_{POLY} we can build a valid PDDL+ plan $\pi_t = \langle\pi, \langle 0, t_e\rangle\rangle$ as follows: (i) for each action a'_i in π_{POLY} such that $a'_i \notin \{start, end\} \cup A_P$, then $\langle a_i, t_i\rangle$ is in π, where t_i is equal to δ multiplied for the occurrences of *start* in π_{POLY} before a'_i; (ii) for each a'_i, a'_j such that $a'_i \prec a'_j$ in π_{POLY} then $\langle a_i, t_i\rangle \prec \langle a_j, t_j\rangle$ holds in π and (iii) t_e is equal to δ multiplied by the number of *start* in π_{POLY}. The plan we get by using this transformation is:

$$\pi_t = \langle\langle\pi = \langle a_0, t_0\rangle, \langle a_1, t_1\rangle, ..., \langle a_{n-1}, t_{n-1}\rangle\rangle, \langle 0, t_e\rangle\rangle$$

In order to show the validity of π_t, we reason on the discrete projection $\mathcal{H}_{\pi}$ of π_t that is determined as shown in Lemma 1 by using Definition 13. That gives us that the number of STPs of $\mathcal{H}_{\pi}$ equates the number of *wait* sequence plus the number of non simulating actions of π_{POLY}, i.e., $|T_{\mathcal{H}}| = |\pi| + \frac{t_e}{\delta}$, where $|\pi|$ is the number of non simulating action in π_{POLY} and $\frac{t_e}{\delta}$ is the number of *wait* sequence in π_{POLY} and then $|T_{\mathcal{H}}| = |\pi_{\text{POLY}}|$. It is easy to see that, given that π_{POLY} is valid, and proceeding by reasoning by induction over the induced sequences of states τ and τ' as done for the opposite direction, π_t can be proved valid for the problem Π under δ discretisation.

□

3.3 Handling Events

An event in PDDL+ models can be triggered at any time during the execution of a plan, and it is necessary to track whether and how such an event changes the state. More importantly, the semantics of PDDL+ prescribes that a cascade of events may also occur.

In order to handle this behaviour, we devise a new action, namely *SIMEV*, that is responsible for keeping track of the arisen events, and their impacts on the state. This action does so by encoding in one single unit the potentially repeated check and execution of several events via a sophisticated usage of conditional effects that are evaluated in rounds. *SIMEV* makes use of 4 sets of conditional effects:

1. W_{trig} that is responsible for actually updating the state with all events having their precondition satisfied;
2. W_{fired} that is responsible for keeping track of whether there is some event triggered;
3. W_{satu} that is responsible for capturing whether no event is triggered in the previous round;
4. $W_{\bot}$ that captures the situation where there is some inconsistency caused by either a set of mutex events active at the same time or a cyclic sequence of events being triggered.

The formalisation of such conditional effects in PDDL2.1 makes use of a number of additional fresh predicates that are accumulated in set F_E. We have a fact *sim-ev* that signals the beginning of the event simulation; then we have a fact $fired_\varepsilon$ for each event in $\varepsilon \in E$.

The *SIMEV* action is formalised in such a way that:

$$\begin{aligned} pre(SIMEV) &= sim\text{-}ev \\ eff(SIMEV) &= W_{trig} \cup W_{fired} \cup W_{satu} \cup W_{\bot} \end{aligned}$$

where:

$$\begin{aligned}
W_{trig} &= \bigcup_{\substack{c \triangleright e \in eff(\varepsilon) \\ \varepsilon \in E}} \{pre(\varepsilon) \wedge c \triangleright e\} \\
W_{fired} &= \bigcup_{\varepsilon \in E} \{pre(\varepsilon) \triangleright fired_\varepsilon\} \\
W_{satu} &= \Big\{ \bigwedge_{\varepsilon \in E} (\neg pre(\varepsilon) \vee fired_\varepsilon) \triangleright \{\neg sim\text{-}ev\} \cup \bigcup_{\varepsilon \in E} \{\neg fired_\varepsilon\} \Big\} \\
W_{\bot} &= \Big\{ \Big(\bigvee_{\substack{\varepsilon, \varepsilon' \in E: \\ \varepsilon \neq \varepsilon' \wedge \\ mutex(\varepsilon, \varepsilon')}} pre(\varepsilon) \wedge pre(\varepsilon') \Big) \vee \bigvee_{\varepsilon \in E} pre(\varepsilon) \wedge fired_\varepsilon \triangleright \{undefined\} \Big\}
\end{aligned}$$

As it is possible to observe from the snippet above, *SIMEV* captures which events have been triggered and, on the one hand, applies their effects, and on the other hand, memorises whether at least one event has been executed. If that is the case, the action needs to re-evaluate the conditional effects; indeed, a cascade of events can be triggered. It is easy to see that the triggering of events is blocked whenever the action detects a cycle, i.e., an event that is deemed to be executed more than once. For this reason, *SIMEV* can be performed up to $|E|$ times, that is, after the termination condition induced by W_{satu} is reached. Observe that effect ($\{\neg\textit{sim-ev}\} \cup \bigcup_{\varepsilon \in E}\{\neg \textit{fired}_{\varepsilon}\}$) not only interrupts the execution of the simulation of the events but also resets all *fired* facts; this way, we keep the memory ready for the next round of simulation of events.

The very last set of conditional effects ensures that the reached state does not contain cycles or mutexes events that can be executed at the same time. If either of these two situations arises, *SIMEV* generates an inconsistent state, thereby denoted by the special state *undefined*. This check guarantees to prune states where interfering events leading to non-deterministic outcomes or an infinite cascade of events arise. This complies with the semantics restrictions imposed by previous works in PDDL+ (Shin & Davis, 2005; Fox & Long, 2006; Fox et al., 2005).

Note that if the PDDL+ problem to be discretised satisfies the required assumptions about the events, that is: (i) events have to deactivate themselves, (ii) events have to be fired at most once for a given time and (iii) the PDDL+ task is event-deterministic; then $W_{\bot}$ is not necessary. Moreover, in some cases, $W_{\bot}$ may be redundant; suppose to have a state where only a single event ε is triggered, and this does not self-deactivate. In such a case *SIMEV* would have two conflicting conditional effects, i.e., W_{trig} and W_{satu}, which make $\textit{fired}_{\varepsilon}$ true and false at the same time. According to the state transition function, this leads to an undefined state and so $W_{\bot}$ is redundant. We have however included $W_{\bot}$ in the conditional effects of *SIMEV*, making the failure to generate a successor state more explicit. This choice makes the encoding more robust w.r.t. violations of the PDDL+ assumptions.

The presented exponential and polynomial translations can be extended to address PDDL+ planning problems with events. This is done by enforcing the action *SIMEV* to be applied in the initial state and switching back to event simulating modality (*sim-ev*) after any occurrence of an instantaneous action (in both translations), and after the execution of the action *SIM* and *end* in EXP and POLY, respectively. Let a be an action, we denote with $\textit{ev-check}(a)$ the new action a' such that: $\textit{pre}(a') = \textit{pre}(a) \wedge \neg\textit{sim-ev}$ and $\textit{eff}(a') = \textit{eff}(a) \cup \{\textit{sim-ev}\}$.

We are now ready to summarise the further translation that is needed in order to make the resulting PDDL2.1 formulation aware of the presence of events, for both cases.

Event-aware EXP Translation. Let $\Pi = \langle F, X, I, G, A, E, P\rangle$ be a PDDL+ problem, and let $\Pi_{\text{EXP}} = \langle F, X, I, G, A \cup \{SIM\}, c\rangle$ the PDDL2.1 problem obtained using EXP ignoring set E, respectively. The handling of events E can be achieved by a further translation into $\Pi^{events}_{\text{EXP}} = \langle F \cup F_E, X, I \cup \{\textit{sim-ev}\}, G \wedge \neg\textit{sim-ev}, A' \cup \{\textit{ev-check}(SIM)\} \cup \{SIMEV\}, c\rangle$ with $A' = \bigcup_{a \in A} \textit{ev-check}(a)$.

Event-aware POLY Translation. Let $\Pi = \langle F, X, I, G, A, E, P\rangle$ be a PDDL+ problem, and let $\Pi_{\text{POLY}} = \langle F \cup D \cup \{pause\}, X \cup X^{cp}, I, G \wedge \neg pause, A_c \cup A_P \cup \{start, end\}, c\rangle$ the PDDL2.1 problem obtained using POLY ignoring set E, respectively. The handling of events E can be achieved by a further translation into $\Pi^{events}_{\text{POLY}} = \langle F \cup D \cup \{pause\} \cup F_E, X \cup X^{cp}, I \cup \{\textit{sim-ev}\}, G \wedge \neg pause \wedge \neg\textit{sim-ev}, A'_c \cup A_P \cup \{start', \textit{ev-check}(end)\} \cup \{SIMEV\}, c\rangle$ with $A'_c = \bigcup_{a \in A_c} \textit{ev-check}(a)$ and $\textit{pre}(start') = \textit{pre}(start) \wedge \neg\textit{sim-ev}$ and $\textit{eff}(start') = \textit{eff}(start)$.

Example 3.3 (EXP Translation with Events - Continuing on Example 3.1)**.** *In the following we show how the problem obtained using* EXP *in Example 3.1, i.e.,* Π_{EXP}*, can be extended to take also the events modelling cars' collisions into account, i.e.,*

$E = \bigcup_{\substack{car, car' \in cars \\ car \neq car'}} \{crash\text{-}fast_{car,car'}, crash\text{-}slow_{car,car'}\}$. *For the sake of conciseness, event aliases are shortened, too. For example, the* $crash\text{-}fast_{car1,car2}$ *event is shortened to* $f_{1\text{-}2}$. *The* PDDL2.1 *task we get taking events into account is* $\Pi^{events}_{\text{EXP}} = \langle F \cup F_E, X, I \cup \{sim\text{-}ev\}, G \wedge \neg sim\text{-}ev, A' \cup \{ev\text{-}check(SIM), SIMEV\}, c\rangle$ *where:*

$$F_E = \{fired_{f_{1\text{-}2}}, fired_{f_{2\text{-}1}}, fired_{s_{1\text{-}2}}, fired_{s_{2\text{-}1}}\}$$

$$A' = \bigcup_{car \in \{car1, car2\}} \{ev\text{-}check(accelerate_{car}), ev\text{-}check(decelerate_{car}), ev\text{-}check(overtake_{car}), ev\text{-}check(return_{car}), ev\text{-}check(start_{car}), ev\text{-}check(stop_{car})\}.$$

In the following we show how actions in A *are modified by using function ev-check( ) over cars:*

$$\begin{aligned}
ev\text{-}check(accelerate_{car}) &= \langle eng\text{-}on_{car} \wedge \langle a_{car} < A_{max}\rangle \wedge \neg sim\text{-}ev, \{\langle increase, a_{car}, 1\rangle, sim\text{-}ev\}\rangle\\
ev\text{-}check(decelerate_{car}) &= \langle eng\text{-}on_{car} \wedge \langle a_{car} > A_{min}\rangle \wedge \neg sim\text{-}ev, \{\langle decrease, a_{car}, 1\rangle, sim\text{-}ev\}\rangle\\
ev\text{-}check(overtake_{car}) &= \langle eng\text{-}on_{car} \wedge \langle v_{car} > 0\rangle \wedge \neg ot_{car} \wedge \neg sim\text{-}ev, \{ot_{car}, sim\text{-}ev\}\rangle\\
ev\text{-}check(return_{car}) &= \langle eng\text{-}on_{car} \wedge \langle v_{car} > 0\rangle \wedge ot_{car} \wedge \neg sim\text{-}ev, \{\neg ot_{car}, sim\text{-}ev\}\rangle\\
ev\text{-}check(start_{car}) &= \langle \neg eng\text{-}on_{car} \wedge \neg sim\text{-}ev, \{eng\text{-}on_{car}, sim\text{-}ev\}\rangle\\
ev\text{-}check(stop_{car}) &= \langle eng\text{-}on \wedge \langle v_{car} = 0\rangle \wedge \langle a_{car} = 0\rangle \wedge \neg sim\text{-}ev, \{\neg eng\text{-}on_{car}, sim\text{-}ev\}\rangle.
\end{aligned}$$

The same for SIM:

$$\begin{aligned}
pre(ev\text{-}check(SIM)) &= \neg sim\text{-}ev\\
eff(ev\text{-}check(SIM)) &= \{W_{\{m1\}}, W_{\{m2\}}, W_{\{m1,m2\}}, sim\text{-}ev\}.
\end{aligned}$$

In both cases, the ev-check ( ) function is used to make sure that whenever an $a \in A$ *or SIM is performed then event testing has to necessarily be performed by executing at least one SIMEV.*

Finally, the operator SIMEV is defined as follows:

$$\begin{aligned}
pre(SIMEV) &= sim\text{-}ev\\
eff(SIMEV) &= \overbrace{\{W^{f_{1\text{-}2}}_{trig}, W^{f_{2\text{-}1}}_{trig}, W^{s_{1\text{-}2}}_{trig}, W^{s_{2\text{-}1}}_{trig}\}}^{W_{trig}} \cup \overbrace{\{W^{f_{1\text{-}2}}_{red}, W^{f_{2\text{-}1}}_{red}, W^{s_{1\text{-}2}}_{red}, W^{s_{2\text{-}1}}_{red}\}}^{W_{red}} \cup W_{satu} \cup W_{?}
\end{aligned}$$

where the conditional effects are defined as follows:

$$
\begin{aligned}
W^{f_{1\text{-}2}}_{trig} &= \langle d_{car1} - d_{car2} < D_T \rangle \wedge \langle d_{car1} - d_{car2} \geq 0 \rangle \wedge ot_{car1} \wedge ot_{car2} \wedge \neg crashed \rhd \{crashed\}\\
W^{f_{2\text{-}1}}_{trig}, &= \langle d_{car2} - d_{car1} < D_T \rangle \wedge \langle d_{car2} - d_{car1} \geq 0 \rangle \wedge ot_{car2} \wedge ot_{car1} \wedge \neg crashed \rhd \{crashed\}\\
W^{s_{1\text{-}2}}_{trig} &= \langle d_{car1} - d_{car2} < D_T \rangle \wedge \langle d_{car1} - d_{car2} \geq 0 \rangle \wedge \neg ot_{car1} \wedge \neg ot_{car2} \wedge \neg crashed \rhd \{crashed\}\\
W^{s_{2\text{-}1}}_{trig} &= \langle d_{car2} - d_{car1} < D_T \rangle \wedge \langle d_{car2} - d_{car1} \geq 0 \rangle \wedge \neg ot_{car2} \wedge \neg ot_{car1} \wedge \neg crashed \rhd \{crashed\}\\
W^{f_{1\text{-}2}}_{red} &= \langle d_{car1} - d_{car2} < D_T \rangle \wedge \langle d_{car1} - d_{car2} \geq 0 \rangle \wedge ot_{car1} \wedge ot_{car2} \wedge \neg crashed \rhd \{fired_{f_{1\text{-}2}}\}\\
W^{f_{2\text{-}1}}_{red} &= \langle d_{car2} - d_{car1} < D_T \rangle \wedge \langle d_{car2} - d_{car1} \geq 0 \rangle \wedge ot_{car2} \wedge ot_{car1} \wedge \neg crashed \rhd \{fired_{f_{2\text{-}1}}\}\\
W^{s_{1\text{-}2}}_{red} &= \langle d_{car1} - d_{car2} < D_T \rangle \wedge \langle d_{car1} - d_{car2} \geq 0 \rangle \wedge \neg ot_{car1} \wedge \neg ot_{car2} \wedge \neg crashed \rhd \{fired_{s_{1\text{-}2}}\}\\
W^{s_{2\text{-}1}}_{red} &= \langle d_{car2} - d_{car1} < D_T \rangle \wedge \langle d_{car2} - d_{car1} \geq 0 \rangle \wedge \neg ot_{car2} \wedge \neg ot_{car1} \wedge \neg crashed \rhd \{fired_{s_{2\text{-}1}}\}\\
W_{satu} &= (pre(f_{1\text{-}2}) \vee fired_{f_{1\text{-}2}}) \wedge (pre(f_{2\text{-}1}) \vee fired_{f_{2\text{-}1}}) \wedge (pre(s_{1\text{-}2}) \vee fired_{s_{1\text{-}2}}) \wedge\\
&\quad (pre(s_{2\text{-}1}) \vee fired_{s_{2\text{-}1}}) \rhd \{\neg sim\text{-}ev, \neg fired_{f_{1\text{-}2}}, \neg fired_{f_{2\text{-}1}}, \neg fired_{s_{1\text{-}2}}, \neg fired_{f_{2\text{-}1}}\}.
\end{aligned}
$$

3.4 Properties

In the following theorem, we prove the soundness and completeness for POLY and EXP translations in the general case for PDDL+ problems with events by using Lemmas 1 and 2.

Theorem 1 (Soundness and Completeness of POLY and EXP)**.** *Let* $\Pi = \langle F, X, I, G, A, E, P \rangle$ *be a* PDDL+ *planning instance, and let* $\Pi^{events}_{\text{EXP}}$ $(\Pi^{events}_{\text{POLY}})$ *be the* PDDL2.1 *planning instance obtained by using the* EXP (POLY) *translation.* Π *admits a solution under* δ *discretisation iff so does* $\Pi^{events}_{\text{EXP}}$ $(\Pi^{events}_{\text{POLY}})$.

Proof. We focus on $\Pi^{events}_{\text{EXP}}$ (the proof for $\Pi^{events}_{\text{POLY}}$ is similar) and prove the two directions largely exploiting the constructions devised for the event-free translations.

($\Rightarrow$) Let $ES(T_i) = \langle E_{trigg}(T_i), ..., E_{trigg}(T_{i+k-1}) \rangle$ be the unique and finite[7] cascade of events from an STP T_i of the discrete projection H of π_t solving Π.

We show that it is possible to construct a plan π_{EXP} from π_t such that π_{EXP} is valid for Π_{EXP}. Differently from an event-free planning task where we can define a mapping from π_t to π_{EXP}, in this case we have to resort to H to build a valid π_{EXP}. Let $T_{\mathsf{H}} = \langle T_0, ..., T_m \rangle$ be the $m+1$ STPs of H, we define π_{EXP} in two steps. As a first step, we define the sequence $\pi^{0}_{\text{EXP}} = \langle a^0_0, ..., a^0_{m-1} \rangle$ such that for all $i \in [0..m-1]$

$$
a^0_i = \begin{cases} a_i & \text{if } \mathsf{H}_A(T_i) = \langle a_i \rangle \neq \langle\rangle \\ SIMEV & \text{if } E_{trigg}(T_i) \neq \langle\rangle \\ SIM & \text{otherwise} \end{cases}
$$

Then we obtain π_{EXP} from π^0_{EXP} by inserting a $SIMEV$ action just before any action $a \in \pi^0_{\text{EXP}}$ such that $a \neq SIMEV$, and just before the end of the plan. Let $\tau^0 = \langle s_0, ..., s_m \rangle$

7. Note that, under the restriction imposed over PDDL+, for each $E, E' \in set(ES)$ with $E \neq E'$, $set(E) \cap set(E') = \emptyset$ and, for each $\varepsilon, \varepsilon' \in set(E)$ with $E \in set(ES)$, ε and ε' are not mutex, as long as $\varepsilon \neq \varepsilon'$.

be the sequence of states obtained by applying iteratively actions from π_{EXP} and filtering out those states produced by any last *SIMEV* of a series, the difficult bit is to show that τ' is equivalent to $\tau = \langle \mathsf{H}_s(T_0), \ldots, \mathsf{H}_s(T_m) \rangle$ under variables $F \cup X$. The first observation is that $|\tau| = |\tau'|$; and this follows directly from the fact that the number of states we are filtering out is exactly the number of *SIMEV* that we have added to the plan. Then, in order to prove that $\mathsf{H}_s(T_i)$ and s_i are equivalent for all $i \in [0..m]$, we can use the same arguments of Lemma 1 extended to account for the case where the transition is due to a cascade of events. More precisely, let $T_i = \langle t_i, n_i \rangle$ be a STP in which a cascade of events $ES(T_i)$ with $|ES(T_i)| = k$ is triggered. By using R1 of Definitions 13-16 we know that, for each $j \in [i..i+k]$ $\mathsf{H}_s(T_j) \models \bigwedge_{\varepsilon \in E_{trigg}(T_j)} pre(\varepsilon)$, $\mathsf{H}_s(T_{j+1}) = \gamma(\mathsf{H}_s(T_j), E_{trigg}(T_j))$ and $T_{j+1} = \langle t_i, n_j + 1 \rangle$.

In order to prove $\mathsf{H}_s(T_{i+k}) = s_{i+k}$ with $s_{i+k} = \gamma(s_i, SIMEV^k)$ we need to show that, for every j such that $j \in [i..i+k]$, $\mathsf{H}_s(T_j) = s_j$; we do so, again, by induction. The base case ($j = i$) is trivially proved (inherited by Lemma 2). For the inductive step it suffices to observe that *SIMEV* exhibits a behaviour that is equivalent to $\langle \top, \bigcup_{\substack{c \rhd e \in e(\varepsilon) \\ \text{with } \varepsilon \in E_{trigg}(T_j)}} c \rhd e \rangle$ and then it follows that $\mathsf{H}_s(T_{j+1})$ and $s_{j+1} = \gamma(s_j, SIMEV)$ are equivalent. The only thing that is missing is to show that when each action is applied in π_{EXP} the variable *sim-ev* is false. But this directly follows from the fact that the last *SIMEV* is applied when all events have been triggered. Indeed, we have that $E_{trigg}(T_{i+k}) = \langle \rangle$, and $\mathsf{H}_s(T_{i+k}) = s_{i+k}$. So *SIMEV* will make *sim-ev* false and get ready for the next round of execution by resetting all the monitoring variables *fired* to false.

($\Leftarrow$) The mapping from π_{EXP} to π_t is identical to what was done for Lemma 1, but for the fact that all occurrences of *SIMEV* are ignored. Then, the π_t plan can be proved valid against the discretised PDDL+ model by observing that, for each series of *SIMEV* of length k the projection H encompasses $k - 1$ STPs, one for each *SIMEV* that triggers a change on variables in $F \cup X$. Each STP generates a set of events whose effects are those that arise from the active conditional effects of the associated *SIMEV*. This is due to the fact that the *SIMEV* conditional effect's condition subsumes the precondition of each event associated with it.

□

In the following theorems, we show how the translations EXP and POLY impact the size of the compiled problems Π_{EXP} and Π_{POLY}.

Theorem 2 (Size of Π_{EXP}). *Let $\Pi = \langle F, X, I, G, A, \emptyset, P \rangle$ be a PDDL+ planning instance the reformulation POLY produces a PDDL2.1 planning instance $\Pi_{\text{EXP}} = \langle F, X, I, G, A \cup \{SIM\}, c \rangle$ that increases the size of Π exponentially.*

Proof. In the compiled problem Π_{EXP} the set of actions is extended with the *SIM* action which has a number of conditional effects equal to $|\mathcal{P}^+(P)| = |\mathcal{P}(P) \setminus \emptyset| = 2^{P} - 1$. □

Theorem 3 (Size of Π_{POLY}). *Let $\Pi = \langle F, X, I, G, A, \emptyset, P \rangle$ be a PDDL+ planning instance the reformulation EXP produces a PDDL2.1 planning instance $\Pi_{\text{POLY}} = \langle F \cup D \cup \{pause\}, X \cup X^{cp}, I, G \cup \{\neg pause\}, A_c \cup A_P \cup \{start, end\}, c \rangle$ that increases the size of Π only polynomially.*

Proof. The propositional variables increase by $|D| + 1$ where, in the worst case, $|D| = |P| \cdot |X|$. Numeric variables are doubled; indeed $|X^{cp}| = |X|$. Finally, since $|A_c| = |A|$, the actions increase by $|A_P| + 2$ therefore, in the worst case, $|A_P| = |P| \cdot |X|$. □

In his study, Nebel (2000) considered the effects of the translations between planning formalisms on the size of the plans solving the reformulated instances, besides the needed temporal and spatial resources.

In our context, given a PDDL+ problem Π and a reformulation $Z \in \{\text{POLY}, \text{EXP}\}$, we say that Z *preserves the plan size exactly*, up to additive constants, iff for each plan $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ which solves Π there exists a PDDL2.1 plan π_Z such that $|\pi_Z| \leq |\pi| + k$ where $k \in \mathbb{N}_0$; Z *preserves the plan linearly* iff for each plan $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ which solves Π there exists a PDDL2.1 plan π_Z such that $|\pi_Z| \leq c \cdot |\pi| + k$ where $c, k \in \mathbb{N}_0$; finally, Z *preserves the plan polynomially* iff for each plan $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ which solves Π there exists a PDDL2.1 plan π_Z such that $|\pi_Z| \leq p(|\Pi|, |\pi|)$ where $p(\cdot)$ is a polynomial expression that depends on the size of Π, i.e., $|\Pi|$, and on the size of π, i.e., $|\pi|$.

Theorem 4 (Size of Plans for Π_{EXP} and Π_{POLY})**.** *Let $\Pi = \langle F, X, I, G, A, \emptyset, P \rangle$ be an event-free* PDDL+ *planning instance, and let Π_{POLY} and Π_{EXP} be the* PDDL2.1 *obtained by using the* POLY *and* EXP *translations, respectively. The reformulations* POLY *and* EXP *preserve the plan size polynomially and linearly, respectively.*

Proof. Let $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ be a solution for Π under δ discretisation and let π_{POLY} and π_{EXP} be the corresponding plan for Π_{POLY} and Π_{EXP}, respectively. Using the rules provided in Lemmas 1-2 for mapping π_t into π_{EXP} and π_{POLY} respectively, we get that for each discrete advance of time in the original plan, i.e., $\frac{t_e - t_s}{\delta}$, we have to execute the action *SIM* in π_{EXP} and the sequence of actions *wait* in π_{POLY}. Then, for EXP we obtain:

$$|\pi_{\text{EXP}}| \leq |\pi| + \frac{t_e - t_s}{\delta}$$

Each *wait* in π_{POLY} consists of two delimiting actions, i.e., *start* and *end*, plus an action for each numeric effect of each process; in the worst case, each process has a numeric effect for each variable of the problem. By adding these contributions we obtain:

$$|\pi_{\text{POLY}}| \leq |\pi| + \frac{t_e - t_s}{\delta} \cdot (|P| \cdot |X| + 2)$$

□

Theorem 5 (Size of Plans for $\Pi_{\text{EXP}}^{events}$ and $\Pi_{\text{POLY}}^{events}$)**.** *Let $\Pi = \langle F, X, I, G, A, E, P \rangle$ be a* PDDL+ *planning instance, and let $\Pi_{\text{EXP}}^{events}$ and $\Pi_{\text{POLY}}^{events}$ be the* PDDL2.1 *obtained by using the* POLY *and* EXP *translations, respectively. The reformulation* POLY *and* EXP *preserves plan size polynomially.*

Proof. Let $\pi_t = \langle \pi, \langle t_s, t_e \rangle \rangle$ be a solution for Π under δ discretisation and let π_{POLY} and π_{EXP} be the corresponding plan for $\Pi_{\text{POLY}}^{events}$ and $\Pi_{\text{EXP}}^{events}$, respectively.

Using the rules outlined in Lemma 2 and Theorem 1 to map π_t into π_{POLY} we can prove the upper-bound on $|\pi_{\text{POLY}}|$ as follows: (i) each action of π_{POLY} has to be followed by at least one *SIMEV* action up to a maximum of $|E|+1$ and such sequence has to be executed also at the beginning of π_{POLY} since $I' \models$ *sim-ev*; (ii) possible event triggers must be checked after each block of actions simulating the passage of time, i.e., the *wait* sequence; therefore the term $|E|+1$ has to be multiplied by the number of time steps occurred within the envelope $\langle t_s, t_e \rangle$, i.e., $\frac{t_e - t_s}{\delta}$; (iii) each *wait* in π_{POLY} consists of two delimiting actions, i.e., *start* and *end*, plus an action for each numeric effect of each process; in the worst case, each process has a numeric effect for each variable of the problem. By adding these contributions we obtain:

$$|\pi_{\text{POLY}}| \leq (|\pi|+1) \cdot \overbrace{(|E|+1)}^{\text{(i)}} + \frac{t_e - t_s}{\delta} \cdot (\overbrace{(|E|+1)}^{\text{(ii)}} + \overbrace{(|P| \cdot |X|+2)}^{\text{(iii)}})$$

The only difference for π_{EXP} is that instead of the *wait* sequence for the contribute iii) we apply the single action *SIM*; then, we obtain:

$$|\pi_{\text{EXP}}| \leq (|\pi|+1) \cdot \overbrace{(|E|+1)}^{\text{(i)}} + \frac{t_e - t_s}{\delta} \cdot (\overbrace{(|E|+1)}^{\text{(ii)}} + \overbrace{1}^{\text{(iii)}})$$

□

4. Optimising POLY and EXP

This section presents two optimisations for our translation schemata, both aimed at reducing the length of valid plans needed to solve the translated problem. Our optimisations preserve both the soundness and the completeness of the approach; the basic idea is to exploit conditions obtained by looking at the structure of our problem. Such conditions intercept whether some action or an event does not trigger any other event. Our first optimisation allows us to prune the *SIMEV* action in case an action cannot be followed by some event. Thanks to this optimisation our translated problem can potentially chain a sequence of actions before synchronising their effects with exogenous events. The second optimisation is aimed at simplifying the *SIMEV* definition through an analysis of the structure of all events. Both optimisations are constructed by introducing the notion of Trigger-Free action (or event), which is under-approximated, by intercepting a sufficient condition obtained through reasoning by regression.

4.1 Avoiding Events Checking

In Section 3.3 we have seen that each action is modified in order to ensure that after its execution, a *SIMEV* sequence is forced to be executed. This is obtained by using the function *ev-check*$(\cdot)$ that takes as input an action a, and returns another action a' where $pre(a') = pre(a) \wedge \neg\,$*sim-ev* and $eff(a') = eff(a) \cup \{$*sim-ev*$\}$.

The rationale behind the first optimisation we present consists of studying conditions under which an action does not need to be followed by the potentially expensive *SIMEV* sequence. To do so, we introduce the notions of *Trigger-Free* and *Universally Trigger-Free* actions.

Definition 18 (Trigger-Free and Universally Trigger-Free action)**.** *Given a* PDDL+ *problem* Π*, let* $a \in A$ *and let* $\varepsilon \in E$*, we say that* a *is* Trigger-Free w.r.t. ε*, denoted with* $TF(a, \varepsilon)$*, iff for each state* s *such that for all* $\varepsilon' \in E$*,* $s \not\models pre(\varepsilon')$*, then* $\gamma(s, a) \models \neg pre(\varepsilon)$*. Moreover,* *we say that a is* Universally Trigger-Free*, denoted with $UTF(a)$, iff a is Trigger-Free w.r.t. all events in E, i.e., $UTF(a) \Leftrightarrow \forall \varepsilon \in E,\ TF(a, \varepsilon)$.*

Intuitively, this notion captures all those actions that do not change the state in a way that some event can be triggered. The UTF definition can be used to slightly modify either POLY or EXP to disallow the application of function *ev-check*( ) to all UTF actions. For example, for the event-aware variant of EXP, the definition of $A' = \bigcup_{a \in A} \textit{ev-check}(a)$ is replaced with:

$$A' = \{\textit{ev-check}(a) \mid a \in A \wedge \neg UTF(a)\} \cup \{a \mid a \in A, UTF(a)\}$$

The optimised variant of the event-aware variant of POLY is straightforward.

It is worth noting that, considering our very expressive language for modelling actions, the computation of an exact Trigger-Free relation is complicated. Firstly, because we have numeric effects and propositional effects. Secondly, because any such effect can depend on the state in which the action is applied. Thirdly, because we allow the combination of any NNF formula in the precondition of the events. For these reasons, in what follows we propose an under-approximation of such a relation.

We under-approximate the Trigger-Free relationship through Algorithm 1, which works by sequencing two checks. The first check (Line 2) is a neutrality test that evaluates whether a does not affect anything that is involved in the preconditions of ε. If that is the case, then there is no way ε can be triggered. This check is done by inspecting if the action affects any variable in some necessary condition for the precondition of ε; formally $vars(\textit{eff}(a)) \cap vars(necessary(pre(\varepsilon)) \neq \emptyset$, where:

- given a precondition expressed as a formula φ, the function $necessary(\varphi)$ returns those terms that have to necessarily be true in order to ensure that φ is true. We restrict the attention to top-level conjuncts. For example, consider the case in which $\varphi = p \wedge \neg q \wedge \langle x > 10\rangle \wedge (c \vee (b \wedge \langle y + z < 10\rangle)$, the necessary conditions are that p and $\neg q$ hold true, and x is greater than 10, i.e., $necessary(\varphi) = \{p, \neg q, \langle x > 10\rangle\}$;

- $vars(\,)$ is a function that returns the set of all variables, whether propositional or numeric, involved in its parameter; to be specific, if the parameter is a formula φ then $vars(\varphi)$ returns all the variables exploring recursively the formula, e.g., given $\varphi = p \wedge \langle x > 10\rangle$, then $vars(\varphi) = \{p, x\}$; if the parameter is an effect e then $vars(e)$ returns all the affected variables, e.g., if $e = \{\top \triangleright \{p\}, \top \triangleright \{\langle inc, x, 20 + y\rangle\}\}$, then $vars(e) = \{p, x\}$.

The second check is performed by using a sufficient condition that checks whether action a always makes the preconditions of ε unsatisfied (Lines 4-6). The procedure seeks if there exists a necessary conjunct g of $pre(\varepsilon)$ such that the logical conjunction of the formula resulting by applying operator R (see below) on g and some necessary preconditions of the action generates a contradiction (Line 5). For example, if there is an operator R that results in $\langle x > 0\rangle$, and the action has one precondition which states that $\langle x > 0\rangle$, then the conjunction $\langle x > 0\rangle \wedge \langle x < 0\rangle$ is not satisfiable and therefore the procedure returns TRUE.[8]. The operator R is similar to a regression function; it takes as an input an action and a

8. For this check we rely on an external solver; in our experiment, we used SimPy (Meurer et al., 2017).

condition (propositional or numeric condition) and returns a formula, i.e., $R : A \times \Lambda \rightarrow \Lambda$, where Λ is the set of conditions (propositional and numeric) that can be expressed in PDDL+. It is formally defined as follows:

$$R(a,g) = \begin{cases} g & \text{if } vars(g) \cap vars(\mathit{eff}(a)) = \emptyset \\ \top & \text{if } g = \langle f = b \rangle \text{ and } \exists\, c \triangleright \{..., \langle f := b \rangle\} \in \mathit{eff}(a) \\ \bot & \text{if } g = \langle f = b \rangle \text{ and } \exists\, \top \triangleright \{..., \langle f := b' \rangle\} \in \mathit{eff}(a) \text{ and } b \neq b' \\ NR(a,g) & \text{if } g = \langle \xi \bowtie 0 \rangle \text{ and } direct(a,g) \\ \top & \text{if } g = \langle \xi \bowtie 0 \rangle \text{ and not } direct(a,g) \end{cases} \tag{5}$$

In the formula, the function *direct* returns true for all actions whose numeric effects on g are unconditionally triggered, i.e., the left-hand side of all numeric conditional effects onto g is $\top$. Formally: $direct(a,g) \triangleq \forall\, c \triangleright e \in \mathit{eff}(a) \wedge g \in e,\ c = \top$.

NR is the effect regressor as for Scala, Haslum, Thiébaux, and Ramírez (2020), slightly reformulated to consider conditional effects as an input.[9] More precisely, NR transforms any numeric condition $\xi \bowtie 0$ in $\xi[x_1/\tau(a,x_1), ..., x_k/\tau(a,x_k)] \bowtie 0$ where

$$\tau(a,x_i) = \begin{cases} \xi' & \text{if } \exists\, \top \triangleright \{..., \langle asgn, x_i, \xi' \rangle\} \in \mathit{eff}(a) \\ x_i + \xi' & \text{if } \exists\, \top \triangleright \{..., \langle inc, x_i, \xi' \rangle\} \in \mathit{eff}(a) \\ x_i - \xi' & \text{if } \exists\, \top \triangleright \{..., \langle dec, x_i, \xi' \rangle\} \in \mathit{eff}(a) \\ x_i & \text{Otherwise} \end{cases} \tag{6}$$

and $x_1, ..., x_k$ are the k numeric variables affected by a. The / operator denotes a substitution operator for manipulating numeric conditions, e.g., for $\langle a + b > 0 \rangle$ then $\xi[a/(a+1), b/(b+1)] = \langle (a+1) + (b+1) > 0 \rangle$.

Intuitively, Equation 5 distinguishes whether the given input is a propositional condition or a numeric one, and regresses failure (i.e., $\bot$) when the action makes unsatisfied the condition independently on the state in which the action is applied. In order to be sure that the action *always* makes a condition violated (for instance deletes it), operator R conservatively excludes all cases where the action *may* achieve the condition. In this latter case, the function safely returns $\top$. If the action is however not affecting the condition in any sensible manner, i.e., the first case of the equation, the condition is left unaltered.

Lemma 3. *Let a be an action and ε be an event. If Algorithm 1 returns* TRUE *the relation $TF(a, \varepsilon)$ holds.*

Proof. Algorithm 1 returns TRUE if the condition at Line 2 holds or if at least one necessary condition within the precondition of ε will not hold after the execution a. If the action does not interact with any of the variables in the precondition of ε, then the action is Trigger-Free w.r.t. ε in that if ε is not triggered in some state, it is certainly not a that will make it active. Regarding the second condition, it suffices to observe that if a makes certainly false at least one of its necessary preconditions, or modifies them in a way that when used

9. Note that conditional effects here are considered only syntactically, but are basically ignored. How to extend regression with conditional effects with numeric effects in order to handle them in a sensible manner is out of the scope of this work.

Algorithm 1: Algorithm for under approximating when a is Trigger-Free w.r.t. an event ε

Input: an action or an event a and an event ε
Output: a Boolean value
1 **Function** $TF(a, \varepsilon)$:
2   **if** $vars(\mathit{eff}(a)) \cap vars(necessary(pre(\varepsilon)) = \emptyset$ **then**
3   **return** TRUE
4   **for** $g \in necessary(pre(\varepsilon))$ **do**
5   **if** $R(a, g) \wedge \bigwedge_{g' \in necessary(pre(a))} g' \models \bot$ **then**
6   **return** TRUE
    end
7   **return** FALSE

---

in conjunction with the precondition of a the arising satisfiability problem is unsatisfiable, then there is no way event ε could be triggered after executing a. Therefore, also in this case, returning TRUE implies that action a is Trigger-Free w.r.t. ε. □

Theorem 6. *The optimisation variants of* EXP *and* POLY *using Algorithm 1 to under-approximate the Trigger-Free relation preserves the soundness and the completeness of both translations.*

Proof. The proof for this theorem follows directly from Definition 18. Indeed, we are basically only avoiding the execution of events that can never be applied after a Trigger-Free action, and moreover, by Lemma 3 we know that Algorithm 1 only returns TRUE if the Trigger-Free relation holds. □

In the following example, we detail how Trigger-Free relations are evaluated in practice.

Example 4.1 (Trigger-free Operators)**.** *Let* $\varepsilon = \langle\langle x = 10\rangle \wedge \langle y+z > 20\rangle \wedge \langle w = \top\rangle, \top \triangleright \{\langle a := \bot\rangle\}\rangle$ *be an event and let* a_1, a_2 *and* a_3 *be three actions such that:*

$$
\begin{aligned}
a_1 &= \langle\top, \{\top \triangleright \{\langle a := \top\rangle\}, \top \triangleright \{\langle inc, b, 10\rangle\}\}\rangle\\
a_2 &= \langle\top, \{\top \triangleright \{\langle w := \bot\rangle\}\}\rangle\\
a_3 &= \langle\langle y < 10\rangle, \{\top \triangleright \{\langle inc, y, 10\rangle\}, \top \triangleright \{\langle asgn, z, 0\rangle\}\}\rangle
\end{aligned}
$$

We study if the property $TF(a, \varepsilon)$ *holds for each* $a \in \{a_1, a_2, a_3\}$ *by using step by step Algorithm 1.*

To establish that a_1 *is Trigger-Free w.r.t.* ε*, it suffices the first check of Algorithm 1 (Line 2). Since* $vars(\mathit{eff}(a_1)) = vars(\{\top \triangleright \{\langle a := \top\rangle\}, \top \triangleright \{\langle inc, b, 10\rangle\}\}) = \{a, b\}$ *and* $vars(necessary(pre(\varepsilon))) = vars(pre(\varepsilon)) = vars(\langle x = 10\rangle \wedge \langle y + z > 20\rangle \wedge \langle w = \top\rangle) = \{x, y, w, z\}$*, then* $TF(a_1, \varepsilon)$ *holds. Indeed, the action is neutral w.r.t. the event.*

Concerning a_2*, since* $vars(\mathit{eff}(a_2)) = \{w\}$*, then* $vars(\mathit{eff}(a_2)) \cap vars(pre(\varepsilon)) = \{w\} \cap \{x, y, w, z\} = \{w\} \neq \emptyset$ *and therefore* a_2 *is not neutral w.r.t.* ε*. It is necessary to investigate further what kind of relationship between* a_2 *and* ε *exists. Let* $pre(\varepsilon) = g_1 \wedge g_2 \wedge g_3$ *where* $g_1 = \langle x = 10\rangle$*,* $g_2 = \langle y + z > 20\rangle$ *and* $g_3 = \langle w := \bot\rangle$*, then* $necessary(pre(\varepsilon)) = \{g_1, g_2, g_3\}$*.*

Before proceeding, note that for a_2 we have $\bigwedge_{g' \in necessary(pre(a_2))} g' = pre(a_2) = \top$*. We iterate over the necessary conditions* $\{g_1, g_2, g_3\}$*:*

- $g_1 = \langle x = 10 \rangle$*; since* $vars(g_1) = \{x\}$ *and* $vars(\mathit{eff}(a_2)) = \{w\}$ *then* $vars(g_1) \cap vars(\mathit{eff}(a_2)) = \emptyset$*; therefore we fall into case (i) of Formula 5. Therefore* $R(a_1, g_1) \wedge pre(a_2) = g_1 \wedge \top = \langle x = 10 \rangle \wedge \top \not\models \bot$*;*

- $g_2 = \langle y + z > 20 \rangle$*; analogously to what seen for* g_1*, we get that:* $R(a_2, g_2) \wedge pre(a_2) = \langle y + z > 20 \rangle \wedge \top \not\models \bot$*;*

- $g_3 = \langle w = \top \rangle$*; since* g_3 *is a propositional condition involving* w *and there exists a propositional assignment* $\top \triangleright \{\langle w := \bot \rangle\} \in \mathit{eff}(a_2)$ *such that the two Boolean values are in opposition, we fall into the case (iii) of Formula 5. Therefore* $R(a_2, g_3) \wedge necessary(pre(a_2)) = \bot \wedge \top \models \bot$*;*

We have shown that there is at least one necessary condition of $pre(\varepsilon)$*, i.e.,* g_3*, which is always falsified by* a_2*, therefore we have proved that* $TF(a_2, \varepsilon)$ *holds.*

As far as it is concerned by a_3*, since* $vars(\mathit{eff}(a_3)) = vars(\{\top \triangleright \{\langle inc, y, 10 \rangle\}, \top \triangleright \{\langle asgn, z, 0 \rangle\}\}) = \{y, z\}$*, then* $vars(\mathit{eff}(a_3)) \cap vars(pre(\varepsilon)) = \{y, z\} \neq \emptyset$ *and therefore* a_3 *is not neutral with respect to* ε*. Before proceeding, note that for* a_3 *we have that* $\bigwedge_{g' \in necessary(pre(a_3))} g' = pre(a_3) = \langle y < 10 \rangle$*. Again, we iterate over the necessary conditions of* $pre(\varepsilon)$*:*

- $g_1 = \langle x = 10 \rangle$*; for* g_1*, following the same steps above for* a_2*, we obtain* $R(a_3, g_1) \wedge pre(a_3) = \langle x = 10 \rangle \wedge \langle y < 10 \rangle \not\models \bot$*;*

- $g_2 = \langle y + z > 20 \rangle$*; since* $vars(g_2) \cap vars(\mathit{eff}(a_3))) \neq \emptyset$*,* $direct(a_3, g_2)$ *holds and* g_2 *is a numeric condition, we fall into the case (iv) of Formula 5; we get that* $R(a_3, g_2) = NR(a_3, \langle y + z > 0 \rangle) = \langle (y + z - 20)[y/y+10, z/0] > 0 \rangle = \langle y + 10 - 20 > 0 \rangle = \langle y - 10 > 0 \rangle$*; finally, we get that* $R(a_3, g_2) \wedge pre(a_3) = \langle y > 10 \rangle \wedge \langle y < 10 \rangle \models \bot$*;*

We can stop the iteration, ignoring the evaluation of g_3*, as we have shown that there is a necessary condition of* $pre(\varepsilon)$ *which is always falsified by the execution of* a_3 *and therefore* $TF(a_3, \varepsilon)$ *holds.*

4.2 Avoiding Cascade of Events Handling

Another optimisation can also be applied to the *SIMEV* action. The idea is to detect if a set of events can not yield a cascade of events. We achieve this by slightly reinterpreting the Trigger-Free relation among events instead of that between an action and an event. That is:

Definition 19 (Trigger-Free and Universally Trigger-Free Event)**.** *Given a* PDDL+ *problem, let* $\varepsilon, \varepsilon' \in E$ *and let* $\varepsilon \in E$*, we say that* ε *is* Trigger-Free w.r.t. ε'*, namely* $TF(\varepsilon, \varepsilon')$ *iff for each state* s *where all* $\varepsilon'' \in E$ *are such that* $s \not\models pre(\varepsilon'')$*, we have that* $\gamma(s, \varepsilon) \models \neg pre(\varepsilon')$*. Moreover, we say that* ε *is* Universally Trigger-Free*, denoted with* $UTF(\varepsilon)$*, iff* ε *is Trigger-Free w.r.t. all events in* E*, i.e.,* $UTF(\varepsilon) \Leftrightarrow \forall \varepsilon' \in E,\ TF(\varepsilon, \varepsilon')$*.*

Note that under our assumptions, $TF(\varepsilon, \varepsilon)$ always holds since events have to self-deactivate.

We use the Trigger-Free notion to prevent the application of the expensive conditional effects W_{fired} and W_{satu} within *SIMEV* used for handling the cascade of events. More precisely, these conditional effects can be avoided if for each $\varepsilon \in E$ then $UTF(\varepsilon)$ holds.

Therefore *SIMEV* can be formulated by adding a switch that controls the relationship between each pair of events:

$$pre(SIMEV) = sim\text{-}ev$$

$$eff(SIMEV) = \begin{cases} W_{trig} \cup W_{\bot} \cup \{\top \rhd \{\neg sim\text{-}ev\}\} & \text{if } \forall \varepsilon \in E,\ UTF(\varepsilon) \text{ holds} \\ W_{trig} \cup W_{fired} \cup W_{satu} \cup W_{\bot} & \text{otherwise.} \end{cases}$$

The Trigger-Free relation can be under-approximated using Algorithm 1, pretty much verbatim.

Theorem 7. *The optimisation of* EXP *and* POLY *using the generalised SIMEV preserves the soundness and the completeness of both of the translations.*

Proof. It suffices to observe that W_{fired} and W_{satu} only serve the purpose of tracking which event has been already triggered and whether all events have been triggered. As none of the events triggers any other event, this is not necessary, so we can stop after exactly one execution of *SIMEV*. □

4.3 Translations Notation

In this section, we clarify the notation used in the translation presented in this work and its relationship with the translations presented in our previous work (Percassi et al., 2021). Given a translation $Z \in \{$EXP, POLY$\}$, we denote with:

- Z_0 the plain translation without any optimisation;
- Z_1 the translation using the optimisation described in Section 4.2;
- Z_2 the translation using the optimisation described in Section 4.1;
- Z_3 the translation jointly using the optimisations described in Sections 4.1 and 4.2.

In our previous work (Percassi et al., 2021), EXP and POLY used in the experiments correspond to EXP$_1$ and POLY$_1$.

5. Experimental Analysis

Our experimental analysis aims at assessing the extent to which the introduced translations allow to reformulate PDDL+ instances into instances amenable for PDDL2.1 planning engines, and investigating the impact of the described optimisations.

5.1 Experimental Settings

We consider three engines at the state of the art for pddl+ planning: Enhsp version 20 (Scala et al., 2020) with the additive interval-based relaxation heuristic (Scala, Haslum, Thiébaux, & Ramírez, 2016), SMTPlan (Cashmore et al., 2020), DiNo (Piotrowski, Fox, Long, Magazzeni, & Mercorio, 2016) and UPMurphi (Penna, Magazzeni, & Mercorio, 2012). As a pddl2.1 planning engine we use the well-known Metric-FF (Hoffmann, 2003). None of the considered systems requires the user to provide a bound on the length of valid plans. We did not consider other numeric planning systems such as lpg (Gerevini, Saetti, & Serina, 2008), Optic (Benton, Coles, & Coles, 2012) or the same Enhsp because none of them provides effective support for conditional effects and negative preconditions. All the planning engines have been run using default parameters. Numeric planners are used on numeric instances obtained using the proposed translations with $\delta = 1$. pddl+ planners which reason over a discrete timeline are used with $\delta = 1$. Finally, to study how the translations affect the makespan w.r.t. a reference, we compared the makespan of the plans found with Enhsp used with the A (h^{blind}) heuristic and those found with Metric-FF on all numeric tasks.

Our experiments were run on an Intel Xeon Gold 6140M CPU with 2.30 GHz. For each instance, we set a cutoff time of 900 seconds, and memory was limited to 8 GB.

For our experimental evaluation, we consider six benchmark domains. Three of them, Linear-Car (Lin-Car), Linear-Generator (Lin-Gen), and Solar-Rover (Rover), are well-known pddl+ benchmarks. Overtaking-Car (OT-Car) is a version of Linear-Car that extends the original domain by considering multiple lanes, and the need for the car to move between lanes in order to avoid obstacles (see Example 2.3 for a description of the model of this domain). Baxter (Bertolucci, Capitanelli, Maratea, Mastrogiovanni, & Vallati, 2019) and Urban-Traffic-Control (UTC) (Vallati, Magazzeni, Schutter, Chrpa, & McCluskey, 2016; McCluskey & Vallati, 2017) are taken from real-world applications. The Baxter domain exploits planning for supporting robots in dealing with articulated object manipulation tasks. The UTC domain models the use of planning for generating traffic light signal plans in order to de-congest an area of an urban region.

Our implementation of the translator is written in Python 3 and makes use of the SymPy library (Meurer et al., 2017) for solving the system of equations that arises for establishing Trigger-Free actions. The benchmark suite and the tool for translating pddl+ instances are available at `https://bit.ly/3OgMyNW`.

5.2 Size of the Translated Instances

First, we turn our attention to the size increase that can result from the use of the proposed translations. A direct comparison is not possible, as the original models and the translated models are encoded using different languages. For this reason, we introduce a notion of size increase ratio as follows. Let $\Pi = \langle F, X, I, G, A, P, E \rangle$ be a pddl+ problem and let $\Pi_T = \langle F', X', I', G', A', c \rangle$ the corresponding pddl2.1 problem obtained by using translation $T \in \{\text{POLY}, \text{EXP}\}$, we define the size increase ratio, denoted with r, introduced by T as:

$$r(T) = \frac{|A'| + |W'|}{|A| + |P| + |E| + |W|}$$

Domain	(P)	(E)	POLY				EXP			
			POLY$_3$	POLY$_2$	POLY$_1$	POLY$_0$	EXP$_3$	EXP$_2$	EXP$_1$	EXP$_0$
Rover (20)	4.0	5.0	1.9	1.9	1.9	1.9	1.8	1.8	1.8	1.8
Lin-Car (10)	2.0	0.0	1.7	1.7	1.7	1.7	1.3	1.3	1.3	1.3
Lin-Gen (10)	6.1	8.3	2.0	2.0	2.5	2.5	16.0	16.0	16.5	16.5
UTC (10)	34.1	15.8	2.5	2.5	2.8	2.8	16384:4	16384:4	16384:8	16384:8
Baxter (10)	56.0	22.0	1.5	1.5	1.7	1.7	—	—	—	—
OT-Car (20)	4.1	5.4	1.3	1.3	1.6	1.6	2.1	2.1	2.4	2.4

Table 1: For each domain, $\mu(jPj)$ and $\mu(jEj)$ denote the average number of grounded processes and events, respectively, while r(POLY) and r(EXP) denote the average size increase ratio of the instances. Between brackets, the number of problem instances considered for each benchmark domain. Symbol "—" indicates a translation failure due to memory limits. " " indicates that in the UTC domain we only consider 4 instances out of 10; the translation failed due to memory limits in the remaining instances.

where W and W^{0} denote the set of conditional effects of of Π and Π_T, respectively. In other words, we measure the size of the PDDL+ model in terms of actions, processes, and events, and we measure the size of the corresponding PDDL2.1 model in terms of actions and the number of conditional effects introduced by the compilation. This is because conditional effects play a major role in the proposed translations, and can be challenging to deal with by planners.

Table 1 provides an overview of the average number of processes and events of the considered benchmark domains, and the increased ratio obtained by using the proposed translations with/without the optimisations. The EXP translations are significantly larger than the POLY translations, regardless of the optimisation. This is not the case in domains where the number of processes and events is very limited, such as Linear-Car and Solar-Rover, where the EXP models are smaller than the POLY ones. Intuitively, this is due to the fact that when the number of processes is very small, the additional actions needed by the POLY translation can lead to a larger model. The use of such actions is instead beneficial in large instances, where the EXP approach can sometimes blow up the memory budget.

Considering the size increase ratio, the POLY$_2$ optimisation can usually lead to smaller instances for both the considered translations.

5.3 Assessment of Performance

Table 2 shows the performance achieved by our translations, and optimisation on the considered benchmark domains when run using the Metric-FF planning system. Results are shown in terms of coverage (number of solved instances), CPU-Time, quality of the generated plans (makespan), and number of nodes evaluates during the planning process. CPU-Time, quality, and evaluated nodes are presented as average over the instances solved by all the encodings under evaluation. Drawing a parallel with Table 1, it is easy to notice that in domains where there is a large number of processes and events, POLY translation delivers the best performance. The EXP translations are extremely large and hard to be dealt with by the planning engine. Conversely, the EXP translation seems to be more suitable for compact problems where the number of processes and events is limited. It is worth

Domain	Coverage							
	POLY				EXP			
	POLY_0	POLY_1	POLY_2	POLY_3	EXP_0	EXP_1	EXP_2	EXP_3
Rover (20)	**20**	19	**20**	19	**20**	**20**	**20**	**20**
Lin-Car (10)	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
Lin-Gen (10)	6	**10**	5	**10**	3	3	5	5
UTC (10)	0	**7**	0	**7**	0	0	0	0
Baxter (20)	1	**19**	1	17	0	0	0	0
OT-Car (20)	14	18	14	17	14	**19**	14	**19**
Σ	51	83	50	80	47	52	49	54

Domain	CPU-Time (seconds)							
	POLY				EXP			
	POLY_0	POLY_1	POLY_2	POLY_3	EXP_0	EXP_1	EXP_2	EXP_3
Rover (20)	64.7	64.6	69.1	60.5	15.0	15.6	13.7	**12.8**
Lin-Car (10)	10.7	7.0	7.2	8.3	8.8	7.5	**5.0**	8.1
Lin-Gen (10)	114.8	**28.7**	107.7	35.0	87.3	76.3	97.1	82.1
UTC (10)	—	90.6	—	**73.0**	—	—	—	—
Baxter (20)	894.1	17.9	637.0	**2.9**	—	—	—	—
OT-Car (20)	61.4	15.8	56.8	12.6	7.3	5.8	5.6	**4.5**

Domain	Makespan							
	POLY				EXP			
	POLY_0	POLY_1	POLY_2	POLY_3	EXP_0	EXP_1	EXP_2	EXP_3
Rover (20)	**522.2**	**522.2**	**522.2**	**522.2**	**522.2**	**522.2**	**522.2**	**522.2**
Lin-Car (10)	16.1	16.1	16.1	16.1	**14.9**	**14.9**	**14.9**	**14.9**
Lin-Gen (10)	**1006.0**	**1006.0**	**1006.0**	**1006.0**	1010.0	1010.0	1010.0	1010.0
UTC (10)	—	**54.6**	—	**54.6**	—	—	—	—
Baxter (20)	**5.0**	**5.0**	**5.0**	**5.0**	—	—	—	—
OT-Car (20)	42.1	42.1	41.2	41.2	24.5	24.5	**23.8**	**23.8**

Domain	Evaluated Nodes (1000)							
	POLY				EXP			
	POLY_0	POLY_1	POLY_2	POLY_3	EXP_0	EXP_1	EXP_2	EXP_3
Rover (20)	14.7	14.7	14.7	14.7	**8.7**	**8.7**	**8.7**	**8.7**
Lin-Car (10)	2.6	2.6	2.6	2.6	**0.4**	**0.4**	**0.4**	**0.4**
Lin-Gen (10)	11.1	11.1	11.1	11.1	2.1	2.1	**2.0**	**2.0**
UTC (10)	—	**131.1**	—	**131.1**	—	—	—	—
Baxter (20)	0.2	**0.1**	0.2	**0.1**	—	—	—	—
OT-Car (20)	122.1	122.1	92.0	92.0	**4.9**	**4.9**	5.6	5.6

Table 2: Domain by domain performance achieved by Metric-FF when run with the presented translations, with $\delta = 1$, and optimisations. Results are presented in terms of coverage (number of solved instances), average CPU-Time, average quality (makespan), and average number of nodes expanded during the search process. Averages are calculated considering instances solved by all approaches. "—" indicates that no instances can be considered for the average calculation.

noticing that these domains are not necessarily leading to instances that are easier to be solved than those domains that include a larger number of processes and events – in fact, there is no direct relationship between the size and complexity of a problem to be solved.

On compact instances, the results presented in Table 2 indicate that the use of the EXP translation can generally lead to the best performance in terms of the number of evaluated nodes, CPU-Time, and quality of the generated solutions. Overall, the quality of computed plans tends to be similar, regardless of the translation used. The only notable exception is Overtaking-Car, where the exp translations better support the planning process of Metric-FF.

With regards to the optimisations, results in Table 2 suggest that POLY_1 is the best optimisation for poly and that EXP_3 is the best combination for the exp translation. Those optimisations are adopted for the remainder of the experimental analysis.

5.4 Results Contextualisation

We are now in the best position to contextualise the results achieved by the proposed translations using the selected planning engine, Metric-FF, with the direct pddl+ representation using different pddl+ planning engines. Table 3 shows the achieved results in terms of the number of solved problems, by the considered planning approaches on the benchmark domains. It is also worth remarking that some of the pddl+ planning engines required the models to be modified in order to generate a solution: those are indicated using "+". The presented results highlight that the proposed translations are effective in supporting the use of pddl2.1 planning engines for solving complex hybrid planning problems. In fact, the use of the proposed translations allows a pddl2.1 planning engine to even outperform native pddl+ engines. This suggests that the proposed translations can foster the exploitation of pddl+ in real-world applications, by extending the pool of domain-independent planning engines.

Figure 2 gives some insights into the CPU-Time needed by the considered systems to solve the benchmark problems. All the approaches are able to quickly solve a large number of considered instances. When using the poly translation, the pddl2.1 planning engine is able to solve approximately 80 instances in less than 100 CPU-Time seconds. Notably, the curve of Metric-FF with poly does not flatten as quickly as others, suggesting that the combination of models and planning engine can effectively tackle also challenging instances, that require a significant amount of CPU-Time to be devoted to the search space exploration.

With regards to the quality of the generated plans, measured as makespan, we did not observe any significant overall difference between plans generated using the pddl+ or the pddl2.1 models. For a more extensive comparison, we used Enhsp with A (h^{blind}) heuristics (to ensure systematic exploration), which is equivalent to UPMurphi in terms of quality of generated solutions. In particular, the comparison between the makespans of the plans found with A (h^{blind}) and Metric-FF, used with different encodings and all optimisations, shows that there are some domains, such as Solar-Rover and UTC, where plans with the same makespan are found. In a few other domains, such as Baxter, Linear-Car and Overtaking-Car, Metric-FF finds plans with longer makespans w.r.t. A (h^{blind}). For instance, in Linear-Car, A (h^{blind}) finds solutions of average quality 11, compared to 14.9 of the best combination of Metric-FF. In Linear-Generator, both poly and exp find plans slightly longer (less than 0.1%) than A (h^{blind}). In general, it emerges that poly, although it allows obtaining the best coverage, produces in some cases plans with longer makespan when compared with those obtainable with exp. However, when comparing the plans generated using translated and original pddl+ models, quality seems to be more affected by the planning approach exploited by the engine, rather than by the use of a specific formulation.

Domain	Metric-FF		DiNo	Enhsp20	SMTPlan	UPMurphi
	POLY_1	EXP_3				
Rover (20)	19	**20**	**20$^+$**	5	19	4
Lin-Car (10)	**10**	**10**	**10$^+$**	**10**	**10$^+$**	**10**
Lin-Gen (10)	**10**	5	**10$^+$**	**10**	**10$^+$**	1
UTC (10)	**7**	0	0	**7**	0	1
Baxter (20)	**19**	0	7	17	8	12
OT-Car (20)	18	**19**	0	19	0	5
Σ	**83**	54	47	68	47	33

Table 3: Number of problems solved by the considered planning approaches. Between brackets, we indicate the number of problem instances considered for each domain. POLY and EXP are used to indicate that, respectively, the polynomial or the exponential translation has been used, in both cases with $\delta = 1$. "+" denotes that the reported result refers to a variant of the domain model we considered, modified to allow the specific engine to reason upon it. Bold indicates the best results.

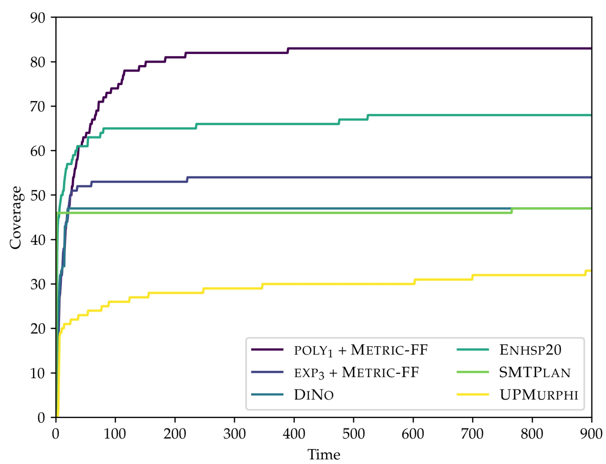

Figure 2: Total number of instances solved by each of the considered planning approaches, over time.

6. Related Work

A range of techniques have been introduced to support the reasoning on PDDL+ instances employing a translation. Balduccini, Magazzeni, Maratea, and Leblanc (2017) proposed an approach for translating a given PDDL+ instance in a Constraint ASP instance (Baselice, Bonatti, & Gelfond, 2005), but the process has to be done manually by an expert of the field. A number of approaches have been introduced to translate PDDL+ instances into Satisfiability Modulo Theories (SMT) (Barrett & Tinelli, 2018) problems (Bryce, Gao, Musliner, & Goldman, 2015; Scala, Ramírez, Haslum, & Thiébaux, 2016; Cashmore et al., 2020) or a mix of linear programming and SAT instances (Shin & Davis, 2005). These translations differ from each other in the way the compilation is carried on; some of them make use of an intermediate translation step into hybrid automata (Bryce et al., 2015; Heinz, Wehrle, Bogomolov, Magazzeni, Greitschus, & Podelski, 2019; Cashmore et al., 2020), other ones perform a more direct translation (Shin & Davis, 2005). We follow on these lines, yet, as we map the problem into another planning problem, we are not required to provide an upper limit on the plan length as these approaches do. It is indeed the planning engine that needs to figure out the length of the plan complying with the problem constraints. Conversely, all the above translations require the ability to anticipate a maximum number of time points, which is a difficulty shared with simpler forms of planning too, such as SAT-based planning (Kautz & Selman, 1992; Rintanen, Heljanko, & Niemelä, 2006). Moreover, as we build PDDL2.1 formulations, we can in principle exploit all heuristics and search guidance mechanisms developed for PDDL2.1 problems (Hoffmann, 2003; Scala et al., 2016, 2020). We indeed do so indirectly by experimenting with all those planners that do implement such heuristics. Yet, as the number of planning engines that supports conditional effects effectively is very limited, we could only exploit Metric-FF to a large extent. We highlight that this is not a limit of our translations, and believe that with the addition of newer and more expressive planning engines our translations can benefit the solving of PDDL+ even more.

Related to some extent is also the work by Coles and Coles (2014), which introduced a translation between a non-discretised PDDL+ problem instance and a PDDL2.1 temporal continuous instance, but showed that such way does not lead to models that are suitable for PDDL2.1 planning engines. Our approach is instead aiming at level 2 of the PDDL2.1 language. The language that we are targeting does not have a notion of time, with the result that constructing a planning engine supporting it is much easier than one that needs to provide native support for temporal reasoning.

A different line of work in automated planning focuses on reformulating models, to make them more amenable for planning engines, but without involving a translation to a different language or formalism. With regards to PDDL+ models, Franco, Vallati, Lindsay, and McCluskey (2019) introduced a technique for minimising the ground size of PDDL+ planning instances by reducing the arity of sparse predicates, i.e., predicates with a very large number of possible groundings, out of which very few are actually exploited in the planning problems.

There is an interesting parallel with compilations devised for classical planning models (Nebel, 2000; Gazen & Knoblock, 1997). In particular, our exponential translation anticipates the possible contexts a system is in much as the exponential encoding by (Gazen & Knoblock, 1997) compiles away conditional effects, while our polynomial translation captures the semantics of processes unrolling them into several actions, much as (Nebel, 2000) proposes to simulate the execution of conditional effects. As these two approaches have contributed to the discovery of several techniques and heuristics for classical planning (Haslum, 2013; Röger, Pommerening, & Helmert, 2014), we believe that our schemata can do the same for the much more involved case of PDDL+.

Finally, it is worth noting that our work provides a slightly alternative, more direct formalisation of the PDDL+ semantics. In fact, in the article presenting PDDL+ (Fox & Long, 2006), the authors resorted to a well-established formalism, known as Hybrid Automata (HA) (Henzinger, 1996), to explain the semantics of the proposed planning language. HAs are typically used for modelling systems having both a logical and a physical part where the latter is characterised by continuous dynamics. Bogomolov, Magazzeni, Podelski, and Wehrle (2014) further refine this translation into HAs by exploiting the more standard semantics of HA with specific attention devoted to must transitions prescribed by events and processes (Bogomolov et al., 2015). This line of research enables the use of tools specifically designed by the model-checking community, such as the SpaceEx model checker (Frehse et al., 2011).

Within the pddl+ literature there has not been a systematic study of what it means to find an optimal plan whereas pddl2.1 allows the definition of customised metrics to characterise the quality of the plans. Chen, Williams, and Fan (2021) address this problem by providing an encoding for translating a linear hybrid system into a Mixed Integer Linear Program (MILP), which can be tackled in an optimal way by using a MILP optimiser. Our work partially addresses this issue in the context of pddl+ under discrete semantics, as the translation into numeric tasks, combined with a simple cost function that is incremented by δ when the time advances (in poly when the *start* action is performed and in exp when the *SIM* action is performed), would allow one to find optimal plans in terms of makespan if they were used in a search scheme that guarantees to demonstrate the optimality of a solution. Further studies are needed in this direction.

Our work grounds on the seminal paper by Shin and Davis (2005), in particular as far as it is concerned by the adopted temporal ontology that is based on the notion of *superdense time* (Maler, Manna, & Pnueli, 1991). A superdense time model extends the real-valued timeline with additional information necessary to model the ordering of multiple simultaneous transitions. The extended timeline model provided by Shin and Davis (2005) allows us to capture when, informally speaking, something happens, i.e., an action, event or process starts or ends but time does not flow. Previous usages of this kind of temporal model can be traced back to TLPlan (Bacchus & Ady, 2001) and Optop (McDermott, 2003).

Batusov and Soutchanski (2019) recently propose an alternative logical semantic of pddl+ which extends *Situation Calculus* (SC), i.e., a logical formalism to represent dynamic domains, giving it a continuous component inspired by the HA interpretation. The target formalism of this mapping is an extension of a particular case of SC, namely *Basic Action Theory* (Reiter, 2001), which is a convenient representation for addressing reasoning problems. There is finally an interesting analogy on how hybrid automata are formalised in terms of timed transition systems (Henzinger, 1996). Indeed, our translations can be understood as a way to direct reasoning over the timed transition system and therefore use a *simpler* planner. In future work, we aim at studying this interpretation in deeper detail with the hope that this would further a prolific cross-fertilisation between techniques developed for planning models and hybrid automata.

7. Discussion

In this section, we frame the scope of this work and we discuss potential challenges that discretisation poses.

One of the main aims of this work is to show that solving a PDDL+ problem under a discrete interpretation of the timeline is equivalent to solving a PDDL2.1 problem without durative actions. This necessitated the formalisation of discrete PDDL+ according to a discretisation step $\delta \in \mathbb{Q}_{>0}$, which we assume to be an external parameter of the problem. This contribution, to the best of our knowledge, is the first attempt to formalise a discrete semantics for PDDL+ and we believe it is useful to the extent that it provides us with a formal framework for evaluating the correctness of those PDDL+ planning engines that natively work on discretised PDDL+.

In a wider sense, one may argue about the practical value of discretised models. A major practical benefit lies in the fact that discretised models allow planning engines to solve challenging and complicated tasks, otherwise impossible to solve by the current state of the art. There is of course the open question about the relation between PDDL+ and its discretised version, which however is beyond the scope of this work. We treat the two problems as two inherently different problems. It shall be noted that not all the possible discretisations have equal value concerning the problem to be solved. There is a trade-off to consider between large and small δ values; a very large δ can drastically reduce the complexity of the planning process, at the cost of an approximation of numeric dynamics that can lead the model to diverge too much from the ideal one, thus making it of little interest in practical terms. An extremely small value of δ increases the complexity of the instance to be solved, not guaranteeing at the same time the validity of solutions in the continuous settings, but allowing a more accurate approximation of the dynamics of the problem. The search for a suitable δ can be done by validating the solutions found towards smaller δ as we did in a recent work (Percassi, Scala, & Vallati, 2022); in this way, it would be possible to compute solutions with large δ_{s} (which stands for *search* δ) by conducting a more lightweight search and validate them effectively with an arbitrarily small δ_{v} (which stands for *validation* δ) which is sufficiently reliable according to the practitioner.

It is also worth noting that the discretisation of a PDDL+ problem may cause a tightening of its solution space because all the plans having actions executed in $\mathbb{R} \setminus \{\delta \cdot i \mid i \in \mathbb{Z}\}$ are expunged from the pool of possible solutions. However, there are problems for which even admitting arbitrarily small δ, would remain unsolved under discrete semantics. For example, suppose to have Linear-Car problem in which the car has to raise a flag at the following distance $d = \frac{1}{2}$ and $d = 1$. In a continuous sense this would mean executing an action, e.g., *raiseFlag*, at time $t = \sqrt{2}$ and $t = 1$. It follows that, since the numeric variables in PDDL2.1 problems take values in $\mathbb{Q}$, and the actions are timestamped in $\mathbb{Q}$, it is not possible to find a rational δ, even arbitrarily small, to solve this problem continuously. It is worth mentioning that Fox and Long (2003) discuss this possibility, suggesting that it is not possible to obtain an arbitrary precision of the plans, or over other numeric quantity, but it is possible to pragmatically validate them by admitting a numeric tolerance. Such a problem is known also in the context of HA (Henzinger & Raskin, 2000).

8. Conclusion

Hybrid PDDL+ models are amongst the most advanced models of systems and the resulting problems are notoriously difficult for planning engines to cope with. To deepen the understanding of PDDL+, and to support the solvability of PDDL+ instances, in this paper we introduced two translations from time-discretised PDDL+ to PDDL2.1 (level 2). The exponential translation leads to a numeric planning problem which is exponentially larger than the initial PDDL+ but preserves the number of discrete transitions. The polynomial translation instead leads to a smaller formulation but requires more transitions to generate a solution. We also presented two optimisations for the proposed translation schemata, that aim at reducing the size of the translated models while preserving the soundness and completeness of the approach. The optimisations exploit information about the structure of the considered planning instance to avoid unnecessary checks on actions and events.

Our experimental analysis demonstrated the usefulness of the introduced translations and optimisations in unlocking the exploitation of PDDL2.1 planning engines to solve challenging PDDL+ instances. The introduced optimisations make instances more amenable for domain-independent PDDL2.1 planning engines. In particular, the use of the optimised polynomial translation allows a PDDL2.1 planning engine to outperform a state-of-the-art PDDL+ planning engine across a wide range of benchmark domains. Summarising, the proposed translations can unlock the use of PDDL2.1 planning engines for tackling hybrid PDDL+ problems, with the clear advantage of significantly expanding the number of approaches that can be used to solve a problem instance.

In future, we plan to explore incomplete translations, where a trade-off can potentially be found between completeness and the size of the resulting PDDL2.1 instances. We are interested in incorporating the introduced translations into existing planning engines, possibly targeting the grounding step to minimise overhead (Scala & Vallati, 2021). Finally, we are interested in investigating potential synergies between the proposed translations and well-known reformulation approaches such as macro-actions, to generate PDDL2.1 models that are more suitable to domain-independent planning engines.

Acknowledgements

Francesco Percassi and Mauro Vallati are supported by the UKRI Future Leaders Fellowship [grant number MR/T041196/1]. Enrico Scala has been partially supported by AIPlan4EU, a project funded by EU Horizon 2020 research and innovation programme under GA n. 101016442.

References

Bacchus, F., & Ady, M. (2001). Planning with Resources and Concurrency: A Forward Chaining Approach In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pp. 417–424.

Balduccini, M., Magazzeni, D., Maratea, M., & Leblanc, E. C. (2017). CASP Solutions for Planning in Hybrid Domains *Theory and Practice of Logic Programming*, *17*(4), 591–633.

Barrett, C., & Tinelli, C. (2018). Satisfiability Modulo Theories In *Handbook of model checking*, pp. 305–343. Springer.

Baselice, S., Bonatti, P. A., & Gelfond, M. (2005). Towards an Integration of Answer Set and Constraint Solving In *Proocedings of the Twenty-First International Conference* *on Logic Programming, ICLP 2005*, pp. 52–66.

Batusov, V., & Soutchanski, M. (2019). A Logical Semantics for PDDL+ In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019*, pp. 40–48.

Benton, J., Coles, A. J., & 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*, pp. 2–10.

Bertolucci, R., Capitanelli, A., Maratea, M., Mastrogiovanni, F., & Vallati, M. (2019). Automated Planning Encodings for the Manipulation of Articulated Objects in 3D with Gravity In *Proceedings of the Eighteenth International Conference of the Italian Association for Artificial Intelligence, AI*IA 2019*, pp. 135–150.

Bogomolov, S., Magazzeni, D., Minopoli, S., & Wehrle, M. (2015). PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*, pp. 42–46.

Bogomolov, S., Magazzeni, D., Podelski, A., & Wehrle, M. (2014). Planning as Model Checking in Hybrid Domains In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2014*, pp. 2228–2234.

Bonassi, L., Gerevini, A. E., Percassi, F., & Scala, E. (2021). On Planning with Qualitative State-Trajectory Constraints in PDDL3 by Compiling them Away In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*, pp. 46–50.

Bryce, D., Gao, S., Musliner, D. J., & Goldman, R. P. (2015). SMT-Based Nonlinear PDDL+ Planning In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*, pp. 3247–3253. AAAI Press.

Cashmore, M., Magazzeni, D., & Zehtabi, P. (2020). Planning for Hybrid Systems via Satisfiability Modulo Theories *Journal of Artificial Intelligence Research*, *67*, 235–283.

Chen, J., Williams, B. C., & Fan, C. (2021). Optimal Mixed Discrete-Continuous Planning for Linear Hybrid Systems In *Proceedings of the Twenty-Fourth International Conference on Hybrid Systems: Computation and Control, HSCC 2021*, pp. 1–12.

Coles, A. J., & Coles, A. I. (2014). PDDL+ Planning with Events and Linear Processes In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*, pp. 74–82.

Cooper, M. C., Maris, F., & Régnier, P. (2010). Compilation of a High-level Temporal Planning Language into PDDL 2.1 In *Proceedings of the Twenty-Second IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010*, pp. 181–188.

Fox, M., Howey, R., & Long, D. (2005). Validating Plans in the Context of Processes and Exogenous Events In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pp. 1151–1156.

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

Fox, M., & Long, D. (2006). Modelling Mixed Discrete-Continuous Domains for Planning *Journal of Artificial Intelligence Research*, *27*, 235–297.

Franco, S., Vallati, M., Lindsay, A., & McCluskey, T. L. (2019). Improving Planning Performance in PDDL+ Domains via Automated Predicate Reformulation In *Proceedings of the Nineteenth Conference of Computational Science, ICCS 2019*, pp. 491–498.

Frehse, G., Guernic, C. L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., & Maler, O. (2011). SpaceEx: Scalable Verification of Hybrid Systems In *Proceedings of the Twenty-Third International Conference on Computer Aided Verification, CAV 2011*, pp. 379–395.

Gazen, B. C., & Knoblock, C. A. (1997). Combining the Expressivity of UCPOP with the Efficiency of Graphplan In *Proceedings of Recent Advances in AI Planning: Fourth European Conference on Planning, ECP 1997*, pp. 221–233.

Gerevini, A., Saetti, A., & Serina, I. (2008). An approach to efficient planning with numerical fluents and multi-criteria plan quality *Artificial Intelligence*, *172*(8-9), 899–944.

Gigante, N., Micheli, A., Montanari, A., & Scala, E. (2022). Decidability and complexity of action-based temporal planning over dense time *Artificial Intelligence*, *307*, 103686.

Grastien, A., & Scala, E. (2020). CPCES: A planning framework to solve conformant planning problems through a counterexample guided refinement *Artificial Intelligence*, *284*, 103271.

Haslum, P. (2013). Optimal Delete-Relaxed (and Semi-Relaxed) Planning with Conditional Effects In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013*, pp. 2291–2297.

Heinz, A., Wehrle, M., Bogomolov, S., Magazzeni, D., Greitschus, M., & Podelski, A. (2019). Temporal Planning as Refinement-Based Model Checking In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019*, pp. 195–199.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks *Artificial Intelligence*, *173*(5-6), 503–535.

Henzinger, T. A. (1996). The Theory of Hybrid Automata In *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science, LICS 1996*, pp. 278–292.

Henzinger, T. A., & Raskin, J. (2000). Robust Undecidability of Timed and Hybrid Systems In *Third International Workshop on Hybrid Systems: Computation and Control, HSCC 2000*, Vol. 1790, pp. 145–159.

Hoffmann, J. (2003). The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables *Journal of Artificial Intelligence Research*, *20*, 291–341.

Kautz, H. A., & Selman, B. (1992). Planning as Satisfiability In *Proceedings of the Tenth European conference on Artificial intelligence, ECAI 1992*, pp. 359–363.

Maler, O., Manna, Z., & Pnueli, A. (1991). From Timed to Hybrid Systems In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, Vol. 600, pp. 447–484.

McCluskey, T. L., & Porteous, J. M. (1997). Engineering and compiling planning domain models to promote validity and efficiency *Artificial Intelligence*, *95*(1), 1–65.

McCluskey, T. L., & Vallati, M. (2017). Embedding Automated Planning within Urban Traffic Management Operations In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*, pp. 391–399. AAAI Press.

McCluskey, T. L., Vaquero, T. S., & Vallati, M. (2017). Engineering Knowledge for Automated Planning: Towards a Notion of Quality In *Proceedings of the Knowledge Capture Conference, K-CAP 2017*, pp. 14:1–14:8. ACM.

McDermott, D. V. (2003). Reasoning about Autonomous Processes in an Estimated-Regression Planner In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling, ICAPS 2003*, pp. 143–152. AAAI.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., & Scopatz, A. (2017). SymPy: symbolic computing in Python *PeerJ Computer Science*, *3*, e103.

Nebel, B. (2000). On the Compilability and Expressive Power of Propositional Planning Formalisms *Journal of Artificial Intelligence Research*, *12*, 271–315.

Palacios, H., & Geffner, H. (2009). Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width *Journal of Artificial Intelligence Research*, *35*, 623–675.

Penna, G. D., Magazzeni, D., & Mercorio, F. (2012). A universal planning system for hybrid domains *Applied Intelligence*, *36*(4), 932–959.

Percassi, F., & Gerevini, A. E. (2019). On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019*, pp. 320–328. AAAI Press.

Percassi, F., Scala, E., & Vallati, M. (2021). Translations from Discretised PDDL+ to Numeric Planning In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*, pp. 252–261. AAAI Press.

Percassi, F., Scala, E., & Vallati, M. (2022). The Power of Reformulation: From Validation to Planning in PDDL+ In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*, pp. 288–296. AAAI Press.

Piotrowski, W. M., Fox, M., Long, D., Magazzeni, D., & Mercorio, F. (2016). Heuristic planning for hybrid systems In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 4254–4255. AAAI Press.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT press.

Rintanen, J., Heljanko, K., & Niemelä, I. (2006). Planning as satisfiability: parallel plans and algorithms for plan search *Artificial Intelligence*, *170*(12-13), 1031–1080.

Röger, G., Pommerening, F., & Helmert, M. (2014). Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting In *Proceedings of the Twenty-First European Conference on Artificial Intelligence, ECAI 2016*, pp. 765–770.

Scala, E., Haslum, P., Thiébaux, S., & Ramírez, M. (2016). Interval-Based Relaxation for General Numeric Planning In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence, ECAI 2016*, Vol. 285, pp. 655–663.

Scala, E., Haslum, P., Thiébaux, S., & Ramírez, M. (2020). Subgoaling Techniques for Satisficing and Optimal Numeric Planning *Journal Artificial Intelligence Research*, *68*, 691–752.

Scala, E., Ramírez, M., Haslum, P., & Thiébaux, S. (2016). Numeric Planning with Disjunctive Global Constraints via SMT In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*, pp. 276–284. AAAI Press.

Scala, E., & Vallati, M. (2021). Effective grounding for hybrid planning problems represented in PDDL+ *The Knowledge Engineering Review*, *36*.

Shin, J., & Davis, E. (2005). Processes and continuous change in a SAT-based planner *Artificial Intelligence*, *166*(1-2), 194–253.

Taig, R., & Brafman, R. I. (2013). Compiling Conformant Probabilistic Planning Problems into Classical Planning In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013*. AAAI Press.

Vallati, M., Chrpa, L., McCluskey, T. L., & Hutter, F. (2021). On the Importance of Domain Model Configuration for Automated Planning Engines *Journal of Automated Reasoning*, *65*(6), 727–773.

Vallati, M., Magazzeni, D., Schutter, B. D., Chrpa, L., & McCluskey, T. L. (2016). Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 3188–3194. AAAI Press.