Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

The Power of Reformulation: From Validation to Planning in PDDL+

Francesco Percassi,[1] Enrico Scala,[2] Mauro Vallati [1]

[1] School of Computing and Engineering, University of Huddersfield, UK
[2] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
f.percassi@hud.ac.uk, enrico.scala@unibs.it, m.vallati@hud.ac.uk

Abstract

PDDL+ allows the formal specification of systems representing mixed discrete-continuous representation, under both discrete and continuous dynamics. To support the validation of PDDL+ tasks, that is currently performed by a very limited number of tools, in this work we propose an approach for exploiting any domain-independent PDDL+ or PDDL2.1 planning engine for validating PDDL+ plans. We introduce a set of translations that, given a PDDL+ plan and the corresponding PDDL+ task, generate a new PDDL+ or PDDL2.1 whose solvability is bound to the validity of the considered plan. We empirically evaluate the usefulness of the proposed approach on a range of PDDL+ benchmarks under an interpretation of time that can be either continuous (through a PDDL+ translation) or discrete (through a PDDL+ or a PDDL2.1 translation).

Introduction

The nature of real-world applications often necessitates the ability to represent and reason in terms of hybrid discrete/-continuous changes. In automated planning, this necessity lead to the design of a dedicated language called PDDL+ (Fox and Long 2006), that supports the compact encoding of hybrid models involving mixed discrete/continuous effects, processes, exogenous events, and continuous change. Hybrid PDDL+ problems are notoriously hard to cope with, due to the intrinsic difficulties of reasoning with numeric variables (Helmert 2002), and time in an intertwined way.

The importance of validation for automated planning has long been recognised in both domain-independent planning (McCluskey and Porteous 1997) and in space applications (Penix, Pecheur, and Havelund 1998). In real-world applications validation is a crucial process to assess the overall quality of a planning model as well as the correctness of the planning process as a whole (Bensalem, Havelund, and Orlandini 2014; McCluskey, Vaquero, and Vallati 2017). In particular, plan validation plays a major role in real-world applications, as it aims at checking if a given plan is executable and produces a final state where all the goal conditions are satisfied, given a corresponding planning knowledge model.

The flagship tool for plan validation is VAL (Howey, Long, and Fox 2004; Fox, Howey, and Long 2005), which supports the validation of plans of a large number of PDDL planning formalisms, e.g., PDDL2.1 (Fox and Long 2003), PDDL3 (Gerevini et al. 2009) and PDDL+ (Fox and Long 2006). The exacerbated complexity of PDDL+ problems is also reflected in the increased complexity of validating PDDL+ solution plans against the corresponding PDDL+ knowledge models (Howey, Long, and Fox 2004). On top of that, it is also arguable that the reliance on a single software tool for validating complex hybrid plans from disparate real-world applications lays itself open to faults due to potential bugs or limited support of PDDL+ language features. To mitigate the mentioned issues, one approach is to extend the number of validating tools (see for instance INVAL[1]). However, the development of such tools for PDDL+ is extremely time-consuming and demanding – and this is one of the reasons why there is just one validation tool available.

A different line of research looked into the reformulation of the validation task into theorem proving (Abdulaziz and Lammich 2018). This has the advantage of reducing the possibility of bugs but is currently limited to classical planning formalism. Arguably, there is a wide availability of planning engines. Their exploitation as part of a validation framework would be beneficial to tackle the issues mentioned above related to the reliance on a single validation system. In this work, we propose an innovative reformulation-based methodology for validating PDDL+ plans, that allows one to tap into the available pool of domain-independent planning engines. More specifically, we introduce a set of translations that, given a PDDL+ task and a solution plan, allows one to generate a corresponding *validating* PDDL+ task whose solvability is intrinsically related to the validity of the considered solution plan against the PDDL+ model. In this way, any planning engine that supports PDDL+ can also be used to validate PDDL+ solutions, thus significantly extending the range of software tools available for the validation process. Further, according to the characteristics of the planning engine, the validity of a solution can be tested against different semantics. The use of engines that exploit a discretisation-based approach to PDDL+, such as ENHSP (Scala et al. 2016), allows demonstrating the validity of a plan with respect to the discrete semantics, while the use of engines capable of reasoning over a continuous timeline, like SMTPLAN (Cashmore, Magazzeni, and Zehtabi 2020),

[1] https://github.com/patrikhaslum/INVAL

allows to test the validity with regards to the continuous semantics. Further, leveraging the work by Percassi, Scala, and Vallati (2021) to translate PDDL+ tasks into numeric PDDL2.1 (level 2) tasks, we introduce an optimised translation that allows extending the pool of planning engines to include those that support PDDL2.1. Our approach enables the validation of domain models using non-polynomial dynamics, and the validation of plans under discrete semantics (Percassi, Scala, and Vallati 2021). We are not aware of approaches that are capable of handling these two aspects.

Our empirical analysis assesses the merits of the proposed approach and evaluates the usefulness of the introduced translations over a large set of PDDL+ benchmarks under both discrete and continuous semantics.

Background

In this section we report on the PDDL+ problem (Fox and Long 2006). We specify our problems using propositional formulas over numeric and Boolean conditions defined over sets of numeric and Boolean variables. A numeric condition is of the form $\langle \xi \bowtie 0 \rangle$ with ξ being a numeric expression, and $\bowtie \in \{\leq, <, =, >, \geq\}$. A Boolean condition is of the form $f = \{\top, \bot\}$ with f being a Boolean variable.

A PDDL+ planning problem Π is the tuple $\langle F, X, I, G, A, E, P \rangle$ in which each element is detailed in the following. F and X are the Boolean and numeric variables. Numeric variables take values from $\mathbb{R}$. 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 $c \triangleright e$. Each conditional effect $c \triangleright e$ is such that (i) c is a formula and (ii) e is a set of Boolean assignments of the form $\langle f := \{\bot, \top\} \rangle$ or numeric assignments of the form $\langle \{asgn, inc, dec\}, x, \xi \rangle$ where ξ is a numeric expression. 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 ξ is the net derivative of x.

Let $a = \langle p, e \rangle$ be an action/event/process, we use *pre*(a) to refer to the precondition p of a, and *eff*(a) to the effect e of a. Moreover, in the following we will use a, ρ, and ε to refer to a generic action, process, and event, respectively. In order to make the notation more concise, Boolean conditions and assignments of the form $\langle f = \bot \rangle$ ($\langle f := \bot \rangle$) and $\langle f = \top \rangle$ ($\langle f := \top \rangle$) are shortened to f and $\neg f$, and conditional effects of the form $\top \triangleright e$ are rewritten as e.

A PDDL+ plan π_t is a pair $\langle \pi, \langle t_s, t_e \rangle \rangle$ where: $\pi = \langle a_1, t_1 \rangle, ..., \langle a_n, t_n \rangle$ with $t_i \in \mathbb{Q}$ is a sequence of time-stamped actions; $\langle t_s, t_e \rangle$, with $t_s, t_e \in \mathbb{Q}$ and $t_s \leq t_e$, is the envelope within which π is performed. We say that π_t is well-formed iff $\forall\, i, j \in [1..n]$ and $i < j$, then $t_i \leq t_j$ and $t_s \leq t_i \leq t_e$ hold. Hereinafter we consider just well-formed plans.

Intuitively, a PDDL+ problem consists in finding plans along a potentially infinite timeline, whilst conforming to a number of processes and events that may change the state of the world as time goes by. Both processes and events are applied as soon as their preconditions become satisfied (must transitions); differently, actions are decisions that need to be taken (may transitions).

Following Percassi, Scala, and Vallati (2021) and Shin and Davis (2005), we formalise the PDDL+ semantics through the notion of time points, histories and the projection of a plan given a domain. We assume the reader is familiar with well known notions of action/event applicability, and use in the rest $\gamma(s, \cdot)$ to denote the state resulting by applying either an action/event ($\gamma(s, a)$) or a sequence of action/events ($\gamma(s, \langle a_1, ..., a_n \rangle)$) in a state s.

A time point T is a pair $\langle t, n \rangle$ where $t \in \mathbb{R}$ and $n \in \mathbb{N}$. Time points over $\mathbb{R} \times \mathbb{N}$ are ordered lexicographically. A history $\mathcal{H}$ over an interval $\mathcal{I} = [T_s, T_e]$ maps each time point in $\mathcal{I}$ into a situation. A "situation at time point T" is the tuple $\mathcal{H}(T) = \langle \mathcal{H}_A(T), \mathcal{H}_s(T) \rangle$, where $\mathcal{H}_A(T)$ is the sequence of actions executed at time point T and $\mathcal{H}_s(T)$ is a state, i.e., an assignment to all variables in X and F at time point T. We denote by $\mathcal{H}_s(T)[v]$ and $\mathcal{H}_s(T)[\xi]$ the value assumed by $v \in F \cup X$ and by a numeric expression ξ, respectively, in state s at time T. $E_{trigg}(T)$ indicates the sequence of active events in T. T is a significant time point (STP) of $\mathcal{H}$ over $\mathcal{I}$ iff either at least an action is applied ($\mathcal{H}_A(T) \neq \langle\rangle$), a non empty sequence of events is triggered ($E_{trigg}(T) \neq \langle\rangle$),[2] at least a process ρ has started or stopped, the precondition of ρ becomes (un)satisfied, or there has been a discrete change just before. A history $\mathcal{H}$ is monotonous over a real interval $\mathcal{I}_t$ if there are no STPs in $\mathcal{I}_t$.

The validity of plans can be determined both over a continuous or a discrete interpretation of time. Both require the definition of a PDDL+ plan (discrete) projection.

Definition 1 (PDDL+ plan projection)**.** *We say that $\mathcal{H}^\pi$ is* a projection of π_t which starts in I *iff $\mathcal{H}^\pi$ induces a sequence of STPs $T_\mathcal{H} = \langle T_0 = \langle t_s, 0 \rangle, ..., T_m = \langle t_e, n_m \rangle \rangle$ such that $\mathcal{H}^\pi$ is defined over $\mathcal{I} = [T_0, T_m]$ with $\mathcal{H}^\pi_s(T_0) = I$, $\mathcal{H}^\pi_A(T_m) = \langle\rangle$, $E_{trigg}(T_m) = \langle\rangle$ and, for $i \in [0..m]$, the following rules hold:*

- **R1** *$E_{trigg}(T_i) \neq \langle\rangle$ iff $\mathcal{H}^\pi_s(T_{i+1}) = \gamma(\mathcal{H}^\pi_s(T_i), E_{trigg}(T_i))$, $\mathcal{H}^\pi_A(T_i) = \langle\rangle$, $t_{i+1} = t_i$ and $n_{i+1} = n_i + 1$;*
- **R2** *$\mathcal{H}^\pi_A(T_i) \neq \langle\rangle$ iff $\mathcal{H}^\pi_s(T_{i+1}) = \gamma(\mathcal{H}^\pi_s(T_i), \mathcal{H}^\pi_A(T_i))$, $E_{trigg}(T_i) = \langle\rangle$, $t_{i+1} = t_i$ and $n_{i+1} = n_i + 1$;*
- **R3** *for each $\langle a_i, t_i \rangle, \langle a_j, t_j \rangle \in \pi$, with $i < j$ and $t_i = t_j$ there exists $T_k, T_z \in T_\mathcal{H}$ such that $a_i \in \mathcal{H}^\pi_A(T_k)$ and $a_j \in \mathcal{H}^\pi_A(T_z)$ where $t_k = t_z = t_i$ and $n_k < n_z$;*
- **R4** *$\mathcal{H}^\pi$ is monotonous over $\mathcal{I}_t = (t_i, t_{i+1})$, iff $t_{i+1} > t_i$ and for each $t \in \mathcal{I}_t$ and for each $x \in X$, we have that:*
  - *$\mathcal{H}^\pi_s(\langle t, 0 \rangle)[x]$ is continuous and differentiable;*
  - *for each $x \in X$ we have that:*

$$\frac{d\mathcal{H}^\pi_s(\langle t, 0 \rangle)[x]}{dt} = \sum_{\substack{\langle x', \xi \rangle \in \mathit{eff}(\rho),\ x' = x \\ \rho \in \mathcal{C}(\mathcal{I}_t)}} \mathcal{H}^\pi_s(\langle t, 0 \rangle)[\xi]$$

[2] We assume that PDDL+ problems are event-deterministic (Fox and Long 2006). Then, given a state in which multiple events are triggered, we can sequence them arbitrarily always obtaining the same outcome.

– *for each* $x \in X$ *we have that* $\mathcal{H}^{\pi}_s(\langle t_{i+1}, 0\rangle)[x] = \lim_{t \to t^-_{i+1}} \mathcal{H}^{\pi}_s(\langle t, 0\rangle)[x]$*, and values of unaffected variables persist up to* t_{i+1} *(frame-axiom).*

Definition 2 (Valid PDDL+ plan)**.** π_t *is* valid plan for Π *iff* $\mathcal{H}^{\pi}_s(T_m) \models G$ *and, for each* $a \in \mathcal{H}^{\pi}_A(T)$ *with* $T \in \mathcal{I}$*,* $\mathcal{H}^{\pi}_s(T) \models pre(a)$*.*

For the PDDL+ plan discrete projection we make use of a fixed discretisation step $\delta \in \mathbb{Q}$. δ implies that time points can only be of the form $\langle t = \delta \cdot n, n'\rangle$ with $n, n' \in \mathbb{N}$. We use $\mathcal{H}$ to indicate a history in the continuous context while $\mathbb{H}$ to indicate a history in the discrete context. Under a discretised semantics, the net derivative expressions are discretised using $\Delta(\xi, \delta) = \xi \cdot \delta$. For example, let $\langle x, 1.5 \cdot y\rangle$ ($\dot{x} = 1.5 \cdot y$) and $\delta = 2$ be a continuous numeric effect and a discretisation parameter, the discretised expression of x is equal to $\Delta(1.5 \cdot y, \delta) = 3 \cdot y$.

Definition 3 (PDDL+ plan discrete projection)**.** *We say that* $\mathbb{H}^{\pi}$ *is a* discrete projection of π_t which starts in I *iff* $\mathbb{H}^{\pi}$ *induces the STPs* $T_{\mathbb{H}} = \langle T_0 = \langle t_s, 0\rangle, ..., T_m = \langle t_e, n_m\rangle\rangle$ *where either* $t_{i+1} = t_i + \delta$ *or* $t_{i+1} = t_i$ *and, for* $i \in [0..m]$*, we have R1–R3 as for Definition 1 and R4 substituted with the following:*

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

$$\mathbb{H}^{\pi}_s(T_{i+1})[x] = \mathbb{H}^{\pi}_s(T_i)[x] + \sum_{\substack{\langle x', \xi\rangle \in eff(\rho),\ x' = x \\ \rho \in \{\rho \in P,\ \mathbb{H}^{\pi}_s(T_i) \models pre(\rho)\}}} \mathbb{H}^{\pi}_s(T_i)[\Delta(\xi, \delta)]$$

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

With reference to the rules R1–R4 of Defs. 1 and 3, R1 (R2) states that if at least an action (event) is executed (triggered) in a significant time point $\langle t, n\rangle$, then there necessarily exists a successor, i.e., $\langle t, n + 1\rangle$, whose state associated is calculated by simply applying the discrete effects of the action(s) (event(s)). R3 is used to enforce how actions of a PDDL+ plan π are projected over an history, preserving their original ordering in case they share the same time-stamp in π. R4 in Def. 1 is used to enforce how a numeric variable change continuously over time according to the active processes in those monotonous temporal intervals. R4 in Def. 3, is used to enforce how a numeric variables changes when time advances for a discrete quantum of time δ.

Definition 4 (Valid PDDL+ plan under δ discretisation)**.** π_t *is a* valid plan for Π under δ discretisation *iff* $\mathbb{H}^{\pi}_s(T_m) \models G$ *and, for each* $T \in \mathcal{I}$ *such that* $\mathbb{H}^{\pi}_A(T) \neq \langle\rangle$*, then* $\mathcal{H}^{\pi}_s(T) \models pre(a)$*.*

In this paper, we are interested in the validation problem. That is, we are given a PDDL+ problem and a plan, and we want to see whether such a plan is a valid solution plan for the problem over a continuous timeline or under a discretisation parameter δ.

Definition 5 ((Discrete) Validation Problem)**.** *Let* Π *be a* PDDL+ *problem,* π_t *be a plan for* Π*. The* validation problem *aims at establishing whether* π_t *is valid for* Π*. The* discrete validation problem *aims at establishing whether* π_t *is valid for* Π *under* $\delta \in \mathbb{Q}$ *discretisation.*

PDDL2.1 is the fragment of PDDL+ where there is no time; from a syntactical perspective, this is reflected in the absence of events and processes. A PDDL2.1 problem simply looks for sequences of actions that are applicable in the initial state and yields a state satisfying the goal. In the next section, we briefly survey a technique that solves the PDDL+ problem through a translation into a PDDL2.1 specification.

Solving Discrete PDDL+ via Numeric Planning

As shown by Percassi, Scala, and Vallati (2021), a PDDL+ problem can be transformed into a numeric one using two different translations, i.e., EXP and POLY. Such encodings called Π_{EXP} and Π_{POLY} can be used to seek discrete valid PDDL+ plans. The key idea in both translations consists in transforming all the processes and events into regular actions and forcing the occurrence of such actions when the planner decides to move time forward. The glaring difference between Π_{EXP} and Π_{POLY} is in how the processes' specifications are transferred into the actions. Indeed, Π_{EXP} devises a single action with exponentially many conditional effects, while Π_{POLY} decomposes all processes in a polynomial number of actions. Π_{POLY} turned out more effective than Π_{EXP} (Percassi, Scala, and Vallati 2021), therefore we turn our attention on this translation. We summarise the event-free translation step.

Let $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$ be an event-free PDDL+ problem, and let $\delta \in \mathbb{Q}$ be 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:

$$X^{cp} = \{x^{copy} \mid x \in X\}\ D = \bigcup_{\substack{ne \in eff(\rho) \\ \rho \in P}} \{done_{ne}\}$$

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

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

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

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

Whenever the passage of a discrete amount of time δ has to be simulated within Π_{POLY}, the sequence of actions $wait = \langle start, seq(A_P), end\rangle$ ($seq(A_P)$ is any permutation of all A_P actions) is enforced to be performed. A thorough discussion of this discretising translation is referred to in the work by Percassi, Scala, and Vallati (2021).

Validation via Translation

This section proposes a collection of translations for producing, given a PDDL+ task Π and a plan π_t for Π, a new PDDL+ problem $\Pi^{\pi_t}_{\mathcal{V}}$, where $\mathcal{V}$ denotes a specific translation. This

PDDL+ problem is built in such a way that finding a solution for it is equivalent to answering the following question: *is* π_t *valid for* Π*?* Conversely, proving that $\Pi^{\pi_t}_{\mathcal{V}}$ is unsolvable is equivalent to demonstrating that π_t is invalid for Π.

We propose a basic translation first, namely $\mathcal{V}_0$, and then two optimisations separately addressing two problems with $\mathcal{V}_0$, namely $\mathcal{V}_{\text{U}}$ and $\mathcal{V}_{\text{D}}$. After presenting these translations, we will discuss how these can be concatenated with POLY and EXP, obtaining a set of translations that generate a PDDL2.1 task. This way the pool of validation systems can be expanded even further; not only we can use PDDL+ planners, but also PDDL2.1 planners.

$\mathcal{V}_0$ translation Here, we present the simplest translation for producing a PDDL+ validating task, namely $\mathcal{V}_0$. The idea is to generate a new PDDL+ problem in which the actions belonging to π_t are the only executable ones. These actions have to be performed according to what is prescribed by the plan, that is, applying them in their respective time-stamps and preserving their order of execution.

Let $\Pi = \langle F, X, I, G, A, E, P\rangle$ be a PDDL+ problem, and let $\pi_t = \langle \pi, \langle t_s, t_e\rangle\rangle$ be a plan for Π. $\mathcal{V}_0$ produces a new validating PDDL+ problem $\Pi^{\pi_t}_{\mathcal{V}_0} = \langle F \cup F_A \cup \{\text{T}\}, X \cup \{\mathit{time}\}, I \cup \{\mathit{a\text{-}d}_0, \langle \mathit{time} := t_s\rangle, \text{T}\}, G \wedge \mathit{a\text{-}d}_n \wedge \langle \mathit{time} = t_e\rangle, A_\pi, E, P \cup \{\rho_{\mathit{time}}\}\rangle$[3] such that $\rho_{\mathit{time}} = \langle \text{T}, \{\langle \mathit{time}, 1\rangle\}\rangle$ and:

$$F_A = \bigcup_{i=0}^{n=|\pi|} \{\mathit{a\text{-}d}_i\}$$

$$A_\pi = \bigcup_{\langle a_i, t_i\rangle \text{ in } \pi} \{\langle \mathit{pre}(a_i) \wedge \mathit{a\text{-}d}_{i-1} \wedge \neg \mathit{a\text{-}d}_i \wedge \langle \mathit{time} = t_i\rangle, \mathit{eff}(a_i) \cup \{\mathit{a\text{-}d}_i\}\rangle\}$$

$\Pi^{\pi_t}_{\mathcal{V}_0}$ contains a new process, i.e. ρ_{time}. Such a process is always active and its only purpose is to progress time explicitly using the numeric variable *time*. Keeping track of the elapsed time is necessary to perform the π actions at their respective time-stamps.

$\mathcal{V}_0$ restricts the actions A of Π to A_π of $\Pi_{\mathcal{V}_0}$; it indeed only considers the actions in π. An action a in A is added to A_π iff it exists $\langle a_i, t_i\rangle$ *in* π such that $a = a_i$ (note that there can be multiple occurrences of the same action in A_π if a appears more than once in the plan). If this condition holds, then a is appropriately modified by extending its precondition so that a can be performed only whenever *time* is equal to t_i ($\langle \mathit{time} = t_i\rangle$). Furthermore, these actions are extended in their preconditions and effects through the novel F_A predicates so that they can be performed by following the prescribed order and executed at most once.

Example 1 ($\mathcal{V}_0$ translation)**.** *Let* $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$ *be a* PDDL+ *problem without events encompassing one Boolean variable, i.e.,* $F = \{f_1\}$*, two numeric variables, i.e.,* $X = \{x_1, x_2\}$ *and one process* $\rho_1 = \langle x_1 > 0, \{\langle inc, x_2, 1\rangle\}\rangle$ *(i.e.,* $P = \{\rho_1\}$*). Furthermore, let* $\pi_t = \langle \pi, \langle 0, 10\rangle\rangle$ *be a* PDDL+ *plan for* Π*, where* $\pi = \langle\langle a_1, 1\rangle, \langle a_2, 5\rangle, \langle a_3, 5\rangle, \langle a_4, 10\rangle\rangle$*. Actions in* π *are such that:* $\{a_1 = \langle f_1, \{\neg f_1\}\rangle, a_2 = \langle \neg f_1, \{f_1\}\rangle, a_3 = \langle\langle x_1 > 0\rangle, \{\langle inc, x_1, 1\rangle\}\rangle, a_4 = \langle\langle x_1 < 0\rangle, \{\langle inc, x_2, 1\rangle\}\rangle\} \subset A$*.*

$\mathcal{V}_0$ produces the validating PDDL+ *task* $\Pi^{\pi_t}_{\mathcal{V}_0} = \langle F \cup \{\mathit{a\text{-}d}_0, \mathit{a\text{-}d}_1, \mathit{a\text{-}d}_2, \mathit{a\text{-}d}_3, \mathit{a\text{-}d}_4, \text{T}\}, X \cup \{\mathit{time}\}, I \cup \{\mathit{a\text{-}d}_0, \langle \mathit{time} := 0\rangle, \text{T}\}, G \wedge \langle \mathit{time} = 10\rangle, \{a'_1, a'_2, a'_3, a'_4\}, \emptyset, \{\rho_1, \langle \text{T}, \{\langle \mathit{time}, 1\rangle\}\rangle\}\rangle$ *where:*

$$a'_1 = \langle f_1 \wedge \mathit{a\text{-}d}_0 \wedge \neg \mathit{a\text{-}d}_1 \wedge \langle \mathit{time} = 1\rangle, \{\neg f_1, \mathit{a\text{-}d}_1\}\rangle$$
$$a'_2 = \langle \neg f_1 \wedge \mathit{a\text{-}d}_1 \neg \mathit{a\text{-}d}_2 \wedge \langle \mathit{time} = 5\rangle, \{f_1, \neg \mathit{a\text{-}d}_1\}\rangle$$
$$a'_3 = \langle \langle x_1 > 0\rangle \wedge \mathit{a\text{-}d}_2 \wedge \neg \mathit{a\text{-}d}_3 \wedge \langle \mathit{time} = 5\rangle, \{\langle inc, x_1, 1\rangle, \mathit{a\text{-}d}_3\}\rangle$$
$$a'_4 = \langle \langle x_1 < 0\rangle \wedge \mathit{a\text{-}d}_3 \wedge \neg \mathit{a\text{-}d}_4 \wedge \langle \mathit{time} = 10\rangle, \{\langle inc, x_2, 1\rangle, \mathit{a\text{-}d}_4\}\rangle$$

Theorem 1 (Soundness and completeness of $\mathcal{V}_0$)**.** *Let* $\Pi = \langle F, X, I, G, A, E, P\rangle$ *be a* PDDL+ *problem, let* π_t *be a plan for* Π*, and let* $\Pi^{\pi_t}_{\mathcal{V}_0}$ *be the validating task obtained by using* $\mathcal{V}_0$*. It follows that* π_t *is valid for* Π *iff* $\Pi^{\pi_t}_{\mathcal{V}_0}$ *admits a solution.*

Proof Sketch. ($\Rightarrow$) Let $\pi_t = \langle \pi, \langle 0, t_e\rangle\rangle$ be a valid plan for Π (assume w.l.o.g. $t_s = 0$), and let $\pi'_t = \langle \pi_{\mathcal{V}_0}, \langle 0, t_e\rangle\rangle$ a plan for $\Pi^{\pi_t}_{\mathcal{V}_0}$ where, for each $\langle a_i, t_i\rangle$ in π there is $\langle a'_i, t_i\rangle$ in $\pi_{\mathcal{V}_0}$, where $a'_i \in A_\pi$ is the corresponding action of $a_i \in A$, executed at time t_i, compiled according to $\mathcal{V}_0$.

The extensions of $\Pi^{\pi_t}_{\mathcal{V}_0}$ w.r.t. Π do not affect the goal G and the original preconditions of the actions A. The novel process ρ_{time} affects only the novel *time* numeric variable, synchronising it with the actual time elapsed. The compiled actions are copies of the original actions except for the *time* condition in the preconditions, and the assignments over F_A variables. Let $\mathcal{H}$ and $\mathcal{H}'$ be the PDDL+ plan projections of π_t and π'_t, respectively and let τ and τ' be the sequences of states associated to each STPs of $\mathcal{H}$ and $\mathcal{H}'$, i.e., $\tau = \langle \mathcal{H}_s(T_0), ..., \mathcal{H}_s(T_m)\rangle$ and $\tau' = \langle \mathcal{H}'_s(T_0), ..., \mathcal{H}'_s(T_m)\rangle$. Given the aforementioned premises, proceed by induction over τ (τ'), and show that (i) if the $\langle a_i, t_i\rangle$ actions of π are applicable, so the corresponding ones $\langle a'_i, t_i\rangle$ are too (*instantaneous transition*), and (ii) numeric variables X change as time passes in $\Pi^{\pi_t}_{\mathcal{V}_0}$ in the same way as they change when time passes in Π (*temporal transition*). This also entails reaching the goal.

($\Leftarrow$) Let $\pi'_t = \langle \pi_{\mathcal{V}_0}, \langle 0, t_e\rangle\rangle$ be a valid plan for $\Pi^{\pi_t}_{\mathcal{V}_0}$, and let $\pi_t = \langle \pi, \langle 0, t_e\rangle\rangle$ a plan for Π built as follows: for each $\langle a'_i, t_i\rangle$ in $\pi_{\mathcal{V}_0}$ there is $\langle a_i, t_i\rangle$ in π, where $a_i \in A$ is the original action of $a'_i \in A_\pi$ that can be obtained by removing from a'_i the conditions and assignments concerning *time* and *a-d*, respectively. As noted in the opposite direction, the $\Pi^{\pi_t}_{\mathcal{V}_0}$ extension w.r.t. Π affects only the novel variables. Then, since Π is a reduced version of $\Pi^{\pi_t}_{\mathcal{V}_0}$, it is easy to notice that the operators of π are still applicable if their correspondents of $\pi_{\mathcal{V}_0}$ are, and numeric variables change as time passes in Π in the same way when time passes in $\Pi^{\pi_t}_{\mathcal{V}_0}$. □

This baseline has a weakness. The state-space induced by $\mathcal{V}_0$ is dense of dead-ends, i.e., states beyond which there is no solution. Indeed, all states where *time* $> t_e$ are dead-ends as they do not belong to the envelope $\langle t_s, t_e\rangle$ prescribed by the plan. Despite this, $\mathcal{V}_0$ allows their exploration.

[3] For lack of space the notation for the propositional variables *a-d* and T are the contraction of *a-done* and TRUE, respectively.

The translations presented below, i.e., $\mathcal{V}_{\text{U}}$ and $\mathcal{V}_{\text{D}}$, are aimed to overcome this problem by explicitly avoiding these dead-ends.

$\mathcal{V}_{\text{U}}$ translation The main idea of the $\mathcal{V}_{\text{U}}$ translation is to avoid some dead-ends through an explicit constraint on the numeric variable *time*. This constraint disallows the occurrence of any process whenever time gets beyond what is prescribed by the plan. More precisely, let $\Pi = \langle F, X, I, G, A, E, P\rangle$ be a PDDL+ problem, and let $\pi_t = \langle \pi, \langle t_s, t_e\rangle\rangle$ be a plan for Π, $\mathcal{V}_{\text{U}}$ produces a new validating PDDL+ problem $\Pi^{\pi_t}_{\mathcal{V}_{\text{U}}} = \langle F \cup F_A \cup \{\text{T}\}, X \cup \{\mathit{time}\}, I \cup \{\mathit{a\text{-}d}_0, \langle \mathit{time} := t_s\rangle, \text{T}\}, G \wedge \mathit{a\text{-}d}_n \wedge \langle \mathit{time} = t_e\rangle, A_\pi, E, P_{\text{U}}\rangle$. The $\Pi_{\mathcal{V}_{\text{U}}}$ problem has a similar structure to what produced using $\mathcal{V}_0$, with the following difference:

$$P_{\text{U}} = \bigcup_{\rho \in P \cup \{\rho_{\mathit{time}}\}} \{\langle \langle \mathit{time} < t_e\rangle \wedge \mathit{pre}(\rho), \mathit{eff}(\rho)\rangle\}$$

All processes in Π, including ρ_{time}, are extended with the same precondition, i.e., $\langle \mathit{time} < t_e\rangle$, so that, when time flows beyond t_e, then the processes P_{U} can not cause any change. In other words, after t_e, the environment gets static.

$\mathcal{V}_{\text{D}}$ translation Even with $\mathcal{V}_{\text{U}}$, it is still possible to do some unnecessary search steps for our validation problem.

Consider a plan $\pi_t = \langle \pi = \langle\langle a1, 0\rangle, \langle a2, 10\rangle\rangle, \langle 0, 10\rangle\rangle$ for a PDDL+ task Π. The action $a2$ executed at time 10 achieves the goal of Π, but its applicability is subordinated to the execution of $a1$ at time 0. Given $\Pi^{\pi_t}_{\mathcal{V}_{\text{U}}}$, the planner can choose at time 0 whether to execute $a1$ or to explore states with higher timings. Yet, this second choice will prevent the execution of $a2$ at time 10, necessary to reach the goal.

The following translation, namely $\mathcal{V}_{\text{D}}$, prevents situations like this by introducing fresh events that make everything else inapplicable when some timing condition is reached. This variant requires some notation. More precisely, let $\pi_t = \langle \pi = \langle\langle a_1, t_1\rangle, ..., \langle a_n, t_n\rangle\rangle, \langle t_s, t_e\rangle\rangle$ be a PDDL+ plan, and let $t \in \mathbb{Q}$, we indicate with $\pi(t)$ the sub-sequence of π containing all actions with time-stamp t, and ordered so as to preserve the order given by π. π can therefore be reshaped as a sequence of sub-sequences each containing actions with identical time-stamp, i.e., $\pi = \pi(t'_1) \cdot ... \cdot \pi(t'_m)$, where, if $t_s < t_e$, then $m \in [2...n+2]$, $t'_1 = t_s$ and $t'_m = t_e$. Note that the sub-sequences placed at the extremes of π may be empty if no action is executed at time t_s and t_e. Let $\pi(t')$ be a non empty sub-sequence of actions of π, $last(\pi(t'))$ denotes the position of the last action of $\pi(t')$ with respect to π. Furthermore we define the set of sub-intervals induced by π_t as follows $\mathcal{I}(\pi_t) = \bigcup_{i=1}^{m-1}\{\langle t'_i, t'_{i+1}\rangle\}$. We clarify this notations with an example.

Example 2 (Introductory notation)**.** *Let $\pi_t = \langle \pi = \langle\langle a_1, 1\rangle, \langle a_2, 5\rangle, \langle a_3, 5\rangle, \langle a_4, 10\rangle\rangle, \langle 0, 10\rangle\rangle$ be a PDDL+ plan. π can be reshaped as $\pi = \pi(0) \cdot \pi(1) \cdot \pi(5) \cdot \pi(10)$, where $\pi(0) = \langle\rangle$, $\pi(1) = \langle\langle a_1, 1\rangle\rangle$, $\pi(5) = \langle\langle a_2, 5\rangle, \langle a_3, 5\rangle\rangle$ and $\pi(10) = \langle\langle a_4, 10\rangle\rangle$. We get that $last(\pi(0))$ is undefined ($\pi(0) = \langle\rangle$), $last(\pi(1)) = 1$ (a_1 is the last action of $\pi(1)$ and the first of π_t), $last(\pi(5)) = 3$ (a_3 is the last action of $\pi(5)$ and the third of π_t) and $last(\pi(10)) = 4$ (a_4 is the last action of $\pi(10)$ and the fourth of π_t). The sub-intervals of π_t are thus defined as $\mathcal{I}(\pi_t) = \{\langle 0, 1\rangle, \langle 1, 5\rangle, \langle 5, 10\rangle\}$.*

Let $\Pi = \langle F, X, I, G, A, E, P\rangle$ be a PDDL+ problem, and let $\pi_t = \langle \pi, \langle t_s, t_e\rangle\rangle$ a plan for Π, $\mathcal{V}_{\text{D}}$ produces a new validating PDDL+ problem $\Pi^{\pi_t}_{\mathcal{V}_{\text{D}}} = \langle F \cup F_A \cup \{\text{T}\}, X \cup \{\mathit{time}\}, I \cup \{\mathit{a\text{-}d}_0, \langle \mathit{time} := t_s\rangle, \text{T}\}, G \wedge \mathit{a\text{-}d}_n \wedge \langle \mathit{time} = t_e\rangle \wedge \text{T}, A_\pi, E \cup E_{\text{D}}, P_{\text{D}} \cup \{\rho_{\mathit{time}}\}\rangle$. $\Pi^{\pi_t}_{\mathcal{V}_{\text{U}}}$ has a similar structure to that produced using $\mathcal{V}_0$, with the following differences:

$$E_{\text{D}} = \bigcup_{\substack{\pi(t') \text{ in } \pi \\ \pi(t') \neq \langle\rangle \wedge t' \neq t_e}} \{\langle \langle \mathit{time} > t'\rangle \wedge \neg \mathit{a\text{-}d}_{\mathit{last}(\pi(t'))} \wedge \text{T}, \{\neg\text{T}\}\rangle\}$$

$$P_{\text{D}} = \bigcup_{\rho \in P} \{\langle \mathit{pre}(\rho) \wedge \text{T}, \mathit{eff}(\rho)\rangle\}$$

The events E of Π are extended in $\Pi^{\pi_t}_{\mathcal{V}_{\text{D}}}$ with the set of novel events E_{D}. Each event $\varepsilon \in E_{\text{D}}$ refers to a non-empty subsequence $\pi(t')$ of π and is triggered whenever time flows beyond t' without the last $\pi(t')$ action being performed, i.e., when the predicate $\mathit{a\text{-}d}_{\mathit{last}(\pi_{t'})}$ is still false. Each triggered event $\varepsilon \in E_{\text{D}}$ makes T false, making the goal unreachable and every process unable to change the state of the world (note that T is added to each process and it required to be true in the goal $G \wedge \mathit{a\text{-}d}_n \wedge \langle \mathit{time} = t_e\rangle \wedge \text{T}$). The T variable is also added to the precondition of each fresh event since an event, when triggered, has to self-deactivate (Fox and Long 2006).

Composition of $\mathcal{V}_{\text{U}}$ and $\mathcal{V}_{\text{D}}$ $\mathcal{V}_{\text{U}}$ and $\mathcal{V}_{\text{D}}$ can be combined in a new schema that incorporates both enhancements denoted with $\mathcal{V}_{\text{UD}} = \mathcal{V}_{\text{U}} \circ \mathcal{V}_{\text{D}}$.

Example 3 ($\mathcal{V}_{\text{UD}}$ translation - Continuing on Ex. 1)**.** *Let Π be a PDDL+ task and let π_t be a plan for Π as defined in Example 1. $\mathcal{V}_{UD}$ produces the validating PDDL+ task $\Pi^{\pi_t}_{\mathcal{V}_{UD}} = \langle F \cup \{\mathit{a\text{-}d}_0, \mathit{a\text{-}d}_1, \mathit{a\text{-}d}_2, \mathit{a\text{-}d}_3, \mathit{a\text{-}d}_4, \text{T}\}, X \cup \{\mathit{time}\}, I \cup \{\mathit{a\text{-}d}_0, \langle \mathit{time} := 0\rangle, \text{T}\}, G \wedge \mathit{a\text{-}d}_n \wedge \langle \mathit{time} = 10\rangle \wedge \text{T}, A_\pi, E_D, P_{UD}\rangle$.*

We show how the events E_D are generated. The sub-sequences placed at the extremes of the plan, i.e., $\pi_{t=0}$, and $\pi_{t=10}$, do not generate events as $\pi_{t=0} = \langle\rangle$ while $\pi_{t=10} \neq \langle\rangle$ but $t = 10 = t_e$, and then they are not considered in generating E_D. The sub-sequences $\pi_{t=1} = \langle a_1\rangle \neq \langle\rangle$, with $last(\pi_{t=1}) = 1$, and $\pi_{t=5} = \langle a_2, a_3\rangle \neq \langle\rangle$, with $last(\pi_{t=5}) = 3$, generate two dead-ends events which are ε^1_D and ε^5_D, respectively (where the superscript denotes the time-stamps they refer to). According to E_D definition, we get that $E_D = \{\varepsilon^1_D, \varepsilon^5_D\}$ where $\varepsilon^1_D = \langle \langle \mathit{time} > 1\rangle \wedge \neg \mathit{a\text{-}d}_1 \wedge \text{T}, \{\neg\text{T}\}\rangle$ and $\varepsilon^5_D = \langle \langle \mathit{time} > 5\rangle \wedge \neg \mathit{a\text{-}d}_3 \wedge \text{T}, \{\neg\text{T}\}\rangle$. Note that, by construction of E_D, $\pi_{t=0}$ and $\pi_{t=10}$ do not generate any unnecessary events. In the first case, because, since no actions are performed when $t_s = 0 \leq \mathit{time} < 5$, time can flow without further preconditions, in the second case because time can not flow beyond $t_e = 10$.

Finally, we get that $P_{UD} = \{\rho'_1, \rho'_2, \rho'_{\mathit{time}}\}$ where $\rho'_1 = \langle \langle \mathit{time} < 10\rangle \wedge \langle x_1 > 0\rangle \wedge \text{T}, \{\langle x_2, 1\rangle\}\rangle$, $\rho'_2 = \langle \langle \mathit{time} < 10\rangle \wedge \langle x_2 > 0\rangle \wedge \text{T}, \{\langle x_1, x_2\rangle, \langle x_2, x_1^2\rangle\}\rangle$ and $\rho'_{\mathit{time}} = \langle \langle \mathit{time} < 10\rangle \wedge \text{T}, \{\langle \mathit{time}, 1\rangle\}\rangle$.

Optimising PDDL+ to PDDL2.1 Translation

All schemata proposed so far can be concatenated with the POLY translation from PDDL+ to PDDL2.1 presented by

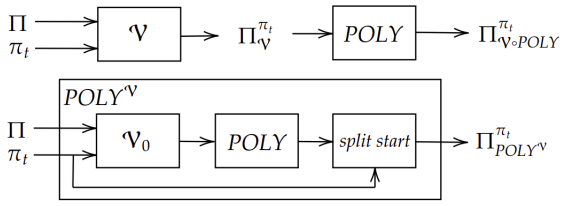


Figure 1: Workflow for producing a numeric validating task. The *upper* part describes the workflow for $\mathcal{V} \circ$ POLY with $\mathcal{V} \in \{\mathcal{V}_0, \mathcal{V}_{\text{U}}, \mathcal{V}_{\text{D}}, \mathcal{V}_{\text{UD}}\}$, while the *lower* part describes the workflow for POLY$^{\mathcal{V}}$. $\Pi_{\mathcal{V}}^{\pi_t}$ is a PDDL+ validating task while $\Pi_{\mathcal{V}\circ\text{POLY}}^{\pi_t}$ and $\Pi_{\text{POLY}^{\mathcal{V}}}^{\pi_t}$ are PDDL2.1 ones.

Percassi, Scala, and Vallati (2021). As POLY is sound and complete, exploiting Theorem 1, we can feed the validating PDDL+ encoding (obtained by any of the previous translations) to POLY, and use the resulting PDDL2.1 formulation to validate PDDL+ plans, too. That is, we can use a simpler numeric planner to check whether a PDDL+ plan is valid or not under discrete semantics by checking if the problem is solvable or not. However, both the optimisations presented, i.e., $\mathcal{V}_{\text{U}}$ and $\mathcal{V}_{\text{D}}$, are not effective when the resulting formulation is given to POLY. Indeed, the *start* action computed by POLY completely obscures the given optimisations. In fact, under $\mathcal{V}_{\text{U}}$, POLY makes useless the precondition added to the processes: the action *start* does not require that $\mathit{time} > t_e$. Similarly, under $\mathcal{V}_{\text{D}}$, POLY makes irrelevant the dead-end events. Every $\varepsilon \in E_{\text{D}}$, when triggered, makes the propositional variables T false; as above, *start* is not aware of T.

To overcome these limitations, we propose a variant of POLY specifically designed for supporting validation; we call such a translation POLY$^{\mathcal{V}}$. Such a variant is designed to be combined with $\mathcal{V}_0$ for producing a PDDL2.1 problem that mimics the optimisations in $\mathcal{V}_{\text{UD}}$. POLY$^{\mathcal{V}}$ is based on the encoding of POLY and extends it by taking the knowledge about the validation problem into account. POLY$^{\mathcal{V}}$ replaces the initialisation action *start* of POLY with a set of actions defined in such a way that one and only one of them can be performed in each sub-interval induced by π_t.

The set of start actions $\mathit{START}_{\mathcal{I}(\pi_t)}$ are defined as follows.

$$\mathit{START}_{\mathcal{I}(\pi_t)} = \bigcup_{\langle t'_i, t'_{i+1}\rangle \in \mathcal{I}(\pi_t)} \{\langle \mathit{pre}(\mathit{start}) \wedge \mathit{Done}(t'_i) \wedge \langle \mathit{time} \geq t'_i\rangle \wedge \langle \mathit{time} < t'_{i+1}\rangle, \mathit{eff}(\mathit{start})\rangle\}$$

$$\mathit{Done}(t'_i) = \begin{cases} \mathit{a\text{-}d}_{\mathit{last}(\pi(t'))} \text{ if } \pi(t') \neq \langle\rangle \\ \text{T otherwise} \end{cases}$$

$\mathit{START}_{\mathcal{I}(\pi_t)}$ encompasses a specific action *start* for each temporal sub-interval $\langle t'_i, t'_{i+1}\rangle \in \mathcal{I}(\pi_t)$; such an action can only be performed when (i) $t'_i \leq \mathit{time} < t'_{i+1}$, and (ii) when all the actions having the time-stamp equal to t'_i, i.e., those in $\pi(t'_i)$, have been performed. Let $\Pi = \langle F, X, I, G, A, \emptyset, P\rangle$ be an event-free PDDL+ problem, and let π_t a PDDL+ plan for Π, POLY$^{\mathcal{V}}$ produces a PDDL2.1 validating task $\Pi_{\text{POLY}^{\mathcal{V}}}^{\pi_t} = \langle F \cup D \cup \{\mathit{pause}, \text{T}\} \cup F_A, X \cup X^{\mathit{copy}} \cup \{\mathit{time}\}, I \cup \{\mathit{a\text{-}d}_0, \langle \mathit{time} := t_s\rangle, \text{T}\}, G \wedge \neg\mathit{pause} \wedge \mathit{a\text{-}d}_n \wedge \langle \mathit{time} = t_e\rangle \wedge \text{T}, A'_\pi \cup A_P \cup \mathit{START}_{\mathcal{I}(\pi_t)} \cup \{\mathit{end}\}, c\rangle$, where $A'_\pi = \{\langle \mathit{pre}(a) \wedge \neg\mathit{pause}, \mathit{eff}(a)\rangle,\ a \in A_\pi\}$ and A_π is defined as $\mathcal{V}_0$.

Figure 1 compares the workflow to generate a PDDL2.1 problem using the naive concatenation $\mathcal{V}\circ$POLY where $\mathcal{V} \in \{\mathcal{V}_0, \mathcal{V}_{\text{U}}, \mathcal{V}_{\text{D}}, \mathcal{V}_{\text{UD}}\}$ (top) with the validation sensitive translation POLY$^{\mathcal{V}}$ (bottom). As it is possible to observe, the main difference is that the validation sensitive translation takes in input the PDDL+ plan too, and uses this to split and precondition the simulation through the above mechanism.

Experimental Analysis

Our experimental analysis aims at corroborating the proposed approach for validating PDDL+ plans under discrete and continuous semantics. As benchmarks, we used the domains adopted by Percassi, Scala, and Vallati (2021) as they include a large variety of planning problems, and we extended their benchmark set with two non-linear domains: HVAC (Scala et al. 2016) and a modification of DESCENT (Piotrowski et al. 2016). The translator and the benchmark suite can be found a https://bit.ly/30gMyNW. For each benchmark domain, we collect between 10 and 20 instances. We generated plans for the discrete case using a planning engine while we used a semi-automated approach to generate plans for the continuous case. This process led to a grand total of 104 valid plans for the discrete semantics with $\delta = 1$. For the continuous semantics experiment, we collected 50 valid plans. To generate invalid plans for the considered domains, for both the continuous and the discrete setting, we modified each of the 104/50 plans by randomly performing one or more of the following modifications: (i) changing the order of the actions; (ii) changing the wait time between two subsequent actions, and (iii) changing actions' parameters. To challenge the validation process, we apply these modifications near the end of the plans.

We evaluate coverage as the number of instances that a planning engine proves valid/invalid, and CPU-Time as the time needed to provide an answer. Average CPU-time and expanded nodes are calculated by considering instances correctly dealt with by all the compared approaches. Experiments ran on an Intel i7-8565U CPU with 1.80GHz, up to 300 CPU-time seconds and 4 GB of RAM available.

Discrete Semantics Evaluation. We consider (i) uniform cost search, hereinafter denoted with BLIND, (ii) A* with the h^{aibr} heuristic (Scala et al. 2016), hereinafter denoted with A$^*(h^{aibr})$, and (iii) METRIC-FF (Hoffmann 2003) with default settings. BLIND is used on the formulation obtained with $\mathcal{V} \in \bigcup_{\mathcal{T}\in\{0,\text{U},\text{UD}\}}\{\mathcal{V}_\mathcal{T}, \mathcal{V}_\mathcal{T}\circ\text{POLY}\} \cup \{\text{POLY}^{\mathcal{V}}\}$, A$^*(h^{aibr})$ on those obtained with $\mathcal{V} \in \{\mathcal{V}_0, \mathcal{V}_{\text{U}}, \mathcal{V}_{\text{UD}}\}$ while METRIC-FF on those obtained with $\mathcal{V} \in \bigcup_{\mathcal{T}\in\{0,\text{U},\text{UD}\}}\{\mathcal{V}_\mathcal{T}\circ$ POLY$\} \cup \{\text{POLY}^{\mathcal{V}}\}$. That is, BLIND performed over PDDL+ and PDDL2.1 inputs, A$^*(h^{aibr})$ over PDDL+ translations while METRIC-FF over PDDL2.1 ones.

Table 1 provides domain-by-domain overview of the results obtained by BLIND (upper table) and by A$^*(h^{aibr})$ and METRIC-FF (lower). For BLIND, the table shows the average number of expanded nodes. Bold highlights the best performer. Considering BLIND, the $\mathcal{V}_0$ compilation is not ef-

		BLIND						
Metric V/I	Domain (n. instances)	PDDL+			PDDL2.1			
		$\mathcal{V}_0$	$\mathcal{V}_U$	$\mathcal{V}_{UD}$	$\mathcal{V}_0 \circ$ POLY	$\mathcal{V}_U \circ$ POLY	$\mathcal{V}_{UD} \circ$ POLY	POLY$^{\mathcal{V}}$
Expanded Nodes	LIN-CAR (10/10)	95/∞	67/66	**28/27**	241/∞	241/∞	306/∞	61/60
	OT-CAR (19/19)	878/∞	495/491	**76/73**	3663/∞	3663/∞	3700/∞	231/224
	LIN-GEN (10/10)	7534/∞	7507/7500	**1014/1013**	136485/∞	136485/∞	136485/∞	16009/15993
	UTC (7/7)	1417/∞	1041/1036	**104/99**	34852/∞	34852/∞	34852/∞	2425/2219
	BAXTER (17/17)	140/∞	86/77	**28/26**	5755/∞	5755/∞	5734/∞	821/759
	ROVER (5/5)	622/∞	609/603	**160.0/158.0**	4239/∞	4239/∞	4240/∞	1066/1059
	DESCENT† (20/20)	121/∞	108/101	**42/38**	918/∞	918/∞	925/∞	292/270
	HVAC† (16/16)	2791/∞	2151/2130	**151/147.1**	8466/∞	8466/∞	10597/∞	374/368
CPU-Time (seconds)	LIN-CAR (10/10)	2.2/∞	2.2/3.3	2.2/**2.4**	**2.1**/∞	**2.1**/∞	2.2/∞	2.2/2.9
	OT-CAR (19/19)	2.7/∞	2.8/2.5	2.7/**2.1**	**2.6**/∞	**2.6**/∞	**2.6**/∞	**2.6**/2.4
	LIN-GEN (10/10)	2.1/∞	2.2/2.4	**2.0/2.0**	3.1/∞	3.1/∞	3.1/∞	2.4/2.4
	UTC (7/7)	**2.3**/∞	**2.3/2.3**	**2.3/2.3**	3.1/∞	3.2/∞	3.1/∞	2.9/2.8
	BAXTER (17/17)	**2.0**/∞	**2.0**/2.1	**2.0/2.0**	2.2/∞	2.2/∞	2.2/∞	2.2/2.3
	ROVER (5/5)	**1.9**/∞	**1.9**/2.5	**1.9/1.9**	2.0/∞	2.0/∞	2.0/∞	2.0/2.3
	DESCENT† (20/20)	**1.8**/∞	**1.8**/1.9	**1.8/1.8**	1.9/∞	1.9/∞	1.9/∞	1.9/2.3
	HVAC† (16/16)	2.1/∞	2.1/2.3	**2.0/2.0**	**2.0**/∞	2.1/∞	2.1/∞	**2.0**/2.3
Coverage	LIN-CAR (10/10)	10/0	10/10	10/10	10/0	10/0	10/0	10/10
	OT-CAR (19/19)	19/0	19/19	19/19	19/0	19/0	19/0	19/19
	LIN-GEN (10/10)	10/0	10/10	10/10	10/0	10/0	10/0	10/10
	UTC (7/7)	7/0	7/7	7/7	7/0	7/0	7/0	7/7
	BAXTER (17/17)	17/0	17/17	17/17	17/0	17/0	17/0	17/17
	ROVER (5/5)	5/0	5/5	5/5	5/0	5/0	5/0	5/5
	DESCENT† (20/20)	20/0	20/20	20/20	20/0	20/0	20/0	20/20
	HVAC† (16/16)	16/0	16/16	16/16	16/0	16/0	16/0	16/16
	Σ	104/0	104/104	104/104	104/0	104/0	104/0	104/104
Planning Approach		A$^*(h^{aibr})$			METRIC-FF			
CPU-Time (seconds)	LIN-CAR (10/10)	2.4/2.5	2.5/2.6	2.5/2.5	**1.9**/2.0	2.0/1.9	2.1/**1.8**	**1.9**/1.9
	OT-CAR (19/19)	2.9/2.8	2.9/2.8	3.0/2.7	**1.9/1.8**	**1.9/1.8**	2.1/1.9	2.0/**1.8**
	LIN-GEN (10/10)	**5.0/4.0**	5.3/4.2	5.3/4.3	19.4/17.4	22.8/20.3	22.6/20.8	23.4/20.5
	UTC (7/7)	**2.5/2.5**	**2.5**/2.6	2.6/2.6	5.4/5.9	5.9/6.2	6.2/6.7	3.0/4.8
	BAXTER (17/17)	2.0/2.0	2.1/2.1	2.1/2.1	**1.5/1.5**	**1.5/1.5**	**1.5/1.5**	**1.5/1.5**
	ROVER (5/5)	2.5/2.7	2.5/2.7	2.7/2.8	2.5/**2.6**	2.4/2.8	∞/∞	**2.2/2.6**
	DESCENT† (20/20)	**2.5**/2.7	**2.5/2.5**	2.6/2.6	*na*/*na*	*na*/*na*	*na*/*na*	*na*/*na*
	HVAC† (16/16)	3.0/**2.6**	**2.9/2.6**	3.0/**2.6**	*na*/*na*	*na*/*na*	*na*/*na*	*na*/*na*
Coverage	LIN-CAR (10/10)	10/10	10/10	10/10	10/10	10/10	10/10	10/10
	OT-CAR (19/19)	19/19	19/19	19/19	19/19	19/19	19/19	19/19
	LIN-GEN (10/10)	10/10	10/10	10/10	10/10	10/10	10/10	10/10
	UTC (7/7)	7/7	7/7	7/7	7/7	7/7	7/7	7/7
	BAXTER (17/17)	17/17	17/17	17/17	17/17	17/17	17/17	17/17
	ROVER (5/5)	5/5	5/5	5/5	5/5	5/5	0/0	5/5
	DESCENT† (20/20)	20/20	20/20	20/20	*na*/*na*	*na*/*na*	*na*/*na*	*na*/*na*
	HVAC† (16/16)	16/16	16/16	16/16	*na*/*na*	*na*/*na*	*na*/*na*	*na*/*na*
	Σ	104/104	104/104	104/104	68/68	68/68	63/63	68/68

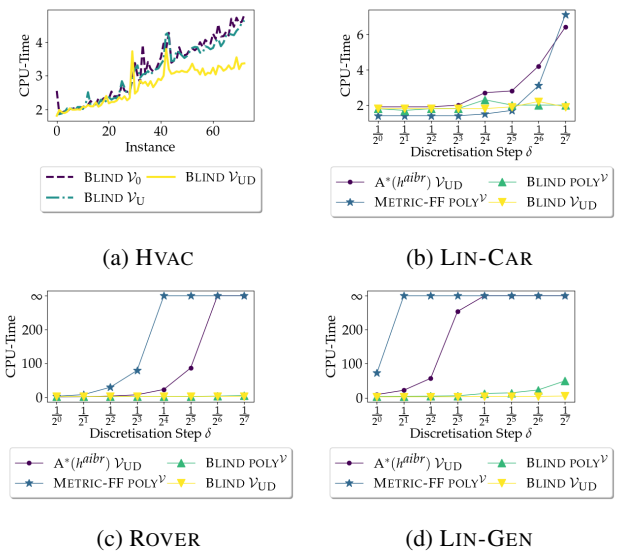
Table 1: Discrete validating results obtained by using BLIND, A$^*(h^{aibr})$ and METRIC-FF with the range of translations in place. ∞ indicates a timeout, while *na* indicates that the system does not support that domain. † denotes a non-linear domain.

fective in demonstrating that plans are not valid. This is due to the uninformed nature of the search, which does not allow the detection of many dead-ends. $\mathcal{V}_D$ and $\mathcal{V}_{UD}$ overcome such a limitation, bringing BLIND to demonstrate the invalidity of all the invalid plans. Similarly, with PDDL2.1, only the optimised POLY$^{\mathcal{V}}$ translation is able to solve all the instances. Concerning the average number of expanded nodes, the $\mathcal{V}_{UD}$ translation is the one that requires the minimum effort. The reduction of expansion w.r.t. the other translation is generally significant, often up to an order of magnitude, but not enough to get a substantial reduction of CPU-Time. This discrepancy is because the $\mathcal{V}_D$ translation leads to larger and more complex instances, and in such small instances such optimisation does not pay off that much. However, in domains like LINEAR-GENERATOR (LIN-GEN) where the difference between the number of expanded nodes is on average of several thousands of nodes, it is possible to notice that the use of $\mathcal{V}_D$ reduces the CPU-Time by approximately 10%. To better understand the behaviour of BLIND, Figure 2a evaluates the performance of different PDDL+ translations over

a set of increasingly larger instances on the HVAC domain. It is easy to notice that the longer are the plans to validate, the larger is the benefit in terms of CPU-Time. In other words, when uniformed search approaches are used, and the plans to validate are complex, it is better to use a more advanced translation. These benefits are less evident in the presence of heuristics effectively guiding the search, e.g., h^{aibr}, which reduces the impact of advanced translations.

Turning our attention to A$^*(h^{aibr})$ and METRIC-FF (bottom part of Table 1), it is interesting to observe that in most of the domains that METRIC-FF supports, the use of the corresponding translations can lead to the best CPU-Time. The notable exception is LIN-GEN where the significant size of the plans (thousands of actions) is challenging to deal with by METRIC-FF. In ROVER, the use of $\mathcal{V}_{\text{UD}} \circ$ POLY does not allow METRIC-FF to demonstrate the validity/invalidity of any of the plans: this is because that translation heavily relies on events – whose translation into conditional effects in PDDL2.1 can be challenging (Percassi, Scala, and Vallati 2021). On the contrary, the use of A$^*(h^{aibr})$ on the corresponding PDDL+ translations lead to CPU-Time values that consistently range between 1.8 and 2.8 seconds. By looking at our raw data, we observed that regardless of the considered translation, A$^*(h^{aibr})$ always expands the same number of nodes. Finally, we did a scalability experiment on LIN-CAR, ROVER and LIN-GEN. We took the largest instances and generate several validation problems with decreasing δ (from 1 to $1/2^7$), and verified that the given plan is valid for each such discretisation. As it is possible to observe in Figures 2b-2c-2d, BLIND has been, surprisingly, the configuration that scales better. Our intuition is that an increasingly huge number of expansions needed to solve the validity tasks, coupled with a very constrained search space make the use of heuristics and smart search enhancements detrimental. We also collect data for the other configurations (not displayed in the figures for clarity reasons), while the optimisations did not affect the behaviour of A$^*(h^{aibr})$, BLIND achieves such performance only when used in combination with $\mathcal{V}_{\text{UD}}$. For example, validating the plan with the smallest discretisation in LIN-GEN took about 15 seconds with BLIND and $\mathcal{V}_0$, while only 1 second with BLIND and $\mathcal{V}_{\text{UD}}$.

Continuous Semantics Evaluation. Table 2 shows the number of correct plans validated by systems adopting a continuous semantics, i.e., VAL and SMTPLAN, over the PDDL+ obtained by using $\mathcal{V} \in \{\mathcal{V}_0, \mathcal{V}_{\text{U}}, \mathcal{V}_{\text{UD}}\}$. We did not consider HVAC, DESCENT (not supported by the systems) and UTC (we could not generate any valid plan). VAL is faster and covers more plans than SMTPLAN as expected, but crashes in BAXTER, while SMTPLAN does not. VAL wrongly loops over a finite sequence of alternating events, likely given by some numeric precision error. The optimisations on validation through SMTPLAN gives mixed results. Finally, VAL always manages to prove the invalidity of plans, while SMTPLAN fails since it is an incomplete planning engine. To limit the exploration, and to guarantee the exhaustion of the search space if the plan considered is invalid, it would be necessary to provide to SMTPLAN a horizon of happenings. However, how to set this bound is not clear, especially since the validation problem under continuous semantics is undecidable (Fox and Long 2006).



(a) HVAC (b) LIN-CAR

(c) ROVER (d) LIN-GEN

Figure 2: (2a) CPU-Time for BLIND on HVAC (extended benchmark set). (2b-2c-2d) Scalability analysis over LIN-CAR, ROVER and LIN-GEN. ∞ indicates a timeout.

Metric	Domain (n. instances)	VAL	SMTPLAN		
			$\mathcal{V}_0$	$\mathcal{V}_{\text{U}}$	$\mathcal{V}_{\text{UD}}$
Coverage	LIN-CAR (10/10)	10	10	10	10
	OT-CAR (19/19)	16	0	0	0
	LIN-GEN (10/10)	10	2	2	2
	BAXTER (17/17)	*na*	9	9	9
	ROVER (5/5)	5	0	5	5
CPU-Time (seconds)	LIN-CAR (10/10)	0.01	2.7	2.7	3.0
	OT-CAR (19/19)	0.02	∞	∞	∞
	LIN-GEN (10/10)	0.01	2.7	2.5	2.4
	BAXTER (17/17)	*na*	3.5	4.3	4.3
	ROVER (5/5)	0.01	∞	2.6	2.6

Table 2: Continuous validating results obtained by using VAL and SMTPLAN over the range of translations in place. ∞ indicates a timeout; *na* an unsupported domain.

Conclusion

We presented a methodology for reformulating plan validation as planning, hence enabling the use of planning engines as validators. We propose a set of translations to generate a PDDL+ problem that has a solution iff the plan is valid w.r.t. the considered model. Such translations can be used both under continuous and discrete semantics. Further, starting from known PDDL+ to PDDL2.1 translations, we present an optimised variant allowing the use of PDDL2.1 engines to validate PDDL+ plans under discrete semantics. Our empirical analysis shows the suitability of the approach for both semantics. Future work will focus on extending our approach to different planning formalism, and on supporting PDDL+ knowledge engineering.

Acknowledgments

Francesco Percassi and Mauro Vallati were supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

Abdulaziz, M.; and Lammich, P. 2018. A Formally Verified Validator for Classical Planning Problems and Solutions. In *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018*, 474–479. IEEE.

Bensalem, S.; Havelund, K.; and Orlandini, A. 2014. Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer*, 16(1): 1–12.

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

Fox, M.; Howey, R.; and Long, D. 2005. Validating Plans in the Context of Processes and Exogenous Events. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, 1151–1156. AAAI Press/The MIT Press.

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

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

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6): 619–668.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, 44–53. AAAI.

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

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2004*, 294–301. IEEE Computer Society.

McCluskey, T. L.; and Porteous, J. M. 1997. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. *Artificial Intelligence*, 95(1): 1–65.

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

Penix, J.; Pecheur, C.; and Havelund, K. 1998. Using model checking to validate AI planner domain models. In *Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard*.

Percassi, F.; Scala, E.; and 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*, 252–261. AAAI Press.

Piotrowski, W. M.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic Planning for Hybrid Systems. In *Proceeding of the Thirtieth AAAI Conference on Artificial Intelligence*, 4254–4255. AAAI Press.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence, ECAI 2016*, volume 285, 655–663. IOS Press.

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