Exploiting Classical Planning Grounding in Hybrid PDDL+ Planning Engines

Enrico Scala
Department of Information Engineering
University of Brescia
Brescia, Italy
enrico.scala@unibs.it

Mauro Vallati
School of Computing and Engineering
University of Huddersfield
Huddersfield, United Kingdom
m.vallati@hud.ac.uk

***Abstract*—Hybrid PDDL+ models are amongst the most advanced models of systems and the resulting problems are notoriously difficult for planners to cope with due to non-linear behaviours and immense search spaces. This difficulty is exacerbated by the potentially huge size of the fully ground representations that are used by modern planners in order to effectively explore the search space, which can make some problems impossible to tackle, with the result that in several situations the grounding phase has to be done externally or manually. This not only produces a much less compact problem description, but also complicates debugging and model reuse.**

To overcome the aforementioned limit, in this paper we investigate two simple grounding techniques for PDDL+ problems. The former method we propose extends the simple mechanism of invariance analysis to limit the groundings of operators upfront. The latter proposes to tackle the grounding process by means of a PDDL+ to Classical Planning abstraction. A preliminary experimental analysis over benchmarks coming from real case study shows that not only the grounding can be sped up, but that also problems that were out of the reach before can now be efficiently solved in an automated manner.

***Index Terms*—Automated Planning, Hybrid PDDL+ Planning, Grounding**

I. Introduction

Automated planning is a prominent Artificial Intelligence challenge, as well as being a common capability requirement for intelligent autonomous agents. The development of domain-independent planners within the AI Planning community is leading to "off the shelf" technology that can be used in a wide range of applications: since they accept the domain and problem description in a standardised interface language and return plans using the same syntax, they can easily be exploited as embedded components within larger frameworks, as they can be interchanged without modifying the rest of the system.

The nature of real-world applications often necessitates the representation of the dynamics of the application in terms of mixed discrete / continuous effects, processes, exogenous events, and continuous activities, which can be encoded in a hybrid model of the domain. Following on from this, a dedicated language called PDDL+ [1] (Planning Domain Definition Language) was designed to support the compact encoding of such hybrid models.

Hybrid PDDL+ models are amongst the most advanced models of systems and the resulting problems are notoriously difficult for planners to cope with due to non-linear behaviours and immense search spaces. This difficulty is exacerbated by the potentially huge size of the fully ground representations that are needed by planners in order to explore the search space: this step alone can make some problems impossible to tackle.

Despite the importance and complexity of grounding for hybrid PDDL+ planning, there is a lack of studies focusing on investigating techniques and approaches for optimising this process. Most of the research on hybrid PDDL+ planning focuses on the design of complete domain-independent planning engines, such as DiNo [2], SMTPlan [3], and CASP [4]; and emphasis has been given to the search module of such engines. Relevant work in the area of efficiently PDDL+ grounding focused on reformulating the input models in order make them more amenable to planning engines [5], on modifying the internal behaviour of planning engines to adapt them to the specific application domain [6], [7] , rather than on designing principled grounding modules for domain-independent planning engines.

While grounding for PDDL+ has been scarcely investigated, a significant amount of work has been dedicated to designing efficient grounding approaches for classical planning. This is due to the decades of work devoted to research in the specific field and, in a significant part, to the International Planning Competition [8]: an efficient and effective grounding can dramatically boost the performance of a planning engine. One question naturally arises: *Can we leverage the extensive work on classical planning grounding to obtain efficient grounders for PDDL+ engines too?* To answer this question, in this paper we study two grounding techniques for PDDL+ problems. The first one exploits an extended static analysis that is aimed at identifying those elements that are necessary to be grounded in order to not lose any valid plan. The second method studies how to construct an abstracted, classical planning version of the PDDL+ and feed such a representation to off-the-shelf classical planning grounders. Both techniques have been designed in a modular fashion, to foster and support their integration into off-the-shelf planning engines in a way that any advancement in classical planning can be reflected in PDDL+ problems. Our experimental analysis, focused on challenging hybrid planning problems –also taken from real-

world applications– and on the state-of-the-art planning engine ENHSP [9], [10], highlights the importance of grounding in PDDL+, and validate the fact that it is indeed possible to leverage on classical planning techniques for obtaining efficient and effective PDDL+ grounders.

The remainder of this paper is organised as follows. First, in Section II we provide the necessary background. Then, in Section III, we introduce the two grounding approaches. The experimental analysis is provided in Section IV. Finally, conclusions are given.

II. Background

For the sake of space, we assume the reader is familiar with the basic concepts of automated planning. For an extensive introduction to the field, the interested reader is referred to Ghallab, Nau, and Traverso [11].

Our work focuses on the language of hybrid planning with autonomous processes, also known as PDDL+ (Planning Domain Definition Language) introduced by Fox and Long [1]. We will briefly report on the PDDL+ syntax and informally define its semantics.

A. Logical Foundations

PDDL+ is a language based on first-order logic that uses propositional, numeric predicates called fluents together with the standard connectives used in logic to postulate boolean and numeric formulas over them[1]. Propositional and numeric fluents are used to create relationship for list of objects and/or typed variables. Each such fluent is given a name through a label specifying the identifier of such a relation. For instance, the propositional fluent `(on A B)` can be used to specify the fact that the object A is on the object B; the numeric fluent `(distance C D)` specifies the distance between city C and D. Objects are finite and typed entities modelling basic elements of the world. Variables are devices by which one can represent generic fluents. A fluent is said to be ground if the associated list does not contain variables; unground otherwise. A propositional fluent can be true or false. A numeric fluent can instead take any value from $Q \cup \{\perp\}$.

A first-order formula over such fluents is defined recursively. Let ψ and ϕ be two first-order formulas:

- ψ being a propositional fluent is a formula
- $\langle \{\geq, >, =\}, \xi, 0\rangle$ where ξ is an arithmetical expression over some numeric fluents is a formula
- $\neg\psi$ is a formula
- $\psi \wedge \phi$ is a formula
- $\psi \vee \phi$ is a formula

We use these formulas with the purpose of constraining assignments to propositional or numeric predicates to satisfy some requirement. For instance we may want to say that `(on A B)` needs to hold in order for a goal to be reached, or some action applicable. A PDDL+ domain models those requirements that are general for a given domain.

[1]The logical pillars of the language resemble those used in Satisfiability Modulo Theory languages [12].

Definition 1 (PDDL+ Domain Model): A PDDL+ planning domain model is defined by the tuple $\langle T, C, F, X, A, E, P\rangle$:

- T (Types) is a set of types.
- C (Constants) is a set of typed objects, each of which is simply a name given to the object, and its type.
- F and X are sets of propositional and numeric fluents, respectively.
- A (Actions), E (Events), and P (Processes) are sets of transition schemata. A transition schema is the tuple $\langle \overline{\sigma}, pre, eff\rangle$ where:
  - $\overline{\sigma}$ is a sequence of objects from C or variables typed in T
  - pre is a first order formula .
  - eff is a set of Boolean and numeric effects. Boolean effects are assignments $\langle p, \{\top, \perp\}\rangle$ with $p \in F$ where numeric effects are assignments $\langle p, \xi\rangle$, with ξ being an arithmetical expression.
  - Both pre and eff only mention fluents from $\overline{\sigma}$ or objects from C.

A planning problem is defined combining a planning domain model D with a set of typed objects, an initial state and a goal. A planning problem asks whether, given a planning domain model, a set of objects, an initial state and a goal, there is a plan that lets the agent achieves the goal from such an initial state, considering the constraints imposed by D, and the actions that can be performed. More formally:

Definition 2 (PDDL+ Problem): Let D be a PDDL+ domain model; a planning problem is the tuple $\Pi : \langle D, O, I, G\rangle$ where:

- O is a set of typed objects.
- I is a function that assigns i) a truth value to all ground propositional fluents in F over compatible objects from $C \cup O$ ii) a rational value to all ground numeric fluents obtained substituting all numeric fluents in X over compatible objects from $C \cup O$.
- G is a first order formulas over ground propositional and numeric fluents.

For the sake of compactness, the initial state is usually specified in closed world assumption using the so called set-theoretic formulation [11]. That is, the initial state is a set of some subset of propositional fluents and a set of assignments for some subset of numeric fluents. Everything that does not belong to this assignment is assumed to be false (for propositional fluents) or undefined (for numeric fluents).

Having defined what a PDDL+ domain model and a problem are, we can now talk about ground transitions.

Definition 3 (Ground Transition and Grounding): A transition is ground if the parameters list only involves objects. Let Ω be a set of typed objects. The groundings of a transition schema a over Ω is denoted by $\sigma(a, \Omega)$ and corresponds to the set of all ground transitions obtained by substituting $\overline{\sigma}$ with a list of compatible objects taken from Ω, and then substituting each occurrence of the variables which were in $\overline{\sigma}$ with the newly introduced objects. Actions, processes and events are all transitions; therefore we will also talk about ground actions/processes/events when needed.

```
(:action switchPhase
    :parameters (?p - phase ?i - intersection)
    :precondition (and
            (controllable ?i)
            (activePhase ?p)
            (contains ?i ?p)
            (> (phaseTime ?i) (minPhaseTime ?p) ))
    :effect (and
            (trigger ?i)
))

(:event triggerCatcher
    :parameters (?p - phase ?i - intersection)
    :precondition (and
        (trigger ?i)
        (activePhase ?p)
        (contains ?i ?p))
    :effect (and
        (not (trigger ?i))
        (not (activePhase ?p))
        (activeIntergreenAfter ?p)
        (assign (phaseTime ?i) 0)
))

(:process flowrun_green
    :parameters (?p - phase ?r1 ?r2 - link)
    :precondition (and
            (activePhase ?p)
            (> (occupancy ?r1) 0.0)
            (> (turnrate ?p ?r1 ?r2) 0.0)
            (< (occupancy ?r2) (capacity ?r2)))
    :effect (and
            (increase
        (occupancy ?r2) (* #t (turnrate ?p ?r1 ?r2)))
            (decrease
        (occupancy ?r1) (* #t (turnrate ?p ?r1 ?r2)))
))
```

Fig. 1. An example of PDDL+ action, process, and event taken from the Urban Traffic Control domain [6], [13].

Hereinafter we subscript every conjunctive structure in the problem with B and N to isolate the components that talk only about propositional (B) or numeric (N) fluents. This gets applied to the structure of formulas and in the initial state. As we will see, this is useful for mapping numeric into classical planning problems.

Figure 1 shows an example of PDDL+ action, process, and event taken from the Urban Traffic Control domain [6], that will be considered in the experimental analysis. The action allows to model the decision of the planning engine to stop early a traffic light phase; the continuous effects of the movement of traffic is modelled by the process. The shown event is mainly for checking boundaries and constraints.

B. Semantics, Plans and Validity (Intuitively)

The semantic of a PDDL+ planning problem is defined using the theory of hybrid automata [14]. We will hereby report the basic semantics, the notion of a plan and its validity intuitively. We say that a formula is satisfied in a state if such a state is a model for the formula. A ground action is applicable in a state if its precondition formula evaluates to true on that state. Events and processes are said to be active in a state if their preconditions are satisfied. The application of an action in a state s instantaneously updates those numeric and propositional fluents which are modified by its effects. Active processes initiate flows of continuous changes for subsets of numeric variables. The numeric effect of a process is to

```
[...]
130.00: ( switchphase J6014-p2 J6014) [0.000]
130.00: ( switchphase J1349-p1 J1349) [0.000]
135.00: ( switchphase J1867-p2 J1867) [0.000]
140.00: ( switchphase J1353-p0 J1353) [0.000]
[...]
```

Fig. 2. An excerpt of a valid plan for a benchmark of the Urban Traffic Control domain.

be understood as the derivative of some variable x. Events trigger instantaneous changes on the state if their preconditions are satisfied. Actions are decisions that can be taken by the planning agent. Processes and Events are responses of the environment and cannot be controlled directly by the agent. A plan is a set of pairs $\langle t_i, a_i \rangle$ where $t_i \in Q$ and $a_i \in \sigma(A)$. A plan is valid if each action is applicable at the time associated with it. Plan validation corresponds to the task of evaluating whether each action precondition is satisfied in the trajectory of states induced by the active processes (that can change over time), the events that have been triggered, and the actions executed. A plan is a solution if the last state of the trajectory induced by all actions processes and events is a goal state, that is a state in which the goal formula is satisfied.

An excerpt of a valid plan for a benchmark from the Urban Traffic Control domain is provided in Figure 2. In this domain model, the only available action for the planning engine is to switch red the current traffic light phase. The action considers the current phase and the affected junction. For instance, the first line of the strategy shown below means that the currently active phase 2 of intersection 6014 has to be stopped after 130 seconds. Between each action the system evolves for the effect of the active processes and events modeling the flow of vehicles and the switching of the semaphores.

A number of approaches have been investigated for solving hybrid planning problems (e.g., [15]–[20]). All these approaches have been conceived to work over representations which are variable-less. However, it is much more convenient to express the problem using the full power of the language of PDDL+. For instance, looking at our example of Figure 1 it is much more convenient to say that there is a generic switch phase among two types of objects, and leave to the system the capability of understanding which of the ground transitions need to be generated in order to get to the goal. The challenge is to understand whether this can be done efficiently keeping the declarative representation and the desired semantics in check. In the next section we show two general methods aimed at automatising the process to bring a representation with variables into one which has no variable. We do so in a way that substantially departs form the naive algorithm of Definition 3. The second approach that we present, in particular, exploits an abstraction from PDDL+ to classical planning.

III. Domain-independent PDDL+ Grounding

This section is devoted to introduce two alternative routes for performing PDDL+ grounding.

A. Static Analysis Method

The first approach that we present is based on the idea of exploiting a static analysis of the domain model for focusing the generation of ground actions only towards a smaller set of parameters. Such a subset of parameters is restricted leveraging from the necessary condition arising by looking at the static conjuncts belonging to the transition schema precondition, and the hidden preconditions emerging from numeric effects that need to be applied.

This idea has been already exploited in classical planning with great success since the earlier work by Hoffmann and Nebel [21] in the FF planning system. As we will see in this section it is possible to straightforwardly extend this approach to the case of hybrid PDDL+ planning. This boils down at extending the static analysis to consider not only actions, but also processes and events as possible transitions that can change the value of some predicate and/or fluent. In a nutshell, the idea is to consider events and processes just as actions. Notice that this is a safe relaxation in that we give to the planning agent the possibility of directly controlling the environment. This obviously leads to enlarging the set of possible trajectories[2].

More formally, let Π be a PDDL+ planning domain model. We say that a propositional or a numeric fluent v is static iff $\forall t \in A \cup E \cup P.abstract(v,t) \cap affected(t) = \emptyset$ where

- $abstract(v,t)$ is the set of propositional or numeric fluents (depending on whether v is a propositional or numeric fluent) obtained by getting abstracted versions of such a fluent through some transition t. This abstraction amounts at substituting the variables in the parameters of v (if any) with compatible variables taken from the parameters of t. Note that there may be different substitutions also in this case, depending on which variables from the parameters list are taken.
- $affected(t)$ is the set of propositional or numeric fluents that are affected by the transition. That is $affected(t) = \{v_1 \mid \langle v_1, \xi\rangle \in eff(t)\} \cup \{v_1 \mid \langle v_1, \{\top, \bot\} \in eff(t)\rangle\}$

Intuitively, given a set of static propositional and numeric fluents, and an action t, the set of possible substitutions for the variables belonging to t parameters, i.e., $\overline{\sigma}$ (t) can be constrained looking at the necessary static precondition conjuncts that need to be true for that transition to be applicable, and looking at those numeric effects that need the evaluation of some numeric fluents in order to be evaluated. More precisely, we start from the universal substitution $Sub : V \rightarrow O \cup C$ that maps every variable to a set of objects. Then, we iterate over all identified static fluents and reduce the objects to be mapped only towards those that cannot be proved statically unreachable.

For instance, let us consider the example presented in Figure 1. The `(trigger ?i)` predicate is not a static predicate because its truth value may depend on whether the action `(switchPhase)` (with compatible parameters) is applied or not. Different is the situation for the `(turnrate ?p ?r1 ?r2)` numeric predicate. The value of this predicate cannot be changed by any action in the problem, so its numerical value solely depends on the initial state of the problem. This is indeed a static numeric predicate. In particular, if some grounding of this predicate is less or equal than 0, the parameters used for that grounding cannot be used in the `flowrun_green` parameters' list. Our static analysis method will not even try to ground the actions associated to those parameters for which `(turnrate ?p ?r1 ?r2)` leads to `(> (turnrate ?p ?r1 ?r2) 0.0)` being unsatisfied. This indeed reduces the number of groundings to only those combinations of parameters that do satisfy `(> (turnrate ?p ?r1 ?r2) 0.0)`; a brute force grounding will require the cartesian product of all objects compatible with variables `?p ?r1 ?r2`. Say you have 100 objects of each kind, this will require $1,000,000$ of groundings. As we will see in our real-world use cases, this situation is not rare, and does happen due to the inherently weaknesses of relational representation in several domains and problems from the International Planning Competition, too [8].

This method may reduce the number of mappings substantially, and therefore limits to some extent the combinatorial explosion of groundings caused by the cross-product of all universes of objects. Yet, it does not really exclude the groundings of some actions that could be easily detected as unreacheable. Let us come back to the example shown in Figure 1. Note that, although we do not know in general whether a specific `(trigger ?i)` will ever be satisfied, we do know that only some of them can eventually be reached. Those are the ones obtained by applying the action `swithcPhase` with some parameter, or by a dedicated additional process not shown in the provided example. Many of these actions and processes are indeed not reachable, and this can be easily detected by noticing that the predicate `(contains ?i ?p)` is itself a static predicate.

In order to fully exploit this intuition in a systematic fashion, next section shows how to make use of a classical planning abstraction, and therefore leverage from relaxed reachability grounding mechanisms present in state of the art classical planning engines.

B. Abstracting PDDL+ Problems into Classical Problems.

In this section we show how to leverage from grounding systems developed for classical planning. A classical planning problem differs from a PDDL+ problem in that: (i) classical actions are not timed and are instantaneous; plans are just sequences of partially ordered ground actions, (ii) there are no numeric fluents and therefore formulas cannot contain them, and (iii) there are no processes nor events that can change the state of the world autonomously; classical planning only models the planning agent's decisions. Everything that is not modified by the action effects remain unchanged (frame axiom).

[2]A similar relaxation schema has been implemented by the `AIBR` relaxation heuristic presented by [18].

For the sake of clarity, let us omit names of structures (e.g., actions, events) when obvious from the context and refer to propositional and numeric predicates using simply proposition and numeric. Let us furthermore denote with $\Pi : \langle T', C', O', F', A', I', G' \rangle$ a classical planning problem obtained by merging a domain model and an instance problem representation[3]. We denote with τ the abstraction from a PDDL+ problem to a classical planning problem. τ is formally a mapping from a PDDL+ problem $\Pi' : \langle T, C, F, X, A, E, P, O, I, G \rangle$ to the classical planning problem $\Pi' : \langle T', C', F', A', O', I', G' \rangle$ satisfying the following assertions:

- $T' = T$
- $C' = C$
- $O' = O$
- $I' = I'_B \cup \bigcup_{\langle f_i, k_i \rangle \in I_N} f_i$
- $A' = \bigcup_{t \in A \cup E \cup P} \langle pre_{N \to F}(t), eff_{N \to F}(t) \rangle$ where
  - $pre_{N \to F}(t) = abs(pre(t)) \wedge \bigwedge_{f_i \in \xi . \langle f_i, \xi \rangle \in eff_N(t)} f_i$
  - $eff_{N \to F}(t) = eff_B(t) \cup \bigcup_{f_i . \langle f_i, \xi \rangle \in eff_N(t)} f_i$
- $G' = abs(G)$
- $F' = F \cup X$

where $abs(\psi)$ is the abstraction of a general formula ψ given in negation normal form[4], defined as follows:

$$abs(\psi) = \begin{cases} \psi & \text{if } \psi \text{ is a proposition} \\ \bigwedge_{f_i \in \xi . \langle R, \xi, 0 \rangle \in \psi} f_i & \text{if } \psi \text{ is a numeric} \\ abs(\alpha) \wedge abs(\beta) & \text{if } \psi = \alpha \wedge \beta \\ abs(\alpha) \vee abs(\beta) & \text{if } \psi = \alpha \vee \beta \\ \neg abs(\alpha) & \text{if } \psi = \neg \alpha \end{cases} \quad (1)$$

Coming back to the example presented in Figure 1, take for instance the constraint `(< (occupancy ?r2) (capacity ?r2))`. This constraint involves two numeric predicates, i.e., `(occupancy ?r2)` `(capacity ?r2)`. These two predicates will be reinterpreted as two new fresh propositional predicates by Equation 1 with the same name. They will be made true, only if some other action, process or events assign them somehow.

Definition 4 (Reachable Ground Actions): The reachable ground actions for a planning problem Π is the set of ground actions that can be eventually reached by iteratively applying actions starting from the initial state up-to saturation.

Proposition 1 (Over-approximation of PDDL+ through τ): Let Π be a PDDL+ planning problem, the set of reachable ground actions, processes and events is a subset of the ground actions reachable in $\tau(\Pi)$ (with the proper transformation from action to process and events).

Proof 1 (Proof Sketch): We can prove this by observing that each atom in some formula of Π that is achievable implies that the same atom, or its abstraction in case we are dealing with numeric condition, is achievable in $\tau(\Pi)$ (note that the contrary is not true, i.e., there can be cases where an atom in the abstraction is reachable, but it is not in the concrete problem). If the set of atoms that is reachable is the same, we can safely observe that all those actions in $\tau(\Pi)$ that have their precondition reachable will become themselves reachable. This observation is trivial for propositional condition, and only a bit more involved for numeric conditions. For this latter case indeed, if some numeric condition is reachable in Π, this means that there is an action that can eventually satisfy it by interacting with some numeric variable in it. If we look at the abstraction operator of Equation 1, we observe that it suffices to have all numeric predicates evaluated in the numeric condition. And this is indeed the case if there is some action assigning it, or the initial state setting them to some value. Analogous is the consideration for the right hand side of all effects in the transitions.

Calculating the exact set of reachable ground actions is however unfeasible because it would require unrolling the complete transitions system, which, in the case of a PDDL+ problem may imply visiting an infinite set of states. However, thanks to the fact that we have a finite abstraction of the problem we can limit this worst case behaviour adopting approximations that are used in classical planning problems. As matter of facts, in order to overcome the problem of detecting all reachable actions, modern classical planners use relaxed reachability grounding, based on ideas borrowed by Answer Set Programming [22]. This gives us a superset of reachable actions in that the transition system is itself approximated with one where the validity of conditions grows monotonically. Thanks to Proposition 1 then, we know that the set of reachable actions for the abstraction of a PDDL+ problem is a superset of the actions reachable in classical planning. By transitivity, we can hence use the classical ground actions as a way to over-approximate the reachable grounding for all events, actions and processes for the concrete PDDL+ problem. Of course, once this approximation is done, we can perform reachability analysis on numeric problems (such as [10]) and refine the set of ground action even further. Yet, this refinement is already done on a smaller set (the one computed by the classical planning abstraction), so it is expected to be done much faster. To some extent, this method can be seen as a two level reachability analysis. The former is dealt with using classical planning. The latter using techniques that are aware of the numeric structure of the problem.

The abstraction based mechanism has the obvious advantage of capturing deeper causal dependencies between actions. However, it introduces a bit of an overhead. The system indeed needs to make use of an external classical planner, and there may be delays caused by encoding and decoding information

[3]A classical planning problem is an obvious subclass of PDDL+ problems. They will not include any numeric condition, any numeric assignment or update, and any process or event.

[4]This can be achieved by pushing negation down to atomic propositional term, and substituting negated numeric constraints with disjunctions.

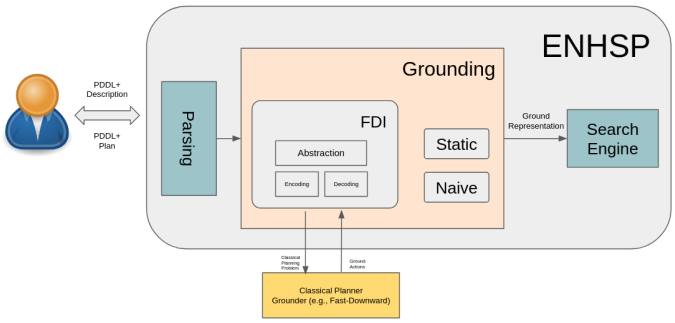

Fig. 3. Grounding and Search Data flow in ENHSP. FDI, $Static$ and $Naive$ represent the different grounding mechanisms implemented in ENHSP to evaluate our proposal. The module in yellow is the classical planning grounder, while the remaining modules are within the ENHSP planning system (parsing, grounding and search)

from and to the PDDL+ representation. Next section studies this aspect empirically.

IV. Experimental Analysis

This experimental analysis aims at evaluating the impact and the importance of the proposed domain-independent PDDL+ grounding techniques for solving hybrid planning instances.

A. Experimental Settings

For the sake of fairness, we implemented the considered grounding techniques as a modular component of the state-of-the-art planning engine ENHSP [9], [10]. ENHSP is a Java-based planning engine that includes a wide range of domain-independent search techniques and heuristics for solving PDDL+ planning instances, and is modular in nature. The results reported in this analysis have been obtained by running ENHSP with default parameters unless differently specified. Additional experiments using different search and heuristics settings, not presented in this paper for the sake of space, show very similar figures with regards to CPU-time; grounding remains of course unaffected. The classical planning abstraction of our PDDL+ problem is given to the Fast-Downward Grounding mechanism. Then, we collect all the ground actions and remap them in the original actions/processes/events they have been generated from. For the sake of clarity Figure 3 reports the overall process. The grounding module can choose from the three different grounders: $Naive$, $Static$, and FDI.

The $Naive$ grounder is our baseline: it naively grounds everything without any sort of pre-processing. The $Static$ grounder refers to the approach introduced in Section III-A, while FDI is the term used to indicate the approach described in Section III-B.

All the experiments were performed on an Intel i7-4750HQ CPU, 8 Gb of RAM and Linux operating system. A 15 CPU-time minutes cut-off time limit was enforced.

B. Considered Benchmarks

Traditional PDDL+ benchmarks include a very limited number of objects and a restricted number of predicates, operators, processes, and events. In fact, most of the existing benchmarks have been built as toy examples to test some specific aspects of the PDDL+ language. For this reason, following the approach exploited by Franco et al. [5], the experimental evaluation is performed by considering three benchmark domains, namely Rover, Urban Traffic Control, and Baxter. While Rover is an extension of an existing benchmark domain, the others refer to real-world application domains and are actually used to solve problems in the corresponding applications. In this analysis, we considered five instances per domain.

The well-known **Rovers** domain model introduced in IPC-3 [23], originally designed as a temporal domain model, has been extended in PDDL+ by modelling as continuous processes the movements of the rovers, and the energy generation via solar power. Each of the mentioned processes can be controlled by the planner using two actions, and is constrained, where appropriate, via events.

The **Urban Traffic Control** (UTC) domain has been originally introduced in [7]. It models the use of planning for generating traffic light signal plans, in order to de-congest an area of an urban region. In this analysis we considered the problems introduced by McCluskey and Vallati [6], which involved a network of 10 junctions, and we extended it by considering problems with 20 and 30 junctions, obtained by connecting identical regions. On benchmarks from this domain, ENHSP has been run with a delta ($-\mathrm{d}$) value of 50.0.

Finally, the **Baxter** domain, recently introduced by Bertolucci et al. [24], exploits planning for supporting robots in dealing with articulated objects manipulation tasks. The available domain model has been extended by adding events for preventing movements wider than 360 degrees. Problems consider articulated objects composed of between 5 and 15 links, and between 2 and 10 grippers.

C. Analysis of the results.

Table I compares the results achieved by ENHSP using the three considered grounding techniques. Results are presented in terms of grounding size, i.e. the sum of instantiated actions + processes + events, the CPU-time needed by the grounding process, and the overall CPU-time needed by the planning engine to solve the considered planning problem. The overall CPU-time includes all the steps needed by the planning engine, and therefore includes grounding, pre-processing, search, etc. The table also reports the averages obtained by the considered grounding techniques, in terms of ground size, ground runtime, and overall runtime. Averages are calculated by considering only instances for which all the 3 considered systems provided a value.

It comes as no surprise that $Naive$ is the technique that is consistently delivering the worst possible performance both in terms of grounding size, and in terms of time needed to solve the planning instance. When compared to the other grounders, $Naive$ can lead to a grounding that is orders of magnitude larger: this is then reflected in the much higher CPU-time needed to solve the planning instances. Despite the fact that such results are quite intuitive, this analysis provides clear evidence of the detrimental impact that an inefficient

TABLE I

RESULTS, IN TERMS OF GROUNDING SIZE, CPU-TIME NEEDED BY THE GROUNDING PROCESS, AND RUNTIME, ACHIEVED BY ENHSP WHEN USING THE THREE INTRODUCED GROUNDERS ON THE CONSIDERED SET OF BENCHMARKS. "–" INDICATES THAT THE GROUNDING PROCESS RUN OUT OF MEMORY. A RUNTIME VALUE OF 900.0 INDICATES TIMEOUT. AVERAGE RUNTIME (GROUNDING) IS CALCULATED BY CONSIDERING ONLY INSTANCES SOLVED (GROUND) BY ALL THE CONSIDERED APPROACHES. BOLD IS USED TO INDICATE BEST RESULTS WITH REGARDS TO THE CONSIDERED METRICS.

	Baxter								
	Grounding Size			Grounding Time			Overall Runtime		
	$Naive$	$Static$	FDI	$Naive$	$Static$	FDI	$Naive$	$Static$	FDI
1	16,425	2,796	**276**	0.6	0.6	0.6	251.5	70.3	**29.0**
2	17,632	2,864	**342**	0.7	0.6	0.6	349.1	89.7	**43.2**
3	8,100	1,457	**621**	0.6	**0.4**	0.6	900.0	900.0	**344.2**
4	5,225	1,004	**756**	0.6	**0.4**	0.6	900.0	900.0	900.0
5	65,700	6,640	**5,589**	0.8	0.6	0.6	900.0	900.0	900.0
Average	22,616	2,952	**1,517**	0.7	**0.5**	0.6	300.3	80.0	**36.1**

	Rovers								
	Grounding Size			Grounding Time			Overall Runtime		
	$Naive$	$Static$	FDI	$Naive$	$Static$	FDI	$Naive$	$Static$	FDI
1	304	87	**73**	0.5	0.4	0.4	1.3	1.3	1.5
2	42,526	1,065	**96**	0.5	0.6	0.5	11.5	5.6	**2.4**
3	472,216	3,255	**96**	2.3	0.6	0.6	47.5	13.2	**3.0**
4	472,216	3,255	**96**	7.4	0.8	**0.6**	48.9	14.1	**3.4**
5	946	219	**162**	0.5	0.5	0.5	900.0	900.0	900.0
Average	197,642	1,576	**105**	2.2	0.6	**0.5**	27.3	8.6	**2.6**

	Urban Traffic Control								
	Grounding Size			Grounding Time			Overall Runtime		
	$Naive$	$Static$	FDI	$Naive$	$Static$	FDI	$Naive$	$Static$	FDI
1	87,660	73,104	**292**	3.4	3.1	**0.8**	900.0	115.5	**1.7**
2	87,660	73,104	**292**	3.6	3.0	**0.8**	900.0	314.5	**2.1**
3	670,311	562,799	**584**	14.2	16.6	**1.0**	900.0	900.0	**3.9**
4	–	–	**876**	–	–	**1.3**	–	–	**61.0**
5	–	–	**876**	–	–	**1.3**	–	–	**72.3**
Average	281,877	236,336	**389**	7.1	7.6	**0.9**	–	–	–

grounder can have on the performance of an otherwise efficient planning system. In a sense, the $Naive$ results give a measure of how important grounding is for PDDL+. To the best of our knowledge, this is the first time that this aspect has been neatly assessed.

When comparing $Static$ and FDI, Table I clearly indicates that FDI is always able to produce a ground that is significantly smaller; at least an order of magnitude smaller. Notably, in instances from the UTC domain, the FDI grounding is between 2 and 3 orders of magnitude smaller than the corresponding $Static$ one. In the Baxter domain, the use of the FDI grounder allows the planning engine to deliver the best runtime performance. In other words, the reduced grounding size is positively affecting also the subsequent steps. When analysing the provided output, we observed that the lower runtime is not only due to the reduced time needed by actually generating the ground problem, but the smaller grounding size is having an impact also on the search steps, resulting in a larger area of the search space being explored. Similar results have been observed in the Rovers domain. In this domain, the $Static$ grounder can lead to runtime performance that are comparable to those achieved by using FDI only on the easiest instance.

The UTC domain provides the most challenging instances in terms of the potential size of the grounding. A large number of processes, events, and functions are used in this model. Noteworthy, in 2 instances of this domain the $Static$ grounder runs out of memory, while the FDI grounder is able to minimise the size of the instantiated problem, and allows the planning engine to find a solution with the cutoff time.

An interesting aspect to observe is that the impact of a grounding technique on the overall CPU-time of the planning engine is not only due to the time needed by the grounder itself. Looking at the results presented in Table I, it is clear that in two of the considered benchmark domains, the differences in CPU-time between the $Naive$, $Static$, and FDI grounders are negligible. In Baxter and Rovers, the main differences on the runtime of the planning engine is due to the fact that the provided ground is small and include the relevant actions needed to solve, and is therefore supporting the search module by effectively making the search easier. What happens in the UTC domain is slightly different: in this case the ground can be extremely large, and in fact significant differences can be observed also in the grounding times. These results

suggest that an effective grounder can positively affect the planning process in many ways, according to the structure of the benchmarks.

It is worth emphasising that grounder alone can determine whether or not the planning engine will be able to solve a given instance at all. This is particularly relevant in Baxter and UTC, that are PDDL+ models derived from real-world planning applications.

V. Conclusions

Hybrid PDDL+ models are needed to correctly and accurately represent the dynamics of real-world applications. PDDL+ models are amongst the most advanced symbolic planning models, and are notoriously difficult for planning engines to cope with. Complexity is exacerbated by the potentially huge size of the fully ground problems, that are needed by planning engines in order to explore the search space. Despite the importance of the grounding step for any domain-independent PDDL+ planning engine, there is a lack of work devoted to the specific topic.

In this paper we introduced two approaches for efficient and effective domain-independent PDDL+ grounding. In particular, we focused on investigating whether the vast amount of work done in the classical planning field could be exploited also for supporting PDDL+ grounding. The approaches have been developed in a modular fashion, and can be easily plugged into existing planning systems based on forward search. Our experimental analysis, that includes large benchmarks derived from real-world applications, showed that: (i) Regardless of the efficiency of the search approach exploited, the grounding step alone can become so critical that it may determine whether a planning instance can be solved or not; (ii) Grounding everything and hoping that the search component will efficiently navigate through the search space is the worst possible option, and (iii) it is indeed possible to fruitfully exploit grounding techniques that have been originally designed for classical planning.

We see several avenues for future work. First, we are interested in assessing whether a smart grounding approach can support the validation of complex scenarios, where the problems tackled are of significant size. Second, we plan to extend our methods by tailoring the grounding to some numeric aspects of the PDDL+ formalism. This may have the potential of further decreasing the number of ground actions at hand. Third, we envisage the exploitation of smart grounding techniques also in the context of knowledge engineering of PDDL+ models, in particular for providing support in terms of static and dynamic analysis and validation; this can be particularly interesting when a domain modeller is investigating different encoding of the problems, and the impacts of such encoding on the size of the problem.

References

[1] M. Fox and D. Long, "Modelling mixed discrete-continuous domains for planning," *J. Artif. Intell. Res.*, vol. 27, pp. 235–297, 2006.

[2] W. M. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio, "Heuristic planning for hybrid systems," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 4254–4255.

[3] M. Cashmore, D. Magazzeni, and P. Zehtabi, "Planning for hybrid systems via satisfiability modulo theories," *J. Artif. Intell. Res.*, vol. 67, pp. 235–283, 2020.

[4] M. Balduccini, D. Magazzeni, M. Maratea, and E. Leblanc, "CASP solutions for planning in hybrid domains," *Theory Pract. Log. Program.*, vol. 17, no. 4, pp. 591–633, 2017.

[5] S. Franco, M. Vallati, A. Lindsay, and T. L. McCluskey, "Improving planning performance in PDDL+ domains via automated predicate reformulation," in *Proceedings of the 19th International Conference Computational Science (ICCS)*, 2019, pp. 491–498.

[6] T. L. McCluskey and M. Vallati, "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.

[7] M. Vallati, D. Magazzeni, B. D. Schutter, L. Chrpa, and T. L. McCluskey, "Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 3188–3194.

[8] M. Vallati, L. Chrpa, and T. L. McCluskey, "What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask)," *Knowledge Eng. Review*, 2018.

[9] E. Scala, P. Haslum, S. Thiébaux, and M. Ramírez, "Interval-based relaxation for general numeric planning," in *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, 2016, pp. 655–663.

[10] E. Scala, P. Haslum, and S. Thiébaux, "Heuristics for numeric planning via subgoaling," in *IJCAI*. IJCAI/AAAI Press, 2016, pp. 3228–3234.

[11] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[12] C. W. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[13] G. Antoniou, S. Batsakis, J. Davies, A. Duke, T. L. McCluskey, E. Peytchev, I. Tachmazidis, and M. Vallati, "Enabling the use of a planning agent for urban traffic management via enriched and integrated urban data," *Transportation Research Part C: Emerging Technologies*, vol. 98, pp. 284 – 297, 2019.

[14] T. A. Henzinger, "The theory of hybrid automata," in *LICS*. IEEE Computer Society, 1996, pp. 278–292.

[15] D. V. McDermott, "Reasoning about autonomous processes in an estimated-regression planner," in *ICAPS*. AAAI, 2003, pp. 143–152.

[16] M. Cashmore, D. Magazzeni, and P. Zehtabi, "Planning for hybrid systems via satisfiability modulo theories," *J. Artif. Intell. Res.*, vol. 67, pp. 235–283, 2020.

[17] J. Shin and E. Davis, "Processes and continuous change in a sat-based planner," *Artif. Intell.*, vol. 166, no. 1-2, pp. 194–253, 2005.

[18] E. Scala, P. Haslum, S. Thiébaux, and M. Ramírez, "Interval-based relaxation for general numeric planning," in *ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. 285. IOS Press, 2016, pp. 655–663.

[19] D. Bryce, S. Gao, D. J. Musliner, and R. P. Goldman, "Smt-based nonlinear PDDL+ planning," in *AAAI*. AAAI Press, 2015, pp. 3247–3253.

[20] G. Della Penna, D. Magazzeni, F. Mercorio, and B. Intrigila, "Upmurphi: A tool for universal planning on PDDL+ problems," in *ICAPS*. AAAI, 2009.

[21] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, 2001.

[22] M. Helmert, "Concise finite-domain representations for PDDL planning tasks," *Artif. Intell.*, vol. 173, no. 5-6, pp. 503–535, 2009.

[23] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis," *Journal of Artificial Intelligence Research*, vol. 20, pp. 1–59, 2003.

[24] R. Bertolucci, A. Capitanelli, M. Maratea, F. Mastrogiovanni, and M. Vallati, "Automated planning encodings for the manipulation of articulated objects in 3d with gravity," in *Proceedings of the XVIIIth International Conference of the Italian Association for Artificial Intelligence (Ai*iA)*, 2019, pp. 135–150.