Relaxed ∃-Step Plans in Planning as SMT

Miquel Bofill, Joan Espasa, Mateu Villaret
University of Girona, Spain.
{miquel.bofill, joan.espasa, mateu.villaret}@udg.edu

Abstract

Planning Modulo Theories (PMT), inspired by Satisfiability Modulo Theories (SMT), allows the integration of arbitrary first order theories, such as linear arithmetic, with propositional planning. Under this setting, planning as SAT is generalized to planning as SMT. In this paper we introduce a new encoding for planning as SMT, which adheres to the relaxed relaxed ∃-step (R^2∃-step) semantics for parallel plans. We show the benefits of relaxing the requirements on the set of actions eligible to be executed at the same time, even though many redundant actions can be introduced. We also show how, by a MaxSMT based post-processing step, redundant actions can be efficiently removed, and provide experimental results showing the benefits of this approach.

1 Introduction

The possibility of several actions being planned at the same time step, i.e., the notion of parallel plans, is crucial for *planning as satisfiability* approaches [Rintanen *et al.*, 2006], in which the feasibility of a plan in a given number of time steps is encoded as a SAT formula; the greater the number of time steps, the bigger the formula. Hence, parallel plans may help improve the performance of the planner because they enable the reduction of the time horizon.

Pioneering work on parallel plans [Kautz and Selman, 1996] was based on the so-called ∀-*step semantics*, which requires that parallel actions can be executed sequentially in any order, and the same terminal state is reached. Therefore, parallel application of actions is allowed only if they do not interfere in any way. The ∃-*step semantics*, proposed in [Dimopoulos *et al.*, 1997] and further developed in [Rintanen *et al.*, 2006], is less restrictive since it only requires that parallel actions can be executed in some order. This relaxation allows much more parallelism. Another improvement is the *relaxed* ∃-*step semantics* [Wehrle and Rintanen, 2007], which allows actions to be scheduled in parallel in a state s even if not all of them are applicable in s. Basically, those actions' preconditions will be satisfied by the effects of other previously executed actions, when serialized. In the *relaxed relaxed* ∃-*step* (R^2∃-*step) semantics* [Balyo, 2013] the application of action effects is relaxed similarly to precondition requirements in the relaxed ∃-step semantics. Therefore the only requirement in the R^2∃-step semantics is that parallel actions can be ordered to form a valid sequential plan. From the ∃-step semantics and its subsequent relaxations, actions planned in a time step cannot be executed in parallel anymore, but the terms parallel plan and parallel step are used anyway.

In this work we go a step further in the pursuit of parallelism in the context of planning as SMT. Inspired by the highly relaxed semantics of [Balyo, 2013] for planning as SAT, we introduce a non-trivial encoding that lifts the R^2∃-step semantics to SMT.

The rest of the paper is structured as follows. In Section 2 we introduce PMT problems, together with the notation used throughout the paper. In Section 3 we present our encoding for planning as SMT, that adheres to the R^2∃-step semantics. In Section 4 we show how the redundant actions occurring in the resulting plans can be easily removed. In Section 5 we provide experimental results showing a dramatic decrease in the number of time steps needed to reach a valid plan in many numeric domains, compared to other similar exact (non-heuristic) planners. Since the presented encoding is parametric to a given order between actions (which governs how, given a parallel plan, actions are serialized and executed) in the experimental section we give some insights on the impact of this order on the efficiency, and some hints on suitable orders. Finally, in Sections 6 and 7 we explain the connections of this work to similar ideas and present some conclusions.

2 Planning Modulo Theories

We follow the notation defined in [Gregory *et al.*, 2012] for Planning Modulo Theories (PMT).

A *state* is a valuation over a finite set of variables X, i.e., an assignment function, mapping each variable $x \in X$ to a value in its domain, D_x. A *state space* for a set of variables X is the set of all valuations over X. In a *state space modulo* T, T is a theory defining the domains of the state space variables and interpretations for the constants, functions and predicates.

Definition 1 (Action)**.** *An* action a*, for a state space* S *modulo* T*, is a state transition function, comprising:*

- *A first-order sentence over* S *modulo* T*,* Pre_a *(the* precondition *of* a*).*

- *A set Eff_a (the* effects *of a), of assignments to a subset of the state variables in S, each setting a distinct variable to a value defined by an expression over S modulo T.*

An action a, for a state space S modulo T, is *applicable* (or *executable*) in a state $s \in S$ if $T, s \models Pre_a$ (that is, the theory together with the valuation s satisfies the precondition of a). We represent actions a as pairs $\langle Pre_a, Eff_a \rangle$, with the effects Eff_a often written as a substitution $\sigma_a = \{x_1 \mapsto exp_1, \ldots, x_n \mapsto exp_n\}$, where exp_i is an expression that defines the value of variable x_i in the resulting state, for each i in $1..n$ (e.g. $x \mapsto x + k$, for increasing a numeric variable x by k). We use $\top$ and $\bot$ to denote the Boolean *true* and *false* values, respectively. Given an expression e and a substitution σ, by $e\sigma$ we will denote the expression e after applying the substitution σ, i.e., after simultaneously replacing the variables in e by the expressions indicated in the substitution. Making abuse of notation, we will talk of a substitution as an assignment.

Following the application of a, the state is updated by the assignments in Eff_a to the variables that they affect, leaving all other variables unchanged. We denote the unique state resulting from applying an action a, in a state s in which is applicable, by $app_a(s)$. For any given sequence of actions $a_1; a_2; \ldots; a_n$ we define $app_{a_1;a_2;\ldots;a_n}(s)$ as $app_{a_n}(\cdots app_{a_2}(app_{a_1}(s))\cdots)$.

Definition 2 (Planning Modulo Theory)**.** *A* Planning Modulo T *problem, for a theory T, is a tuple $\pi = \langle S, A, I, G \rangle$ where:*

- *S is a state space in which all variable domains are defined in T,*
- *A is a set of actions for S modulo T,*
- *I is a valuation in S (the* initial state*), and*
- *G is a first order sentence over S modulo T (the* goal*).*

A (sequential) plan *for π is a sequence of actions $a_1; \ldots; a_n$ such that, for all i in $1..n$, a_i is applicable in state s_{i-1} and s_i is the result of applying a_i to s_{i-1}, where $s_0 = I$ and $T, s_n \models G$.*

3 Encoding Plans under the $R^2\exists$-Step Semantics in PMT

The notion of $R^2\exists$-step plan in the context of PMT is the following.

Definition 3 ($R^2\exists$-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S modulo T, a* relaxed relaxed $\exists$-step ($R^2\exists$-step) plan *for A and I is a sequence $P = [A_0, \ldots, A_{l-1}]$ of sets of actions together with a sequence of states $s_0, \ldots, s_l$ (the execution of P), for some $l \geq 0$, such that $s_0 = I$, and for all $i \in \{0, \ldots, l-1\}$ there is a total ordering $a_1 < \cdots < a_n$ of A_i, such that $T, app_{a_1;\ldots;a_{j-1}}(s_i) \models Pre_{a_j}$ for all $a_j = \langle Pre_{a_j}, Eff_{a_j} \rangle \in A_i$, and $app_{a_1;\ldots;a_n}(s_i) = s_{i+1}$.*

This is a weakening of the definition of relaxed $\exists$-plan in [Wehrle and Rintanen, 2007], since we remove the consistency requirement between effects of actions occurring at the same time step and, hence, the only requirement left is that those actions can be ordered to form a valid sequential plan. The definition also generalizes to the setting of PMT. It is equivalent to the definition of $\exists$-step plan in [Bofill *et al.*, 2016b], as well as to the definition of $\exists$-step plan in [Rintanen *et al.*, 2006] for the propositional case, which already captures $R^2\exists$-step plans. In those works, however, the encodings given for $\exists$-step plans are restricted to *happenings* which require, among other things, that preconditions of actions in each parallel step hold at the same time. On the contrary, here we are properly considering $R^2\exists$-step plans in the sense of [Balyo, 2013]. Notice however that no formal definition of $R^2\exists$-step plan is given in [Balyo, 2013].

Let us introduce a motivating example.

Example 1. *A merchant is looking to amass a small fortune in the fastest way. We will use the variable x to represent his gains. Suppose the merchant starts with no money ($x = 0$) and can perform two actions: work carrying boxes of tulips and gaining 10 coins a month, or invest some of his money in the tulip industry, doubling his earnings. His objective is reaching the sum of 20 coins. The actions can be modeled as follows:*

$$\textit{work:} \quad a_1 = \langle \top, x \mapsto x + 10 \rangle$$
$$\textit{invest:} \quad a_2 = \langle x > 5, x \mapsto x * 2 \rangle$$

If we consider the $\exists$-step semantics of [Rintanen et al.*, 2006], one of the requirements is that preconditions of parallel actions must hold at the start of the time step in which they are planned. Since the precondition of a_2 is not satisfied in the initial state, this would make the shortest $\exists$-step plan the following: $\prod = [\{a_1\}, \{a_2\}]$.*

With the Relaxed $\exists$-step semantics of [Wehrle and Rintanen, 2007], there is no requirement that forces preconditions of parallel actions to hold at the start of the time step in which they are planned. Still, there is a requirement of consistency between the effects applied at the same time step. The concept of consistency with numerical variables could be generalized as that all effects should be commutative. Since the effects $x \mapsto x + 10$ and $x \mapsto x * 2$ are not commutative, the shortest plan would also be $\prod = [\{a_1\}, \{a_2\}]$.

With the $R^2\exists$-step semantics, these requirements are lifted, so, considering the ordering between actions $a_1 < a_2$, our merchant can reach its goal with the one step plan $\prod = [\{a_1, a_2\}]$. This ordering can be used to model the application of non-commutative effects as follows, where x^t (resp. a^t) denotes the value of variable x (resp. execution of action a) at time step t:

$x^0 = 0$	*initial state*
$a_1^0 \rightarrow \top$	*precondition of* a_1
$a_1^0 \rightarrow x_1^0 = x_0^0 + 10,\ \neg a_1^0 \rightarrow x_1^0 = x_0^0$	*effects of* a_1
$a_2^0 \rightarrow x_1^0 > 5$	*precondition of* a_2
$a_2^0 \rightarrow x_2^0 = x_1^0 * 2,\ \neg a_2^0 \rightarrow x_2^0 = x_1^0$	*effects of* a_2
$x^0 = x_0^0,\ x_2^0 = x^1$	*tying constraints*
$x^1 = 20$	*goal*

The additional variables x_0, x_1 and x_2 (superscripted with the time step) permit us to accumulate effects over the variable x. The variable x_0 denotes the initial value of x. The

variable x_1 embodies the value of x after the possible execution of a_1. Note that x_1 gets an updated value if a_1 is executed, or keeps the previous value x_0 otherwise. Therefore, in the precondition of action a_2 we need to check x_1 instead of x. The effects of action a_2 are applied on x_1 and captured on x_2. Finally, the tying constraints link these auxiliary variables to the initial and final values of x. This encoding of chained effects will be the key to increase parallelism.

3.1 Encoding into SMT

In this section we propose an encoding for *planning as SMT*, as a particular case of PMT, that adheres to the $R^2\exists$-step semantics. The given encoding is valid for any theory T under quantifier-free first-order logic with equality. In particular, for numeric planning we could take T as the theory of the integers (or the reals) and use quantifier free linear integer (or real) arithmetic formulae.

Let $\pi = \langle S, A, I, G\rangle$ be a planning problem modulo theory T under a quantifier-free first-order logic with equality. For each variable x in $var(S)$ and each time step t, a new variable x^t of the corresponding type is introduced, denoting the value of x at step t. Moreover, for each action a and each time step t, a Boolean variable a^t is introduced, denoting whether a is executed at step t.

Given a term s, by s^t we denote term s, where all variables x in $var(S)$ have been replaced by x^t, and analogously for formulas. For example $(x + y)^t = x^t + y^t$, and $(p \wedge x > 0)^t = p^t \wedge x^t > 0$. For the case of effects, we define $\{x \mapsto \top\}^t = x^{t+1}$, $\{x \mapsto \bot\}^t = \neg x^{t+1}$ and $\{x \mapsto s\}^t = (x^{t+1} = s^t)$, where s is a non-Boolean term belonging to theory T. For example, for an assignment $\{x \mapsto x + k\}$, where k is a constant, we have $\{x \mapsto x + k\}^t = (x^{t+1} = x^t + k)$. For sets of assignments, i.e., action effects, we define $(\{x \mapsto s\} \cup \mathit{Eff})^t = \{x \mapsto s\}^t \wedge \mathit{Eff}^t$ and $\emptyset^t = \top$ where s is a term (either Boolean or not) and Eff is a set of assignments.

For each action $a = \langle Pre_a, \mathit{Eff}_a\rangle$ and each variable $x \in var(S)$, let $\mathit{Eff}_{a,x}$ denote the assignment $\{x \mapsto exp\}$ in Eff_a if any, or the empty set if there is no such assignment. For each $x \in var(S)$, let $A_x = \{a \mid a \in A \wedge \mathit{Eff}_{a,x} \neq \emptyset\}$, i.e., the set of actions that modify x.

As it has already been said, the only requirement in the $R^2\exists$-step semantics is that actions in each parallel step can be ordered to form a valid sequential plan. Then, let $L = [a_1, a_2, \ldots, a_{|A|}]$ be a list enumerating all actions. The relative position of each action in L will determine the total ordering $<_L$ needed to serialize the actions in each parallel step. Although according to the definition of $R^2\exists$-step a different ordering could be used in each parallel step, we will use the same ordering everywhere. It is worth noting that the chosen ordering governs the amount of possible parallelism. In any case, completeness of the method is guaranteed by the fact that the possibility of choosing exactly one action per time step is retained.

The encoding will need to refer to the i-th action (according to $<_L$) in each set A_x. To this purpose, a mapping ρ_x is defined, such that $\rho_x(i) = j$ if the i-th action in A_x, according to $<_L$, is a_j. Formally: for each $x \in var(S)$, let $\rho_x^{-0} : \{1, \ldots, |A_x|\} \to \{1, \ldots, |A|\}$ be a mapping such that $a_{\rho_x^{-0}(i)} \in A_x$ for all i in $1..|A_x|$ and $a_{\rho_x^{-0}(i)} <_L a_{\rho_x^{-0}(i+1)}$ for all i in $1..|A_x| - 1$. Let $\rho_x : \{0, \ldots, |A_x|\} \to \{0, \ldots, |A|\}$ be $\rho_x^{-0} \cup \{0 \mapsto 0\}$. The mapping $\{0 \mapsto 0\}$ is added for notational convenience (see below).

Example 2. *Consider a set of actions $A = \{a_1, a_2, a_3, a_4\}$ and $L = [a_3, a_2, a_1, a_4]$, i.e., $a_3 <_L a_2 <_L a_1 <_L a_4$. Suppose that variable x is modified by actions a_1 and a_3, so $A_x = \{a_1, a_3\}$ and $|A_x| = 2$. Then we have $\rho_x(1) = 3$ and $\rho_x(2) = 1$, because $a_3 <_L a_1$. Semantically, $\rho_x(1)$ can be read as "What is the first action that modifies x, given $<_L$?".*

For each time step t and variable x, we introduce $x^t_{\rho_x(0)}, \ldots, x^t_{\rho_x(|A_x|)}$ new *chaining variables* of the same type of x. Variable $x^t_{\rho_x(0)}$ (i.e., x^t_0) will denote the value of x at time step t and, for all i in $1..|A_x|$, $x^t_{\rho_x(i)}$ will denote the value of x after the sequential application (or not) of actions $a_{\rho_x(1)}, \ldots, a_{\rho_x(i)}$. These variables allow us to encode a possible *chain of assignments* at each parallel step. The formulation here is more complicated than in [Balyo, 2013], where only Boolean variables are considered and so serialization of actions is very simple.

To represent *chains of assignments*, in the encoding we need to refer, for a given action a_i and variable x, to the last action before a_i (according to $<_L$) that may have modified x. Therefore, we define $prev_x : \{1, \ldots, |A|\} \to \{0, \ldots, |A|\}$ to be the mapping satisfying $prev_x(i) = \rho_x(max(\{0\} \cup \{k \in 1..|A_x| \mid a_{\rho_x(k)} <_L a_i\}))$. Notice that, if there is no previous action that may modify x, then $prev_x(i) = 0$.

Example 3. *Continuing with Example 2, we would have $prev_x(1) = 3$, $prev_x(2) = 3$, $prev_x(3) = 0$ and $prev_x(4) = 1$. Semantically, $prev_x(1) = 3$ could be read as "Given the ordering $<_L$, what action that modifies x comes before a_1?". Note that $prev_x(3) = 0$ because no action before a_3 modifies x, given the ordering $<_L$.*

The constraints of the proposed encoding are the following. The execution of an action implies its preconditions, with the variables conveniently renamed in order to consider the effects of the execution of previous (according to $<_L$) actions in the same time step t:

$$a_i^t \rightarrow Pre^t_{a_i}\sigma^t_{prev}(i) \qquad \forall a_i \in A \tag{1}$$

where

$$\sigma^t_{prev}(i) = \cup_{x \in var(S)}\{x^t \mapsto x^t_{prev_x(i)}\}$$

This substitution is in charge of renaming all variables in $Pre^t_{a_i}$ that may have been modified *previously* in the same time step.

The execution of an action implies its effects (again, with the variables conveniently renamed):

$$a_i^t \rightarrow \mathit{Eff}^t_{a_i}\sigma^t_{mod_i}\sigma^t_{prev}(i) \qquad \forall a_i \in A \tag{2}$$

where

$$\sigma^t_{mod_i} = \cup_{x \in Dom(\mathit{Eff}_{a_i})}\{x^{t+1} \mapsto x^t_i\}$$

Recall that each effect in Eff_{a_i} is translated as an equality. Substitution $\sigma^t_{mod_i}$ only renames the left hand side of the equality x^{t+1} by x^t_i, while $\sigma^t_{prev}(i)$ renames all variables occurring in the right hand side that could have been possibly

modified by previous actions according to $<_L$. See Example 4 for a particular case of this renaming.

If an action is not executed, the previous value of each variable that it would have modified is carried forward:

$$\neg a_i^t \rightarrow \bigwedge_{x \in Dom(Eff_{a_i})} x_i^t = x_{prev_x(i)}^t \qquad \forall a_i \in A \qquad (3)$$

Moreover, initial and final auxiliary variables are linked with the original variables x^t and x^{t+1}:

$$x^t = x_0^t \quad \text{and} \quad x^{t+1} = x_{\rho_x(|A_x|)}^t \quad \forall x \in var(S) \qquad (4)$$

Finally, explanatory axioms express the reason of a change in state variables:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{a \in A_x} a^t \qquad \forall x \in var(S) \qquad (5)$$

That is, a change in the value of x implies the execution of at least one action that has an assignment to x among its effects.

Remark 1. *Explanatory axioms* (5) *are redundant in our setting, since they follow from Equation* (4) *and the inductive application of Equation* (3)*.*

The following example illustrates the behavior of the substitutions in Equation (2).

Example 4. *Let $\{a_1, a_2\}$ be a set of actions such that $a_1 <_L a_2$. If actions are defined as*

$$a_1 = \langle \top, y \mapsto y + 1 \rangle,\ a_2 = \langle \top, x \mapsto x + y \rangle$$

then the effects of a_1 and a_2 at time step t will be encoded as

$$a_1^t \rightarrow y_1^t = y_0^t + 1,\ a_2^t \rightarrow x_1^t = x_0^t + y_1^t$$

3.2 Soundness and Completeness

Definition 4. *Let $\pi = \langle S, A, I, G \rangle$ be a PMT problem, $<_L$ a total order on the actions in A and n a number of steps greater than 0. We denote by $E(\pi, n, <_L)$, the SMT formula resulting from the encoding of π described in Section 3.1 using order $<_L$, for n consecutive time steps.*

For each $t \in \{0..n\}$, we define $X^t = \{x^t | x \in var(S)\}$.

We define I^0 as the formula describing the initial state I with variables superscripted by time point 0.

We define G^n as the formula describing the goal G with variables superscripted by time point n.

Theorem 1 (Soundness)**.** *Given a PMT problem $\pi = \langle S, A, I, G \rangle$, a number of steps n and a total order $<_L$ between actions in A, if M is a model of the SMT formula $\varphi = E(\pi, n, <_L) \wedge I^0 \wedge G^n$, then we can infer a valid sequential plan for π from M.*

Proof sketch. We first prove that the theorem is true for $n = 1$ and then argue why this can be generalized to $n > 1$.

Let $n = 1$. Let $a_0^1, \ldots, a_0^k$ be the, according to $<_L$, ordered action variables set to true in the model of φ. Then, the corresponding sequence of actions $a_0^1; \ldots; a_0^k$ is a sequential plan of π: if $k = 0$ we are done since this means that G is already satisfied from X^0 without executing any action. Notice that if no action is executed, constraints (3) enforce equality between chained variables in X^0 and X^1. If $k > 0$ we need to prove that each action a_0^i is applicable after applying $a_0^1; \ldots; a_0^{i-1}$. This is guaranteed by constraints (1), (2) and (3). By (2) the effects of the previous actions are applied, resulting in a "temporal state". By (1) the precondition of a_0^i is satisfied by this temporal state. Roughly, this temporal state consists of the valuation assigning, to each $x \in S$, its updated value due to the effects of actions $a_0^1; \ldots; a_0^{i-1}$. This value is captured by the closest previous chaining variable $x_{prev(i)}^0$, thanks to constraints (2) and (3).

For $n > 1$, constraint (4) properly links, for each m in $1..n$, the variables X^{m-1} to the first temporal state of step m and the last temporal state of step m to X^m. Hence, if $a_0^1, \ldots, a_0^{k_0}, \ldots, a_n^1, \ldots, a_n^{k_n}$ are the action variables set to true in the model of $E(\pi, n, <_L)$, where each subset $a_i^1, \ldots, a_i^{k_i}$ is ordered according to $<_L$, the corresponding sequence of actions $a_0^1; \ldots; a_0^{k_0}; \ldots; a_n^1; \ldots; a_n^{k_n}$ is a valid sequential plan of π. □

Completeness is guaranteed since the possibility of executing exactly one action per time step is retained.

Theorem 2 (Completeness)**.** *Given a PMT problem $\pi = \langle S, A, I, G \rangle$, if there exists a valid sequential plan $a_1; \ldots; a_n$ for π then, for any total order $<_L$ on the actions of A, the SMT formula $\varphi = E(\pi, n, <_L) \wedge I^0 \wedge G^n$ is satisfiable.*

Proof sketch. Let $<_L$ be an arbitrary order on the actions in A. Mimicking the valid sequential plan $a_1; \ldots; a_n$, we build an assignment M to the variables of φ such that I^0 holds, and for all i in $1..n$,

a. a_i^i is true in M, and a_j^i is false in M for all $j \neq i$,

b. $Eff_{a_i}^i \sigma_{mod_i}^i \sigma_{prev}^i(i)$ holds under M,

c. $\bigwedge_{x \in Dom(Eff_{a_j})} x_j^i = x_{prev_x(j)}^i$ holds under M for all $j \neq i$, and

d. $x^i = x_0^i$ and $x^{i+1} = x_{\rho_x(|A_x|)}^i$ hold under M for all $x \in var(S)$.

Now let us consider any particular step number i. According to c, and by transitivity of equality, we have that $x_{prev_x(i)}^i$, which corresponds to x_j^i for some $a_j <_L a_i$, has the same value as x_0^i under M, for every variable x. Moreover, we have that $x^i = x_0^i$ holds by d. Therefore, (the succedent of) constraint (1) will hold if a_i can be executed in step i of the sequential plan, which is the case, since by assumption the sequential plan is valid, and I^0 holds by construction.

Analogously to before, by c and d, we have that x^{i+1} has the same value as x_i^i under M, for every variable x modified by a_i, i.e., the new value of x is carried forward to the next step. By induction, and validity of the sequential plan, this implies that G^n holds under M.

Note that constraint (2) holds by a and b, constraint (3) holds by a and c, and constraint (4) by d. Consistency of M follows from the validity of the sequential plan at hand. □

4 Elimination of Redundant Actions

Roughly speaking, redundant actions are those that can be removed from a plan, resulting in a plan still valid.

In our setting, either during the search or when retrieving a model, there is the possibility that some variables, denoting the execution of actions that may not be necessary to achieve the goal, are set to true by the SMT solver. Since highly parallel encodings like the one we have presented make this issue more noticeable, here we show how a plan can be optimized by removing redundant actions in a post-processing step.

For this purpose, a MaxSMT solver will be used, as the idea is to add soft clauses that penalize the optimum when variables that represent the execution of actions are assigned to true by the solver. Given a plan $\prod$ for the PMT problem, two new sets of clauses are added. The first is a set of *soft* clauses $\{\neg a^t | a \in \prod\}$ that will (softly) ask the solver to set to false all actions included in the plan. This will force the MaxSMT solver to set to true the minimum number of actions already included in the plan.

The second set consists of *hard* clauses $\{\neg a^t | a \notin \prod\}$ to force the solver to not consider any action that was outside of the original plan. After solving the new problem, the resulting plan will not necessarily be makespan-optimal, but the quantity of actions pruned will be usually notable, as shown in the next section. Moreover, the optimization time with respect to the solving time turns out to be negligible.

The presented approach is slightly different to the one presented in [Balyo *et al.*, 2014]. The difference lies in that in our approach there is no need to reformulate all the problem from scratch to include only clauses referring to the plan returned by the solver. It suffices to add the new clauses to the problem and ask the solver for an optimal solution.

5 Empirical Evaluation

In this section we evaluate the impact of the presented encoding under the $R^2\exists$-step plan semantics and the proposed strategy for eliminating redundant actions. The proposed encoding (R^2Chained onwards)[1] is compared with the encoding in [Bofill *et al.*, 2016b] which uses the $\exists$-step semantics (from now on noted as *SEM+C*), and the two planners Springroll [Scala *et al.*, 2016] and SMTPlan [Cashmore *et al.*, 2016]. The three systems are the most recent non-heuristic numeric planners available. The *SEM+C* encoding is similar to the disabling graph based encodings for $\exists$-step semantics of [Rintanen *et al.*, 2006]. It differs by incorporating the idea of chains of assignments that occur at the same time step. It also provides a semantic notion of interference to reduce the number of false positives in interference detection. This *SEM+C* encoding and the R^2Chained encoding are implemented in the RANTANPLAN system [Bofill *et al.*, 2016a]. Springroll also uses the planning as SMT approach, but with a $\forall$-step semantics. The planner focuses on producing more succinct encodings by "rolling up" an unbounded yet finite number of instances of an action into a single plan step. In problems where "foldable" actions occur, the planner is able to greatly reduce the number of time steps. SMTPlan proposes an approach to PDDL+ planning through SMT, with an encoding that captures all the features of the PDDL+ language. Its encoding focuses on domains with nonlinear and continuous change.

Solved	Springroll	SMTPlan	SEM+C	R^2C
Depots (22)	**7**	1	4	7
Driverlog (20)	12	7	11	**12**
Petrobras (60)	0	3	6	**51**
Planes (12)	3	5	7	**8**
Rovers (20)	12	4	6	**16**
Zenotravel (20)	*	6	**16**	15
Total	34	26	50	**109**

Table 1: Number of instances solved by each planner in each domain (total number of instances between parentheses), with a timeout of 1 hour. Boldface indicates the best results, with ties broken by total solving time. "*" denotes an execution problem.

In [Balyo and Barták, 2015] it was found experimentally that none of the considered orderings between actions could be clearly defined as the best, for the considered encoding under the $R^2\exists$-Step semantics for planning as SAT. In the first experiments, the ordering considered for the R^2Chained encoding is the order from which actions are read from the input files, which we refer as *dec*. Some insights and results on more clever orderings, other than *dec*, are given in Section 5.1. It is also worth noticing that, although we choose the same order for each time step, the encoding is general enough to allow for a different order in each time step.

We consider the domains of the third IPC [Long and Fox, 2003] with integer numeric fluents and without quantified preconditions, as the rest of the domains contain features that are not commonly supported by the considered planners. These domains are: *Zenotravel*, *Driverlog*, *Depots* and *Rovers*. The *Petrobras* and *Planes* domains from [Bofill *et al.*, 2016b] are also considered since they have a higher numerical component.

Experiments have been run on 8GB Intel® Xeon® E3-1220v2 machines at 3.10 GHz, using Yices [Dutertre and De Moura, 2006] v2.5.1 as the back-end SMT solver, under the quantifier-free linear integer arithmetic logic [Barrett *et al.*, 2010]. Z3 [de Moura and Bjørner, 2008] v4.5.1 is used as the MaxSMT solver to remove redundant actions. The total timeout is set to 1 hour.

Table 1 shows that with the R^2Chained (R^2C) encoding we are able to solve notably more instances than the rest of the approaches. Springroll is unable to process the Zenotravel domain.[2] The R^2Chained encoding dominates in most of the families. Thanks to the increased number of actions selected at each step, it generally needs fewer steps to find a valid plan than the rest of the planners. The number of Petrobras instances solved by this approach is noticeable. This domain is notably bigger in terms of formula size and more constrained in terms of resources than the rest. If this domain is set aside, the R^2Chained encoding still solves a few more instances than the other approaches.

Next we evaluate the plan quality in terms of makespan. As it has already been stated, the R^2Chained approach allows many more actions per time step. This increases the

[1] The planner and the non-IPC instances can be obtained from http://imae.udg.edu/recerca/lap/rantanplan/rantanplan.html

[2] After reading the instance, it reports "Error in the encoding".

	Springroll	SMTPlan	SEM+C	R^2C	R^2O
depots1	13/6	13/6	13/6	13/2	13/2
depots7	29/10	-	25/10	30/4	25/4
depots16	34/8	-	-	59/3	33/3
driverlog5	30/8	25/8	18/4	30/4	20/4
driverlog6	26/5	17/5	21/4	22/3	22/3
petro-A2	-	9/3	11/4	11/2	10/2
petro-A6	-	-	33/9	38/5	34/5
petro-B15	-	-	-	82/2	57/2
petro-C1	-	-	-	34/2	5/2
planes2	17/16	18/11	17/11	19/7	19/7
planes3	19/17	27/13	19/10	23/7	23/7
planes8	-	-	24/12	27/7	25/7
rovers1	11/9	11/8	11/8	13/3	9/3
rovers4	31/6	10/6	8/5	8/1	8/1
rovers14	50/11	-	-	41/3	32/3
zeno7	*	16/6	19/3	18/2	16/2
zeno8	*	-	22/3	37/2	22/2

Table 2: Number of actions / number of steps, per instance and planner. "-" denotes a timeout. R^2O denotes the R^2C approach plus the redundant action removal presented in the previous section. "*" denotes an execution problem.

number of instances solved, since the number of time steps is in general smaller and, hence, so is the resulting formula. But unfortunately, this is bad in terms of plan quality, since it may add some redundant actions in the plan.

Table 2 shows the number of actions and time steps of the plans found in a selected number of instances. Comparing the R^2C and the R^2O columns we can see that the reduction is notable in all domains, except for Planes. We remark that the time spent on the process of removing redundant actions is negligible (typically less than two seconds).

In general, in instances where the reduction is small, the plan was already reasonably good, in terms of number of actions, compared to the rest of the planners. See for example instances driverlog6, petro-A2, planes2 or rovers4. In instances where the reduction is significant, the original plan was too long and the optimized one turns to be reasonably good compared to the others. See for instance depots7, depots16, driverlog5 or zeno8. In particular, (see petro-C1 for example) there can be many agents (namely, the ships) that are not relevant for the plan objective, so the procedure can remove many actions. In contrast, the Planes domain is very tight, as there are very few agents that need to act, and thus all planners produce similar plans in terms of makespan.

Since the R^2Chained encoding relaxes the SEM+C encoding, we provide some insights on why, in general, it behaves better. The relaxation is done by applying the idea of creating chains of assignments to all variables, while in the SEM+C encoding only a subset of them are eligible to be chained, due to the interference notion considered. In contrast to R^2Chained, there is no possible chain for Boolean variables in SEM+C. In fact, the difference between the two encodings can be seen as a trade-off between the need for mutex clauses and the extra number of chaining variables and linking constraints. Moreover, the relaxation by the $R^2\exists$-step

R^2C w.r.t. SEM+C	clauses/ time step	variables/ time step	final num. variables	final num. clauses
Depots(4)	+24.1%	+39.3%	-48.1%	-54.1%
Driver.(11)	-1.7%	+18.1%	-12.5%	-28.2%
Petrobras(6)	-54.9%	+27.7%	-27.1%	-74.2%
Planes(7)	+2.1%	+14.3%	-26.2%	-29.7%
Rovers(6)	+65.5%	+52.4%	-48.2%	-52.0%
Zeno.(15)	-0.6%	+17.2%	-10.0%	-23.2%

Table 3: Average problem size difference between the R^2*Chained* encoding and the *SEM+C* encoding for each domain. The number of commonly solved instances is between parentheses. The first two columns show the average size difference in a single time step, and the second pair similarly but at the step where the solution is found.

Time steps	SEM+C		R^2C	
	t. steps	avg. steps	t. steps	avg. steps
Depots (4)	37	9.25	14	3.50
Driverlog (11)	51	4.64	38	3.45
Petrobras (6)	36	6.00	21	3.50
Planes (7)	76	10.86	49	7.00
Rovers (6)	41	6.83	13	2.17
Zenotravel (15)	49	3.27	37	2.47

Table 4: Results on the number of time steps needed for the commonly solved instances between the R^2*Chained* and the *SEM+C* approaches. The number of commonly solved instances is shown between parentheses. Columns *t. steps* show the sum of all the steps of the commonly solved instances of each family, and columns *avg. steps* show the average steps per instance commonly solved.

semantics of actions' applicability at the beginning of a time step, as well as that of the consistency of effects, not only allows us to solve more instances, but also to use many less time steps on the commonly solved ones.

Table 3 essentially shows that although R^2C uses more clauses and variables in each formula tested for satisfiability, it is able to solve the problem at an earlier step than SEM+C. This reduction in steps is especially noticeable in Depots and Rovers, where although the formula size per time step is bigger, the size of the last checked formula (i.e., the first satisfiable formula) is nearly cut in half. In the Petrobras domain, even using a semantic notion of interference, the SEM+C approach generates many mutexes, as there are many incompatibilities between actions. The removal of mutexes lets the R^2Chained encoding state the problem more compactly at each time step. This decrease in size per step, combined with the decrease in the number of needed time steps, lets the planner find a feasible solution with a reduction of nearly 75% of the problem clauses. The gains of these reductions are also reflected in Table 1, where the difference in the number of problems solved on the Depots, Rovers and Petrobras domains is noticeable.

To better illustrate the size of the final formulas shown on the second pair of columns of Table 3, in Table 4 the results on the number of steps are shown. Note that Depots and Rovers instances with the R^2Chained encoding need nearly a third of the steps needed by the SEM+C approach, and in the Petrobras domain this is nearly the half.

	R^2C *dec*		R^2C informed *dec*		
	t. time	t. steps	t. time	t. steps	Δ_i
Depots(7)	7994.7	34	116.6	29	+2
Driverlog(11)	1734.3	49	2049.3	42	+2
Petrobras(49)	9546.3	150	57.1	57	-2
Planes(8)	1196.2	65	67.0	48	+2
Rovers(16)	375.5	72	257.4	58	+1
Zenotravel(15)	3735.6	52	631.6	42	+1
Total (96)	24582.6	422	3179.0	276	+6

Table 5: Total time and steps needed to solve the commonly solved instances with R^2Chained and the *dec* ordering, with and without informing. The number of commonly solved instances is shown between parentheses. Column Δ_i shows the difference in total instances solved.

5.1 Orderings

In this section we try to reason about the effect of the selected ordering between actions on the efficiency of the encoding. Imagine a planning task with three actions a_1, a_2, a_3, applicable in the initial state and with a goal state that can be reached by the execution of the three tasks. Consider that a_1 disables a_2, a_2 disables a_3, and there are no further disabling relations. The disabling graph based encodings for $\exists$-step semantics of [Rintanen *et al.*, 2006] will not produce any constraints with respect to parallelism, because there are no cyclic disabling relations. Supposing that all actions are initially applicable and have consistent effects, it will find the plan a_3, a_2, a_1 with only one time step. However, bad orderings like $a_1 < a_2 < a_3$ would force the R^2Chained encoding to make three time steps.

To avoid this, we have considered the *rdfs* ordering, being the reverse of a depth-first search on the disabling graph. The rationale for this ordering is to try to minimize the number of possible interferences on actions appearing later in the ordering, and thus maximizing the number of actions potentially executed at the same time step. Surprisingly, the *rdfs* ordering solved globally three fewer instances. Probably, this happens because the a-priori computed interferences are not a good indicator on how the actions should be ordered.

Intuitively, a good ordering for the encoding at hand would be one inferred from a valid sequential plan, because the sequence of actions needed for a valid plan is strongly influenced by the objective and the initial state. Thus, finding an optimal ordering should be as hard as finding a plan itself.

To experimentally validate the previous assumption, we propose to *inform* a given total ordering using a sequential plan obtained from a relaxed version of the planning problem. This plan is obtained by using delete relaxation heuristics [Bonet and Geffner, 2001] on the original problem and removing the predicates belonging to the considered theory T. Once a plan is obtained by solving the relaxed problem, we extract an ordering by serializing this plan and removing duplicate occurrences of each action. Then, given a total ordering on the actions, we can *inform* it by only reordering the subset of actions appearing in the relaxed plan, according to the order in which they occur in the relaxed plan.

Table 5 shows that informing the previously used *dec* ordering, results in needing 146 steps fewer to solve the same number of instances than without informing it. This increase in parallelism is followed by a dramatic reduction on solving times (of about two orders of magnitude) in most of the families, resulting in 6 more instances solved.

We also informed the *rdfs* ordering, a random ordering and all its inverted versions. None of the orderings was clearly better. However, the gains on the number of solved instances by informing them was always positive, ranging from 4 to 9 extra instances, experimentally supporting the intuition.

Regarding plan quality, with respect to the number of actions, experiments with informed orderings show that no significant change can be seen. Note that fewer time steps imply less space for possible redundant actions, but more parallelism can also mean more selected actions per time step.

6 Related Work

A macro-action [Korf, 1985; Minton, 1985] expresses the combination of one or more actions. Methods for automatically learning macro-actions have been developed [Botea *et al.*, 2005]. Also, macro-actions have been extended for numeric planning problems [Scala, 2014].

The way actions are sequenced in a time step is de-facto a combination of regression and progression of precondition and action effects. This can be achieved via substitution, and it amounts to computing weakest precondition and cumulative effects, which is how numeric macro-actions can be built. However, a difference with macro-actions is that, in the presented encoding, the actions that are going to be sequenced are not fixed. In other words, our encoding benefits from letting the solver decide which subsequence is necessary to use.

7 Conclusion

The R^2Chained encoding has empirically proven to allow for more parallelism than that of the rest of the considered planners for planning modulo theories. It is able to put many more actions per step, also making the search space wider for each step considered. However, generally a shorter horizon pays off in terms of formula size and solving time, as can be seen from the experiments. The post-processing step for eliminating redundant actions has proven to be cheap in terms of solving time, and useful for maintaining the plan quality when considering highly parallel encodings. Also, a good heuristic for improving orderings for actions has been proposed.

Designing better orderings and heuristics is a matter of future work. We would also like to study the effects of adding or removing implied constraints, like (5), in the encoding.

Acknowledgments

Work supported by grants TIN2015-66293-R (MINECO/FEDER, UE) and MPCUdG2016/055 (UdG). We would like to thank the reviewers for their constructive comments and the observations regarding Constraint (5) of the encoding. We also thank Dr. Alan Frisch for his helpful comments.

References

[Balyo and Barták, 2015] Tomas Balyo and Roman Barták. No One SATPlan Encoding To Rule Them All. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, pages 146–150, 2015.

[Balyo *et al.*, 2014] Tomas Balyo, Lukás Chrpa, and Asma Kilani. On different strategies for eliminating redundant actions from plans. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014.*, 2014.

[Balyo, 2013] Tomas Balyo. Relaxing the Relaxed Exist-Step Parallel Planning Semantics. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, pages 865–871, 2013.

[Barrett *et al.*, 2010] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[Bofill *et al.*, 2016a] Miquel Bofill, Joan Espasa, and Mateu Villaret. The RANTANPLAN planner: system description. *The Knowledge Engineering Review (KER)*, 31(5):452–464, 2016.

[Bofill *et al.*, 2016b] Miquel Bofill, Joan Espasa, and Mateu Villaret. A semantic notion of interference for planning modulo theories. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 56–64, 2016.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Botea *et al.*, 2005] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-ff: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[Cashmore *et al.*, 2016] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full PDDL+ language into SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 79–87, 2016.

[de Moura and Bjørner, 2008] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS March 29-April 6, 2008.*, pages 337–340, 2008.

[Dimopoulos *et al.*, 1997] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding Planning Problems in Nonmonotonic Logic Programs. In *Recent Advances in AI Planning, Fourth European Conference on Planning, ECP 1997*, volume 1348 of *LNCS*, pages 169–181. Springer, 1997.

[Dutertre and De Moura, 2006] Bruno Dutertre and Leonardo De Moura. The Yices SMT Solver. Technical report, Computer Science Laboratory, SRI International, 2006. Available at `http://yices.csl.sri.com`.

[Gregory *et al.*, 2012] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. Planning Modulo Theories: Extending the Planning Paradigm. In *Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI, 2012.

[Kautz and Selman, 1996] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 2.*, pages 1194–1201, 1996.

[Korf, 1985] Richard E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[Long and Fox, 2003] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research (JAIR)*, 20:1–59, 2003.

[Minton, 1985] Steven Minton. Selectively generalizing plans for problem-solving. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, IJCAI 1985. Los Angeles, CA, USA, August 1985*, pages 596–599, 1985.

[Rintanen *et al.*, 2006] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[Scala *et al.*, 2016] Enrico Scala, Miquel Ramírez, Patrik Haslum, and Sylvie Thiébaux. 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.*, pages 276–284, 2016.

[Scala, 2014] Enrico Scala. Plan repair for resource constrained tasks via numeric macro actions. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.

[Wehrle and Rintanen, 2007] Martin Wehrle and Jussi Rintanen. Planning as Satisfiability with Relaxed Exists-Step Plans. In *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, pages 244–253, 2007.