Towards Lifted Encodings for Numeric Planning in Essence Prime

Joan Espasa[1], Jordi Coll[2], Ian Miguel[1], and Mateu Villaret[2]

[1] School of Computer Science, University of St Andrews, St Andrews KY16 9SX, UK
{jea20,ijm}@st-andrews.ac.uk
[2] Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, E-17003 Girona, Spain
{jordi.coll,villaret}@imae.udg.edu

Abstract. State-space planning is the *de-facto* search method of the automated planning community. Planning problems are typically expressed in a high-level language called the Planning Domain Definition Language (PDDL), where action and variable templates describe the sets of actions and variables that occur in the problem. For many planners, one of the first steps is generating the full set of instantiations of these templates. In this way, the planners are able to derive useful heuristics that guide the search. Thanks to this success, there has been limited research in other directions.
In this work, we explore a different approach, where we keep the compact representation by directly reformulating the problem in PDDL into Essence Prime. In contrast with some heuristic planners, by using this reformulation procedure we are able to directly guarantee optimality on the solved instances in terms of makespan.
Our contribution revolves around two different encodings from PDDL to Essence Prime, how they represent action parameters and their performance. The encodings are able to maintain the compactness of the PDDL representation, and while they differ slightly, they perform quite differently on various instances from the International Planning Competition.

Keywords: Numeric Planning · Lifted Planning · Reformulation

1 Introduction

Given a model of the environment, a planning problem attempts to find a sequence of actions that leads from an initial state to a given goal state. These models are typically expressed in the Planning Domain Definition Language (PDDL). Typically, the user describes the problem in terms of predicates, actions and functions with parameters. In turn, these parameters can be instantiated with a set of defined objects.

Example 1. A simple example of a planning problem could be a logistics problem, where we must transport two persons from the city of Barcelona to the

airport, so they are finally able to embark on a plane that will take them home. Figure 1 shows a simple graphical representation of the problem. A valid plan for this problem would be to move Bob to the Airport, move Alice to the Airport, embark Bob into the plane, and finally embark Alice into the plane.

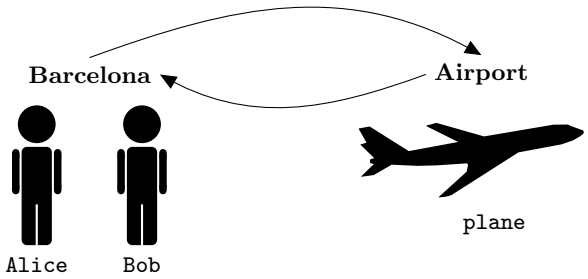

Fig. 1. A representation of the initial state. Persons are in the city of Barcelona, and they must reach the airport to embark.

In spite of having a compact model representation, when planners ingest the model, one of their first steps is producing a totally grounded representation. The action of grounding (or instantiation) will replace all variables that represent parameters in actions by their possible values, creating all the possible instantiations of the actions. After grounding, no variables are left free and all valid instantiations of predicates and functions in the actions are computed. The size of the fully grounded planning problem is exponential regarding the maximum number of arguments of all the actions in the original problem.

Example 2. An action `refuel(?x - vehicle, ?where - location)` is an action template. Considering three locations `L1`, `L2` and `L3`, and two vehicles `A` and `B` will result into six ground instances `refuel_A_L1`, ..., `refuel_B_L3` for each step considered.

Depending on the original problem and how the task is grounded, this growth sometimes can result in an instance that cannot be efficiently handled. There have been some approaches in the literature that try to alleviate this grounding problem in various ways. For example, one could ground only relevant parts of the problem [10,9], make clever representations of actions [1] or simplify the input problem [13].

The contrast of grounded planning would be lifted planning, where grounding is fully avoided. Grounding is normally seen as a necessary step, and there are very few approaches to lifted planning that skip grounding entirely [17,18,4]. These approaches are not that popular mainly due to the efficiency of grounding to easily compute informative heuristics that are difficult to compute at the lifted level. Also, reasoning in a more abstract level is typically more difficult.

In this work, our aim is to take advantage of the expressivity of the constraint satisfaction technologies to obtain lifted representations of PDDL problems. One of the successful approaches for solving combinatorial problems is by translation to more low-level languages, such as SAT, SMT [6], or FlatZinc. Examples of these tools might be Savile Row [16], MiniZinc [14] or Picat [21]. Ideally, such encodings would be compact (in terms of the number of clauses and additional variables) and would have good propagation properties.

In this work, we will use Savile Row. The interest of using it emanates from its declarative CSP modelling language Essence Prime and its capabilities of automatic domain pruning or common subexpression elimination [15]. Moreover, we will take advantage of its support for multiple backend solvers.

Our contributions in this paper are two different encodings from PDDL to Essence Prime. They differ by how they represent action parameters. The encodings are able to maintain the compactness of the PDDL representation, and while they differ slightly, they perform quite differently.

The rest of the paper proceeds as follows. In Section 2 we recall the theoretical framework. In Section 3 we propose the encodings from PDDL to Essence Prime. Section 4 is devoted to the experimental evaluation of the encodings. Finally, Section 5 discusses some future work and concludes.

2 Preliminaries on Automated Planning

This paper considers numeric planning problems, which extend propositional planning with numeric state variables. We formally define the numeric planning problem only in terms of the grounded representation of the problem.

Definition 1 (Numeric Planning Problem). *A planning problem can be defined as a tuple* $\prod = (B, O, F, X, A, I, G)$*, where*

- B *is a set of names for all the objects,*
- O *is a set of object state variables,*
- F *is a set of propositional state variables,*
- X *is a set of numeric state variables,*
- A *is a set of actions,*
- I *is the initial state and*
- G *is the goal.*

An action $a \in A$ *is defined as a tuple* $a = \langle Pre_a, Eff_a \rangle$*, where* Pre_a *refers to the precondition and* Eff_a *to the effects of the action.*

Definition 2 (State). *Given a planning problem* $\prod$*, a* state *is a variable-assignment (or valuation) function over state variables* $O \cup F \cup X$*, which maps each* $o \in O$ *to an object in* B*, each* $f \in F$ *into a truth value, and each* $x \in X$ *to an integer. A state is represented as a set of ordered pairs*

$$\{(v_1, z_1), (v_2, z_2), \ldots, (v_n, z_n)\}$$

where each v_i *is the variable and* z_i *the value mapped to it.*

An *object condition* has the form $\zeta \otimes b$, where ζ is an expression over O, $\otimes \in \{=, \neq\}$ and b is an object in B. A *numeric condition* has the form $\zeta \otimes k$, where ζ is a linear integer arithmetic expression over X, $\otimes \in \{\leq, <, =, >, \geq\}$ and k is an integer numeric constant.

Preconditions (Pre) and the goal G are sets (conjunctions) of object conditions, numeric conditions and propositions. Action effects (Eff) are sets of assignments to propositional variables, assignments to object variables and increase/decrease/assign a numeric variable by a numeric expression. A *conditional effect* is a pair $\langle c, e \rangle$ where c is a set of object, numeric and propositional conditions; and e is an effect. e is applied only if c is satisfied in the state where the action is applied.

An action a is *applicable* in a state s only if its preconditions are satisfied in s ($s \models Pre_a$) and the applied numeric, object and propositional effects do not induce conflicting assignments. The outcome after the application of an action a will be the state where variables that are assigned in Eff_a take their new value, and variables not referenced in Eff_a keep their current values.

A sequence of actions $\langle a_0, \ldots, a_{n-1} \rangle$ is called a *plan*. We say that the application of a plan starting from the initial state I bring the system to a state s_n. If each action is applicable in the state resulting from the application of the previous action and the final state satisfies the goal (i.e., $s_n \models G$), the sequence of actions is a *valid plan*. A planning problem has a solution if it can be found a valid plan for the problem.

3 Encodings

In this section, we propose various encodings for a numeric planning problem, as described in the previous section. First, we will explain how the planning as satisfiability approach works, then in what kind of input we will receive the planning problem and finally the proposed encodings.

3.1 Planning as Satisfiability

As it is typically done in the planning as SAT or CSP approaches [11,3], we will solve the planning problem by considering a sequence of CSPs ϕ_0, ϕ_1, ϕ_2, ..., where ϕ_i encodes the existence of a plan that allows to reach a goal state from the initial state in i steps. The solving procedure will test the satisfiability of ϕ_0, ϕ_1, ϕ_2, and so on, until a satisfiable formula ϕ_n is found, proving the existence of a valid plan of n steps.

Each ϕ formula will need variables to represent the state for each step and need to define the values of the variables in the initial step. Then, it will also need some variables to represent what action is executed at each step. We will need to make sure that if an action is executed, its precondition holds with respect to the problem variables. We will need to make sure that the goal conditions are met and we will do it by adding some constraints on the variables representing the state of the final step. Finally, we will need to make *frame axioms* explicit. That

is, constraints that specify that if no action has modified a variable, it keeps its value between steps. Approximations such as the ∀-step or ∃-step semantics [19] allow parallel actions, but for now we will assume that one action will be executed per time step.

3.2 Input

Notice that in contrast to the formal definition of a planning problem given in the previous section, PDDL allows us to specify problems in a lifted manner. Although being normally represented in this way, most solving approaches ground them.

```
(define (domain transport)
 (:types person aircraft - locatable
         location - object)
 (:predicates (at ?p - locatable ?l - location)
              (in ?p - person ?a - aircraft))
 (:functions (seats ?p - aircraft) - number)
 (:action move
    :parameters (?p - person ?from ?to - location)
    :precondition (at ?p ?from)
    :effect (and (not (at ?p ?from)) (at ?p ?to)))
 (:action embark
    :parameters (?p - person ?l - location ?a - aircraft)
    :precondition (and (at ?p ?l) (at ?a ?l) (> (seats ?a) 0))
    :effect (and (not (at ?p ?l)) (in ?p ?a) (decrease (seats ?a) 1))))

(define (problem example)
 (:domain transport)
 (:objects plane - aircraft
           Bob Alice - person
           Barcelona Airport - location)
 (:init (at Bob Barcelona) (at Alice Barcelona)
        (at plane Airport) (= (seats plane) 2))
 (:goal (and (in Bob plane) (in Alice plane))))
```

Fig. 2. Domain and problem file in PDDL, representing the example problem in Figure 1. A valid plan for the problem would be: `(move Bob Barcelona Airport)`, `(move Alice Barcelona Airport)`, `(embark Bob Airport plane)` and `(embark Alice Airport plane)`.

A *fluent*, in the area of automated planning, refers to a variable that represents some attribute of the problem and changes over time. Roughly speaking, our framework will be numeric planning, with the consideration of object fluents. This means that, apart from reasoning with integer fluents, we will be

able to have actions that work with objects and refer to attributes of these objects. Therefore, a fluent declared as `(location ?p - object) - place` will be able to express where objects are, and expressions like `(= (location plane) (location person))` or `(> (fuel plane) 10)` will be valid.

More concretely, our formalism will derive from PDDL 2.1 [7], without temporal semantics or metric optimizations. We will also consider the functional strips semantics [8] incorporated in the recent revisions of the PDDL language. As explained, this will enable us to refer directly to problem objects.

Even though planning formalisms do not consider templates, they are widely used in PDDL to make the representation compact. Types are also used in PDDL to make the problem more readable and to give more information to the planners. It can be seen in Figure 2 how types, templates for actions, predicates and functions are expressed. As our input will be a problem defined in the PDDL language, we will need to directly consider them. In fact, the instantiations of the predicate templates will correspond to the predicate state variables of the planning problem at hand, and the instantiations of the function templates will correspond to the object and numeric state variables of the planning problem, depending on its return type.

Templates can be *state variable templates* or *action templates*. These are comprised of a name and a sequence of typed parameters, or "ordinary" variables.

Example 3. Consider `(location ?p - object) - place`, being an object state variable template. Its name is `location` and its parameters, the sequence [`?p`], where the only parameter `?p` has the name `p` and the `object` type. The domain of this object state variable is the set of objects with type `place` in the problem.

For instance, in the PDDL specification, expressions such as preconditions and effects can also contain variables, belonging to the action template parameters. For example, the effect `(and (not (at ?p ?from)) (at ?p ?to))` belonging to the `move` action template in Figure 2 contains three variables: `p`, `from` and `to`.

3.3 Basic Encoding

In this section we describe formulas ϕ_h, that is, the existence of a valid plan with h actions.

Again, our purpose in this work is to encode these PDDL instances into Essence Prime in a lifted manner. Roughly, a grounded representation would have a Boolean variable stating whether action $move\_alice\_Barcelona\_airport^t$ is performed in a given time step t. Instead of that, for each time step t, we will have an integer variable stating which action template is applied and an integer variable per parameter of each action template stating what particular object is used as a parameter of that action. Moreover, and for each time step t, we will also have a CP variable for each concrete instantiation of each state variable template.

To express the encoding, we will need some auxiliary definitions. Let E be the set of types specified in the PDDL model. Each object $b \in B$ has a type associated with it. Also, each type $e \in E$ has a domain associated to it, being $Domain_e \subseteq B$. Let A_T be the set of action templates in the PDDL problem. Similarly, O_T, F_T and X_T will be the sets of object, propositional and numeric state variable templates, respectively.

Let V be the set $O \cup F \cup X$, representing the set of all state variables , without taking their type into account. V_T will represent the set of all state variable templates. For $x \in A_T \cup V_T$, let $Parameters_x$ be the sequence $[z_1, \ldots, z_n]$, representing the parameters of the template. For each parameter z_i, let $Type_{z_i}$ be the type associated to z_i, and $Name_{z_i}$ its name. Let $l(v) \to \mathbb{Z}$ be an injective function defined for all $v \in B \cup A$. It serves as a labelling function, that maps an object or action to a unique integer. This will be useful to later encode objects and object state variables as integers and integer state variables respectively.

We will start by introducing the following CP variables:

- We introduce variables $state_v^t$ that hold the value of state variable v in step t. This representation corresponds to a new CP variable for each grounded state variable.

$$state_v^t \qquad \forall v \in V, \forall t \in 0..h \tag{1}$$

- We introduce a variable $action^t$ to express which action is scheduled at time step t. The domain of these $action^t$ variables is $\{l(a) \mid a \in A_T\}$, being the set of integers the labelling function l assigns to the problem action templates.

$$action^t, \forall t \in 1..h \tag{2}$$

- We introduce new variables $param_{a,i}^t$ denoting the value of i-th parameter in action template a at each step t. Each of these variables will have a domain of the parameter type.

$$param_{a,i}^t \qquad \forall a \in A_T, \forall i \in Parameters_a, \forall t \in 1..h \tag{3}$$

Note that variables introduced in (2) and (3) correspond to the action templates. With this representation, there is no need to ground all the possible instantiations of the actions, and the solver will be responsible to choose what action template is executed and with which parameters.

We state initial and goal states

$$state_v^0 = z \qquad \forall (v, z) \in I \tag{4}$$

$$g^h \qquad \forall g \in G \tag{5}$$

where G is a conjunction of conditions on state variables, and g^h is the Essence Prime translation of these conditions on CP variables $state_v^h$ for all variables v in the conditions of G. Note that the initial state must be fully specified.

```
letting plane be 1
letting Bob be 2
...
letting aircraft be domain int(1)
letting person be domain int(2,3)
...
find actions: matrix indexed by [time] of domain_actions
find at: matrix indexed by [locatable,location,full_time] of bool
```

Fig. 3. An example for the encoding in Essence Prime of variables introduced in (1) and (2). We use of the `letting` keyword to define constants. The `find` keyword defines matrices of variables, and we use auxiliary domain definitions to make declarations more easily to read.

Frame axioms express that, given a state variable, if it has changed from a time step to the next one, is because an action that is able to change it has been executed.

$$state_v^{t-1} \neq state_v^t \rightarrow \bigvee_{\substack{\forall a \in A_T, \\ \forall m \in modify(a,v)}} \left(action^t = l(a) \ \wedge \bigwedge_{\forall (j,o) \in m} param_{a,j}^t = l(o) \right) \begin{array}{l} \forall t \in 1..h, \\ \forall v \in V \end{array} \tag{6}$$

We also need a function $modify(a, v)$ that given an action template a and a state variable v, returns the set of all combinations of parameter assignments (expressed as a pair (j, o)) that make action a modify variable v. For instance, the state variable `at(Bob,Barcelona)` is modified by action template `move`, with the following set of parameter assignments:

$$\{\{(\texttt{p}, \texttt{Bob}), (\texttt{from}, \texttt{Barcelona}), (\texttt{to}, \texttt{airport})\}, \\ \{(\texttt{p}, \texttt{Bob}), (\texttt{from}, \texttt{airport}), (\texttt{to}, \texttt{Barcelona})\}, ...\}$$

Figure 4 is an example of a frame axiom.

Finally, actions are expressed

$$action^t = l(a) \rightarrow Pre_a^t \wedge Eff_a^t \qquad \forall a \in A_T, \forall t \in 1..h \tag{7}$$

Preconditions are sets of conditions and effects are sets of assignments. When translating Pre_a^t and Eff_a^t into Essence Prime, we use the *element* global constraint to access the corresponding state variables according to the values given to the action parameters. The translation of conditions and state variable assignments to Essence Prime is straightforward. However, we remark that conditions and right-hand sides of assignments will consult the state variables of time $t - 1$, and left-hand side of the assignments will update state variables of time t. For instance, when considering the effect on the number of free seats in the `embark` action: `seats[embark_a[k],k] = seats[embark_a[k],k-1]-1`.

```
forAll k : int(1..T) .
 forAll p : person .
  forAll a : aircraft .
    in[p, a, k-1] != in[p, a, k]) ->
      actions[k] = embark /\ embark_p[k] = p /\ embark_a[k] = a
```

Fig. 4. Encoding in Essence Prime of Equation (6) for the *in* fluent in the example from Figure 2. Informally, if the *in(p,a)* fluent changes between time steps, it's because an embark action has been executed and its parameters correspond to p and a.

3.4 Encoding Compaction

Approximations such as the ∀-step or ∃-step semantics [19] allow parallel actions as long as they are not interfering. For now, our encoding assumes that one action will be executed per time step.

With one action executed per time step, we can see that most of the variables from (3) are rarely used. That is, only the parameters belonging to the selected action are used, and the others are ignored. Here we introduce two variants of the encoding with the aim of reducing the total number of variables: *Type sharing* and *Max Parameters*.

Before explaining the encodings, we need to introduce the concept of a substitution. A *substitution* (or *renaming*) σ is a partial mapping from variables to variables. It can be represented explicitly as a function by a set of bindings of variables to variables. That is, if $\sigma = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$, then $\sigma(x_i) = y_i$ for all i in $1..n$, and $\sigma(x) = x$ for every other variable. Using an infix notation and given any expression τ containing variables, $\tau\sigma$ is τ with all the contained variables replaced, as specified by σ.

Example 4. Given a substitution $\sigma = \{p \mapsto q\}$ and the term representing an effect $\tau =$ `(and (not (at ?p ?from)) (at ?p ?to))`, the result of $\tau\sigma$ would be `(and (not (at ?q ?from)) (at ?q ?to))`

Type Sharing The rationale behind this encoding is that, even though actions can have lots of parameters, they will have very few parameters of the same type. Therefore, actions will now substitute each of its parameters of a given type for a new parameter that will be shared by all the actions that need a parameter of that type.

Let C_e for each $e \in E$ be the maximum number of parameters on all actions that share type e. Then, variables introduced in (3) are substituted with

$$param^t_{e,i} \qquad \forall e \in E, \forall i \in 1..C_e, \forall t \in 1..h \tag{8}$$

Example 5. If the PDDL action that has most parameters with the `place` type is an action such as `move(?p - person, ?from - place, ?to - place)`, then $C_{place} = 2$. Then, the previous Equation will introduce parameter variables $param^t_{person,1}$, $param^t_{place,1}$ and $param^t_{place,2}$ for each time step.

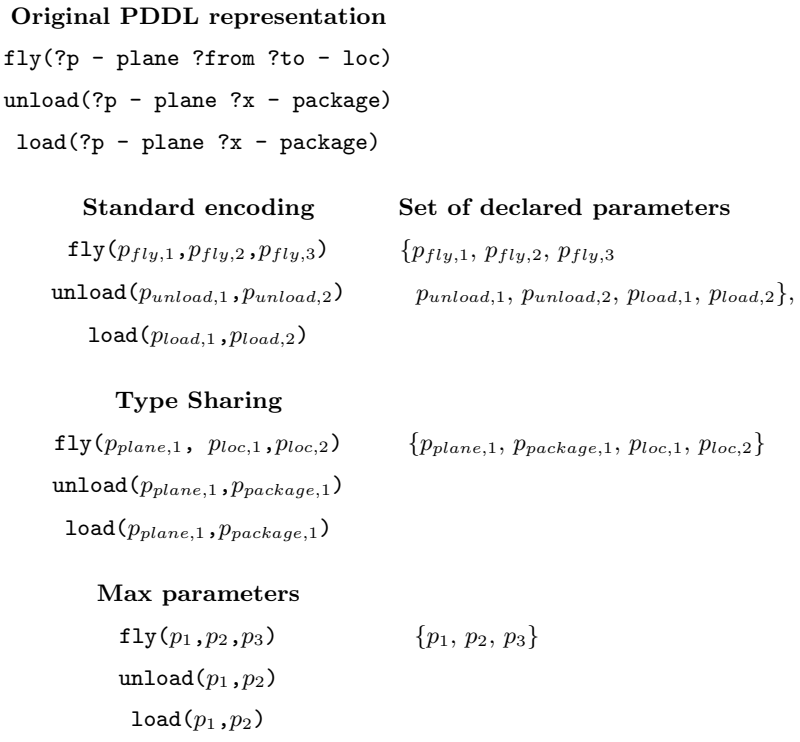

Fig. 5. A graphical representation of how parameters are shared in the various encodings for the planes domain used in the experimental section. The left column shows how parameters are substituted, and the right column what parameter variables are created.

Given an action template $a \in A_T$, a parameter $q \in Parameters_a$ and its type $Type_q \in E$, let $pos(q, a) = [z \mid z \in Parameters_a, Type_z = Type_q]$. That is, the subsequence of parameters of a that have the same type as q.

Then, we can define a substitution σ_a for every action $a \in A_T$, such that

$$\sigma_a = \{param_{a,q} \mapsto param_{e,i} \mid q \in Parameters_a, e = Type_q, i \in 1..|pos(q,a)| \ , pos(q,a)[i] = q\} \quad (9)$$

Finally, to Equation (7) is modified to use these new parameter variables

$$action^t = l(a) \rightarrow Pre_a^t\sigma_a \wedge Eff_a^t\sigma_a \qquad \forall a \in A_T, \forall t \in 1..h \quad (10)$$

Following Example 5, this will substitute all appearances on the *Pre* and *Eff* of ?*p* by $param_{person,1}$ and so on. Figure 6 represents the encoding in Essence Prime with the substitutions applied.

```
forAll k : int(1..MAX_ACTIONS) .
 actions[k] = embark ->
 ((at[p_person1[k], p_location1[k], k-1] /\
   at[p_aircraft1[k], p_location1[k], k-1] /\
   seats[p_aircraft1[k], k-1] > 0)
 /\
 (!at[p_person1[k], p_location1[k], k] /\
  in[p_person1[k], p_aircraft1[k], k] /\
  (seats[p_aircraft1[k], k] = seats[p_aircraft1[k], k-1] - 1)),
```

Fig. 6. Encoding in Essence Prime of Equation (10) for the *embark* action from Figure 2. Note that in the *Pre* and *Eff* we substituted the parameter names for the ones according by σ_a.

Max parameters Another approximation for representing parameters more efficiently is to share parameters independently of its types. That is, instead of dedicated parameter variables for each action, we will only declare n parameters, where n is equal to the number of parameters of the action with most parameters. Formally, $n = max(\{|Parameters_a| \mid a \in A_T\})$. These parameters will be representing different types depending on which action is executed. Therefore, the domain of each one will be the union of all possible objects. We will again substitute variables in (3) by

$$param_q^t \qquad \forall q \in 1..n, \forall t \in 1..h \tag{11}$$

Now, let σ_a be a substitution for every action $a \in A_T$, such that

$$\sigma_a = \{param_{a,q} \mapsto param_q | q \in Parameters_a\}$$

This substitution will replace the mentioned parameters in the action by the new declared parameters in (11). Finally, Equation (7) is also modified to use these new variables

$$action^t = l(a) \rightarrow Pre_a^t \sigma_a \wedge Eff_a^t \sigma_a \qquad \forall a \in A_T, \forall t \in 1..h \tag{12}$$

To help the encoding, if using a CSP solver as a backend, a table constraint can be added on the Essence Prime model to limit the possible values of the parameters depending on the action chosen. This makes that once an action has been decided, the domains of the parameters are restricted to its declared types.

4 Experimental Evaluation

In this section, we evaluate the performance of the presented encodings by solving a set of numeric planning problems coming from the third IPC [12]. These domains contain integer numeric fluents without quantified preconditions, as the rest of the domains contain features that we still do not support. These domains

are: *Zenotravel*, *Driverlog*, *Depots*. This competition edition has been chosen as it was the last that focused on problems with a numeric component. The *Petrobras* and *Planes* domains from [5] are also considered since they have an interesting numerical component. Although some domains give various optimization criteria, we only consider the satisfiability of the problem, minimizing the makespan.

As we said, our approach reformulates the PDDL description to the Essence Prime language. In turn, this Essence Prime model is given as input to Savile Row [16] to generate a SAT or CSP model. Finally, we use Glucose [2] and Chuffed[3] as the backend solvers. We validated the usefulness of the Savile Row preprocessing steps such as common subexpression elimination or symmetry breaking capabilities by turning them off and determining that solving times were gravely hindered, at least by a factor of two.

To compare the presented encodings with a similar approach, we use the linear encoding provided by RanTanPlan [5], using planning as SMT with a fully grounded encoding. We additionally evaluated the SpringRoll planner [20], as it also supports linear semantics, but it only solved 5 instances and therefore it is not included in the results table. The experiments were run on a cluster of machines, running the CentOS operating system, equipped with Intel® Xeon® E3-1220v2 Processors at 3.10 GHz with Turbo Boost disabled, and 8GB of main memory. The total timeout is set to 1 hour.

We do not consider the basic encoding without compacting action parameters, as it behaves worse than the two proposed improvements. Table 1 shows the time taken to solve and the optimal makespan for instances solved by any of the approaches considered. The three approaches are comparable in terms of total number of instances solved, but differ considerably between families.

The maximum number of actions we see is 24, and we speculate that over these numbers symmetries between the application of actions are too big for the solvers to be able to give a response in the given timeout.

The *Depots* domain seems too big, as all the approaches are only able to return a solution for a very few number of instances. The same happens with the *Petrobras* domain.

If we look at the *Driverlog*, *Zenotravel* and *Planes* domains, the different approaches differ between them. The RanTanPlan planner is clearly better in the *Zenotravel* domain. The Type Sharing encoding is roughly one order of magnitude faster than the rest in the *Driverlog* domain and is clearly better in the *Planes*. Finally, the Max. Parameters encoding is comparable in number of instances to RanTanPlan in the *Zenotravel*.

Chuffed is able to solve 8 instances of *Petrobras*, and seems to benefit greatly from the type information in the Type Sharing encoding. Strangely, we find lots of out of memory errors when using the Max. Parameters encoding with Chuffed. Even though typically the Max. Parameters encoding generate a few less parameters, the type information is very useful, as Savile Row is able to generate smaller encodings when translating to SAT.

[3] https://github.com/chuffed/chuffed

Table 1. Solved instances by any of the approaches. SR - SAT columns are the combination of Savile Row and Glucose, and SR - CP are Savile Row with Chuffed. TO denote a timeout and MO a memory out.

Instance	steps	SMT RanTanPlan	SR - SAT T. Sharing	SR - SAT Max. Param	SR - CP T. Sharing	SR - CP Max. Param
depots-1	10	28.21	111.80	119.34	27.89	2889.01
depots-2	15	TO	437.56	523.08	2653.75	TO
depots-7	21	TO	3513.73	TO	TO	TO
driverlog-1	7	0.90	5.75	23.21	5.30	MO
driverlog-2	19	1298.04	39.44	439.14	1417.61	MO
driverlog-3	12	39.76	15.30	87.48	26.00	MO
driverlog-4	16	1877.58	29.17	292.47	1358.93	MO
driverlog-5	18	3535.26	42.63	594.14	1509.25	MO
driverlog-6	11	168.82	14.66	100.39	31.51	MO
driverlog-7	13	3228.56	20.77	174.69	363.30	MO
driverlog-9	22	TO	132.74	2459.13	TO	MO
driverlog-10	17	TO	130.22	2259.11	TO	MO
driverlog-11	19	TO	105.48	1711.46	TO	MO
petrobras-A1	5	39.29	204.92	145.53	6.74	MO
petrobras-A2	10	1444.23	TO	TO	47.93	MO
petrobras-B1	5	498.21	281.91	435.94	8.85	MO
petrobras-B2	10	TO	TO	TO	404.23	MO
petrobras-C1	5	496.99	277.70	403.17	8.74	MO
petrobras-C2	10	TO	TO	TO	406.26	MO
petrobras-D1	5	496.09	267.19	416.62	9.69	MO
petrobras-D2	10	TO	TO	TO	405.96	MO
planes-1	14	10.90	63.73	TO	25.37	MO
planes-2	17	87.90	86.76	TO	135.69	MO
planes-3	19	703.04	321.03	TO	1509.96	MO
planes-4	22	TO	1318.36	TO	TO	MO
planes-5	21	TO	491.70	TO	TO	MO
planes-6	24	TO	2145.01	TO	TO	MO
planes-7	22	TO	612.99	TO	TO	MO
planes-8	23	TO	1031.65	TO	TO	MO
zenotravel-1	1	0.07	0.78	2.29	0.51	3.75
zenotravel-2	6	0.13	4.45	69.73	4.27	56.92
zenotravel-3	7	0.67	105.07	250.62	6.67	130.07
zenotravel-4	10	2.69	336.9	580.81	14.37	593.98
zenotravel-5	12	69.72	TO	1072.97	52.58	TO
zenotravel-6	12	66.32	TO	1235.67	106.58	TO
zenotravel-7	13	191.70	TO	1679.50	114.21	TO
zenotravel-8	13	977.26	TO	TO	TO	TO
Total Solved	-	24	29	23	27	5

5 Conclusions and Future Work

In this work, we have presented two lifted encoding approaches of planning problems to CP, which seem promising as a first step. The use of Savile Row has enabled us to try different solving backends easily. Its interesting to see that small changes in how the parameters are shared lead to important differences in solving times. The difference between efficiency between the two encodings depends on the differences between the number of parameters in actions and its types. This could lead to a simple preprocess where an encoding is selected depending on the problem structure.

These encodings can also be helped by considering the symmetries between the successive application of different actions, or by incorporating the application of various actions in the same step. The search strategy now goes naively from one step to infinity, until a plan is found or the timeout makes the procedure stop. A possible improvement is to consider computing a lower bound using a relaxation of the problem.

For now we have considered a few similar numeric domains from the IPC competition. We should extend the comparison further, with more domains and get a full grasp of the strengths and weaknesses of the encodings. Non-numeric domains would be also interesting to try. The comparison should also be extended to include numeric planners with different approaches.

Acknowledgements

This work has been partially supported by grant MPCUdG2016/055 (UdG), grant TIN2015-66293-R (MINECO/FEDER, UE), grant RTI2018-095609-B-I00 and grant EP/P015638/1 (EPSRC, UK).

References

1. Areces, C., Bustos, F., Dominguez, M.A., Hoffmann, J.: Optimizing Planning Domains by Automatic Action Schema Splitting. In: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014 (2014)
2. Audemard, G., Simon, L.: Predicting Learnt Clauses Quality in Modern SAT Solvers. In: IJCAI. pp. 399–404 (2009)
3. van Beek, P., Chen, X.: CPlan: A Constraint Programming Approach to Planning. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA. pp. 585–590 (1999)
4. Bofill, M., Espasa, J., Villaret, M.: Efficient SMT Encodings for the Petrobras Domain. In: Proceedings of the 13th International Workshop on Constraint Modelling and Reformulation (ModRef 2014). pp. 68–84. Lyon, France (2014)
5. Bofill, M., Espasa, J., Villaret, M.: The RANTANPLAN planner: system description. The Knowledge Engineering Review (KER) **31**(5), 452–464 (2016). https://doi.org/10.1017/S0269888916000229

6. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with sat modulo theories. Constraints **17**(3), 273–303 (2012)
7. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research (JAIR) **20**, 61–124 (2003)
8. Geffner, H.: Functional STRIPS: a more flexible language for planning and problem solving. In: Logic-based artificial intelligence, pp. 187–209. Springer (2000)
9. Gnad, D., Torralba, A., Domınguez, M., Areces, C., Bustos, F.: Learning How to Ground a Plan–Partial Grounding in Classical Planning. In: Proceedings of the thirty-Third AAAI Conference on Artificial Intelligence AAAI January 27 - February 1, 2019, Honolulu, Hawaii, USA (2019)
10. Helmert, M.: Concise finite-domain representations for PDDL planning tasks. Artificial Intelligence **173**(5-6), 503–535 (2009)
11. Kautz, H.A., Selman, B.: Planning as Satisfiability. In: European Conference on Artificial Intelligence (ECAI). pp. 359–363 (1992)
12. Long, D., Fox, M.: The 3rd International Planning Competition: Results and Analysis. Journal of Artificial Intelligence Research (JAIR) **20**, 1–59 (2003). https://doi.org/10.1613/jair.1240
13. Masoumi, A., Antoniazzi, M., Soutchanski, M.: Modeling Organic Chemistry and Planning Organic Synthesis. In: Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015. pp. 176–195 (2015)
14. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a Standard CP Modelling Language. In: Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings. pp. 529–543 (2007)
15. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I.: Automatically Improving Constraint Models in Savile Row through Associative-Commutative Common Subexpression Elimination. In: Principles and Practice of Constraint Programming. pp. 590–605 (2014)
16. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Spracklen, P.: Automatically improving constraint models in Savile Row. Artificial Intelligence **251**, 35–61 (2017). https://doi.org/10.1016/j.artint.2017.07.001
17. Penberthy, J.S., Weld, D.S.: UCPOP: A Sound, Complete, Partial Order Planner for ADL. In: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992. pp. 103–114 (1992)
18. Ridder, B., Fox, M.: Heuristic Evaluation Based on Lifted Relaxed Planning Graphs. In: Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014 (2014)
19. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search. Artificial Intelligence **170**(12-13), 1031–1080 (2006)
20. Scala, E., Ramírez, M., Haslum, P., Thiébaux, S.: Numeric planning with disjunctive global constraints via SMT. In: Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016. pp. 276–284 (2016)
21. Zhou, N., Kjellerstrand, H.: The Picat-SAT Compiler. In: Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings. pp. 48–62 (2016)