Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

A Semantic Notion of Interference for Planning Modulo Theories

Miquel Bofill and **Joan Espasa** and **Mateu Villaret**
Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, Spain
{mbofill,jespasa,villaret}@imae.udg.edu

Abstract

The aim of being able to reason about quantities, time, space and much more has been the main objective of the many efforts on the integration of propositional planning with extensions to handle different theories. Planning Modulo Theories (PMT) is an approximation inspired by Satisfiability Modulo Theories (SMT) that generalizes the integration of arbitrary theories with propositional planning. Parallel plans are crucial to reduce plan lengths and hence the time needed to reach a feasible plan in many approaches. Parallelization of actions relies on the notion of (non-)interference, which is usually determined syntactically at compile time. In this paper we present a general semantic notion of interference between actions in PMT. Along its generality, this notion can be efficiently checked at compile time by means of satisfiability checks.

Introduction

The problem of planning, in its most basic form, consists in finding a sequence of actions that allows to reach a goal state from a given initial state. In its classical approach, state variables are propositional, but in many real world problems we need to reason about concepts such as time, space, capacities, etc. that are impractical to represent using only propositional variables. Many planners have been built to be able to deal with such problems. These solvers glue together a classical planner with a specific theory solver that exclusively handles the non-propositional part. Planning Modulo Theories (PMT) (Gregory et al. 2012) is a modular framework inspired in the architecture of lazy SMT, which is the natural extension of SAT when propositional formulas need to be combined with other theories.

The possibility of several actions being planned at the same time step, i.e., the notion of parallel plans, can be considered a key idea to reduce plan lengths and hence the time needed to reach a feasible plan in many approaches. Parallel plans increase the efficiency not only because they allow to reduce the time horizon, but also because it is unnecessary to consider all total orderings of the actions that are performed in parallel when seeking for a plan.

A problem that arises when considering parallel plans is that of interference between actions. Is is commonly accepted that it should be possible to serialize every parallel plan while preserving its semantics. Hence, it is usually required that effects of actions planned at the same time commute, and that there is no interaction between preconditions and effects of different actions. Existing works on numeric planning use syntactic or limited semantic approaches to determine interference between actions, in a fairly restrictive way (Kautz and Walser 1999; Fox and Long 2003; Gerevini, Saetti, and Serina 2008). In this work we propose a general notion of interference between actions, and a new relaxed semantics for parallel plans, in the context of PMT. We prove its correctness, motivate its usefulness with some examples, and show how it can be easily implemented.

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Planning Modulo Theories

We follow the concepts and notation defined in (Gregory et al. 2012) for Planning Modulo Theories (PMT). The key to extending classical planning into PMT is to support first order sentences modulo theories in the preconditions of actions.

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. The expression $s[x]$ denotes the value state s assigns to variable x, and $s[x \mapsto v]$ is the state identical to s except that it assigns the value v to variable x. A *state space* for a set of variables X is the set of all valuations over X. By $var(S)$ we denote the state variables of a state space S.

A *first order sentence over a state space S modulo T* is a first order sentence over the variables of the state space, constant symbols, function symbols and predicate symbols, where T is a theory defining the domains of the state space variables and interpretations for the constants, functions and predicates.[1] A *state space modulo T* is a state space ranging over the domains defined in T. A *term* over S modulo T is, similarly, an expression constructed using the symbols defined by S and T. A formula ϕ is T-satisfiable if $\phi \wedge T$ is satisfiable in the first-order sense. By $eval^s_T(\phi)$ we denote the value of ϕ under the assignment s, according to the interpretation defined by theory T.

A *substitution* is a partial mapping from variables to

[1] In some other contexts, such as mathematical logic, a theory is understood as being just a set of sentences.

terms. It can be represented explicitly as a function by a set of bindings of variables to terms. That is, if $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, then $\sigma(x_i) = t_i$ for all i in $1..n$, and $\sigma(x) = x$ for every other variable.

Substitutions are extended homomorphically to a total mapping from terms to terms. We use the postfix notation $t\sigma$ for the image of a term t under a substitution σ. This is defined inductively on the structure of terms as follows:

$$t\sigma = \begin{cases} \sigma(t) & \text{if } t \text{ is a variable} \\ f(t_1\sigma, \ldots, t_m\sigma) & \text{if } t \text{ is of the form } f(t_1, \ldots, t_m) \end{cases}$$

In the second case of this definition, $m = 0$ is allowed: in this case, f is a constant symbol and $f\sigma$ is f. Thus $t\sigma$ is t with all variables replaced by terms as specified by σ. The image of a formula under a substitution is defined similarly.

The composition of two substitutions σ_1 and σ_2, denoted by juxtaposition, is defined as the composition of two functions, that is, $t\sigma_1\sigma_2 = (t\sigma_1)\sigma_2$.

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 a term 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 $\langle Pre_a, \mathit{Eff}_a \rangle$, 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. Making abuse of notation, we will talk of a substitution as an assignment.

Following 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 action a, in a state s in which is applicable, by $app_a(s)$. Formally, the resulting state s' is the mapping where, for each variable $x \in var(S)$, $s'(x) = eval^s_T(x\sigma_a)$, where σ_a is the substitution representing the effects of a.

For sequences $a_1; a_2; \ldots; a_n$ of actions 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 of the 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 $T, s_n \models G$.

As it is usual in SMT, we assume T is a first-order theory with equality, which means that the equality symbol $=$ is a predefined predicate, interpreted as the identity on the underlying domain. Sometimes we say that a sequence of actions is a plan starting from an initial state I, without specifying the goal. In this case we mean that the plan is executable starting from I.

The ILP-PLAN framework (Kautz and Walser 1999), based on integer optimization of linear integer programs, is a particular case of this, taking linear inequalities as preconditions, and limiting effects to increasing, decreasing or setting the value of a variable. Numeric planning, as defined in (Helmert 2002) or in (Gerevini, Saetti, and Serina 2008) is also a particular case, using a very limited fragment of first-order logic in the preconditions of actions, and taking T as the theory of rational functions, i.e., fractions between polynomials. The proposal in (Rintanen, Heljanko, and Niemelä 2006) raises preconditions to general Boolean formulae, but does not consider numeric variables.

A Semantic Notion of Interference

In the following, we consider plans as sequences of *sets of actions*. A set of actions planned at the same time is commonly called a *happening* (Fox and Long 2003). Two actions can be concurrently planned if, roughly, they do not interfere. It is commonly accepted that two actions are non-interfering only if the composition of their effects is commutative, and there is no interaction between effects and preconditions. In (Fox and Long 2003; Gerevini, Saetti, and Serina 2008) the state resulting from executing a happening is defined as the one obtained after applying the composition of effects of the actions in the happening.

Example 1. Let $a = \langle \top, \{x \mapsto x + y + z\} \rangle$ and $b = \langle \top, \{x \mapsto x + 1, y \mapsto y + 1, z \mapsto z - 1\} \rangle$. These actions do not interfere, as their preconditions are true (and hence cannot interact with effects) and their effects commute: executing first a and then b, as well as executing first b and then a, produces the same effect, which is that of an action of the form $\langle \top, \{x \mapsto x + y + z + 1, y \mapsto y + 1, z \mapsto z - 1\} \rangle$, defining the state transition function of their simultaneous execution.

Example 2. Let $c = \langle \top, \{x \mapsto x + y + z\} \rangle$ and $d = \langle \top, \{x \mapsto x + 1, y \mapsto y + 2, z \mapsto z - 1\} \rangle$. These actions interfere, since their effects do not commute. Executing first c and then d is equivalent to executing $\langle \top, \{x \mapsto x + y + z + 1, y \mapsto y + 2, z \mapsto z - 1\}$, whereas executing first d and then c is equivalent to executing $\langle \top, \{x \mapsto x + y + z + 2, y \mapsto y + 2, z \mapsto z - 1\}$. Then they would not be allowed to be planned in parallel.

Thanks to the commutativity requirement, effects of non-interfering actions can be composed in any order, allowing parallel plans to be serialized in any order, while preserving their semantics. This adheres to the $\forall$-step semantics

of (Rintanen, Heljanko, and Niemelä 2006), but it does not lift to the ∃-step semantics (introduced in the same work for the Boolean case), where it is only necessary that actions can be executed it at least one order, making it possible to increase the number of parallel actions.

The main contributions of this paper are

- a new relaxed semantics of *happening execution*, and
- a new notion of interference

that are suitable for both ∀-step and ∃-step semantics (Rintanen, Heljanko, and Niemelä 2006), in the context of PMT.

As we show in the following section, the proposed notion of interference can moreover be fully checked at compile time by means of satisfiability checks. As far as we know, previous approaches used syntactic or limited semantic approaches (Kautz and Walser 1999; Fox and Long 2003; Gerevini, Saetti, and Serina 2008). Note that non-interference of actions such those in Example 1 cannot be easily determined syntactically.

In the rest of this section we introduce the new semantics of happening execution, define the new notion of interference, and prove that their combination is valid for both ∀-step and ∃-step semantics.

Definition 3 (Commuting Assignments)**.** Two assignments $\{x \mapsto exp_1\}$ and $\{x \mapsto exp_2\}$ *commute*, for a variable x and two expressions (terms) exp_1 and exp_2 over a state space S modulo T, if $T \models (exp_2\{x \mapsto exp_1\} = exp_1\{x \mapsto exp_2\})$.

Example 3. If T is the theory of the reals, then $\{x \mapsto x+1\}$ and $\{x \mapsto x-2\}$ commute, since $T \models ((x-2)+1 = (x+1)-2)$, whereas $\{x \mapsto x+1\}$ and $\{x \mapsto x*2\}$ do not commute, since $T \not\models ((x+1)*2 = (x*2)+1)$.

Definition 4 (Simply Commuting Actions)**.** We will refer to a set $A = \{a_1, \ldots, a_n\}$ of actions as *simply commuting*, for a state space S modulo T, if for every variable $x \in var(S)$ and every pair of assignments $\{x \mapsto exp_1\}$ and $\{x \mapsto exp_2\}$ in the effects of actions in A, $\{x \mapsto exp_1\}$ and $\{x \mapsto exp_2\}$ commute.

Definition 5 (Happening Action)**.** Let $A = \{a_1, \ldots, a_n\}$ be a set of simply commuting actions. We define the *happening action* for A as an action $h(A) = \langle Pre_{h(A)}, \sigma_{h(A)} \rangle$ with

$$Pre_{h(A)} = \bigwedge_{a \in A} Pre_a$$

and

$$\sigma_{h(A)} = \bigcup_{x \in var(S)} \{\sigma_{x,1} \circ \cdots \circ \sigma_{x,n}\}$$

where $\sigma_{x,i}$, for i in $1..n$, is the mapping of variable x in the effects of action a_i, and $\circ$ denotes the composition of functions.

Note that the effects on each variable can be composed in any order, because of the commutation requirement. Therefore, $h(A)$ is well-defined.

Definition 6 (Happening Execution)**.** Let $A = \{a_1, \ldots, a_n\}$ be a set of simply commuting actions. Then, the state resulting from the execution of the happening A in state s, denoted $app_A(s)$, is defined as $app_{h(A)}(s)$, where $h(A)$ is the happening action corresponding to A.

Note that if some action in A is not applicable in state s then $app_A(s)$ is undefined.

Example 4. Let $a = \langle \top, \{x \mapsto x+1, y \mapsto y+1\} \rangle$ and $b = \langle \top, \{y \mapsto y+x\} \rangle$. Then $app_{\{a,b\}}(s)$, for a state s, is $app_{h(\{a,b\})}(s)$, with $h(\{a,b\}) = \langle \top, \{x \mapsto x+1, y \mapsto (y+x)+1\} \rangle$.

We claim that the transition functions for happenings defined in (Fox and Long 2003) and (Gerevini, Saetti, and Serina 2008) are particular cases of Definition 6. A key difference we want to point out is that, instead of considering the *composition of functions* (i.e., the composition of effects of actions, seen as functions on all variables), we are considering the *function of compositions* (i.e., the function defined by the composition of assignments to each single variable across all actions). We consider the possibility of composing the effects on each variable in any order, as a minimal requirement to be able to serialize plans in some order (see definitions and proofs below). We believe that in the restricted case of (Fox and Long 2003) and (Gerevini, Saetti, and Serina 2008), where commutativity of all effects is required, both concepts coincide, but proving it is beyond the scope of this paper. Our aim is to show that the proposed semantics for happenings allows us to increase the number of parallel actions in the context of ∃-step plans, where parallel semantics and interference notions of existing approaches to numeric planning are too restrictive.

Definition 7 (Affecting Action)**.** Given two actions $a = \langle Pre_a, \sigma_a \rangle$ and $b = \langle Pre_b, \sigma_b \rangle$, for a state space S modulo T, we consider a to *affect* b if

1. $Pre_a \wedge Pre_b \wedge \neg(Pre_b\sigma_a)$ is T-satisfiable, or
2. either a and b are not simply commuting, or $Pre_a \wedge Pre_b \wedge \neg(x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$ is T-satisfiable for some variable $x \in var(S)$, where $h(\{a,b\})$ denotes the happening action for a and b,

that is, a can impede the execution of b, or they are not simply commuting, or they are simply commuting but executing first a and then b has a different effect than that of the happening $\{a,b\}$.

Recall that $h(\{a,b\})$ is defined only for simply commuting actions.

Example 5. Following Example 4, where actions a and b are simply commuting, we have that a affects b since $y\sigma_{h(\{a,b\})} = (y+x)+1$, while $y\sigma_b\sigma_a = (y+x)\sigma_a = (y+1)+(x+1)$, and thus $Pre_a \wedge Pre_b \wedge \neg(y\sigma_{h(\{a,b\})} = y\sigma_b\sigma_a)$ is T-satisfiable. On the contrary, b does not affect a, since the preconditions of both actions are true, $x\sigma_{h(\{a,b\})} = x+1 = x\sigma_b\sigma_a$, and $y\sigma_a\sigma_b = (y+1)\sigma_b = (y+x)+1$. This is to say that the effect of the happening $\{a,b\}$ is the same as executing first b and then a, but not first a and then b. In fact, in this example we have $app_{\{a,b\}}(s) = app_{b;a}(s) \neq app_{a;b}(s)$ for all s.

Definition 8 (Interference)**.** Given two actions a and b, we consider a and b to *interfere* if a affects b or b affects a.

∀-Step Plans

Lack of interference guarantees that actions in a happening can be executed sequentially in any total order and that the final state is independent of the ordering (see Theorem 1). The notion of ∀-step plan, defined in (Rintanen, Heljanko, and Niemelä 2006), can be generalized to the setting of PMT as follows.

Definition 9 (∀-Step Plan)**.** Given a set of actions A and an initial state I, for a state space S modulo T, a *∀-step plan* for A and I is a sequence $P = \langle A_0, \ldots, A_{l-1} \rangle$ of sets of actions for some $l \geq 0$, such that there is a sequence of states $s_0, \ldots, s_l$ (the execution of P) such that

1. $s_0 = I$, and
2. for all $i \in \{0, \ldots, l-1\}$ and every total ordering $a_1 < \cdots < a_n$ of A_i, $app_{a_1;\ldots;a_n}(s_i)$ is defined and equals s_{i+1}.

Lemma 1. Let A be a set of actions, for a state space S modulo T, and let $s \in S$ be a state such that all actions in A are applicable in s. Then $app_{a_1;\ldots;a_n}(s)$ is defined for every ordering $a_1 < \cdots < a_n$ of A such that if $a_i < a_j$ then a_i does not affect a_j.

Proof. By induction on the number of actions n in A. If $n = 1$ we are trivially done. If $n \geq 2$, consider any ordering $a_1 < \cdots < a_n$ of A such that if $a_i < a_j$ then a_i does not affect a_j. Let $a_1 = \langle Pre_{a_1}, \sigma_{a_1} \rangle$. First of all we show, by contradiction, that $app_{a_i}(app_{a_1}(s))$ is defined for every $a_i = \langle Pre_{a_i}, \sigma_{a_i} \rangle$ such that $a_1 < a_i$. Suppose that $T, app_{a_1}(s) \not\models Pre_{a_i}$, i.e., that a_i is not applicable after applying a_1 in state s. This is equivalent to say that $T, s \not\models Pre_{a_i}\sigma_{a_1}$ and, since s is an assignment, to $eval^s_T(Pre_{a_i}\sigma_{a_1}) = \bot$. Now, by assumption, we have $T, s \models Pre_{a_1}$ and $T, s \models Pre_{a_i}$, since all actions are applicable in state s. Therefore, $T, s \models Pre_{a_1} \wedge Pre_{a_i} \wedge \neg(Pre_{a_i}\sigma_{a_1})$, i.e., $Pre_{a_1} \wedge Pre_{a_i} \wedge \neg(Pre_{a_i}\sigma_{a_1})$ is T-satisfiable, contradicting that a_1 does not affect a_i. Finally, since all actions a_i such that $a_1 < a_i$ are applicable in state $app_{a_1}(s)$, by the induction hypothesis we have that $app_{a_2;\ldots;a_n}(app_{a_1}(s))$ is defined for any ordering $a_2 < \cdots < a_n$ of $A \setminus \{a_1\}$ such that if $a_i < a_j$ then a_i does not affect a_j, and hence so is $app_{a_1;a_2;\ldots;a_n}(s)$ for the ordering we have considered. □

Lemma 2. Let a and b be two simply commuting actions, for a state space S modulo T, such that a does not affect b, and let $s \in S$ be a state such that a and b are applicable in s. Then $app_{\{a,b\}}(s) = app_{a;b}(s)$.

Proof. Let $a = \langle Pre_a, \sigma_a \rangle$ and $b = \langle Pre_b, \sigma_b \rangle$. Since a and b are applicable in s, we have that $app_{\{a,b\}}(s)$ is defined. Moreover, since a does not affect b, by Lemma 1 we have that $app_{a;b}(s)$ is defined.

We conclude by showing that $app_{\{a,b\}}(s)[x] = app_{a;b}(s)[x]$ for every variable x. Recall that $app_{\{a,b\}}(s) = app_{h(\{a,b\})}(s)$, where $h(\{a,b\})$ denotes the happening action for a and b. Now, by definition of application, we have $app_{h(\{a,b\})}(s)[x] = eval^s_T(x\sigma_{h(\{a,b\})})$ and $app_{a;b}(s)[x] = eval^s_T(x\sigma_b\sigma_a)$, for every variable x. On the other hand, since a does not affect b, $Pre_a \wedge Pre_b \wedge \neg(x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$ is T-unsatisfiable. And, since a and b are both applicable in state s, we have $T, s \models Pre_a$ and $T, s \models Pre_b$. Therefore $T, s \models Pre_a \wedge Pre_b \wedge (x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$, and thus $eval^s_T(x\sigma_{h(\{a,b\})}) = eval^s_T(x\sigma_b\sigma_a)$, which lets us conclude. □

Lemma 3. Let a and b be two non-interfering actions, for a state space S modulo T, and let $s \in S$ be a state such that a and b are applicable in s. Then $app_{a;b}(s) = app_{b;a}(s)$.

Proof. Since a and b are non-interfering, then they are simply commuting, and neither a affects b nor b affects a. Then, by Lemma 2, we have $app_{\{a,b\}}(s) = app_{a;b}(s)$, and $app_{\{a,b\}}(s) = app_{b;a}(s)$. □

Lemma 4. Let A be a set of non-interfering actions, for a state space S modulo T, and let $s \in S$ be a state such that all actions in A are applicable in s. Then $app_{a_1;\ldots;a_n}(s)$ is the same state for every total ordering $a_1 < \cdots < a_n$ of A.

Proof. Since actions in A are non-interfering, and applicable in state s, by Lemma 1 we have that $app_{a_1;\ldots;a_n}(s)$ is defined for any total ordering $a_1 < \cdots < a_n$ of A. We conclude by showing that any two consecutive actions in the sequence $a_1; \ldots; a_n$ can be permuted, preserving the final state. Consider any two consecutive actions a_i and a_{i+1} in the sequence $a_1; \ldots; a_n$. Since $app_{a_1;\ldots;a_n}(s)$ is defined, so is $app_{a_1;\ldots;a_i}(s)$, and a_i is applicable in state $app_{a_1;\ldots;a_{i-1}}(s)$ (in case that $i = 1$, let $app_{a_1;\ldots;a_{i-1}}(s)$ denote the state s). Now, since actions in A are non-interfering, by Lemma 1 we have that $app_{a_1;\ldots;a_{i-1};a_{i+1}}(s)$ is also defined, so a_{i+1} is also applicable in state $app_{a_1;\ldots;a_{i-1}}(s)$. Finally, by Lemma 3, it follows that $app_{a_i;a_{i+1}}(app_{a_1;\ldots;a_{i-1}}(s)) = app_{a_{i+1};a_i}(app_{a_1;\ldots;a_{i-1}}(s))$ which, by definition of application, is equivalent to $app_{a_1;\ldots;a_{i-1};a_i;a_{i+1};\ldots;a_n}(s) = app_{a_1;\ldots;a_{i-1};a_{i+1};a_i;\ldots;a_n}(s)$. □

Lemma 5. Let A be a set of simply commuting actions, for a state space S modulo T, such that $|A| \geq 2$. Then, for every action $a \in A$, we have that a and $h(A \setminus \{a\})$ are simply commuting, and $h(\{a, h(A \setminus \{a\})\}) = h(A)$.

Proof. Let $A = \{a_1, a_2, \ldots, a_n\}$, $a = a_1$ and $A' = A \setminus \{a\} = \{a_2, \ldots, a_n\}$. According to the definition of happening action, we have $\sigma_{h(A')} = \cup_{x \in var(S)}\{\sigma_{x,2} \circ \cdots \circ \sigma_{x,n}\}$, where $\sigma_{x,i}$, for i in $2..n$, is the mapping of variable x in the effects of action a_i. Now, since composition of functions is associative, we have that $\sigma_{x,1} \circ (\sigma_{x,2} \circ \cdots \circ \sigma_{x,n}) = \sigma_{x,1} \circ \sigma_{x,2} \circ \cdots \circ \sigma_{x,n}$ for every variable x, being $\sigma_{x,1}$ the mapping of variable x in the effects of action a_1. And, since actions in A are simply commuting, we have that $\sigma_{x,1} \circ \sigma_{x,2} \circ \cdots \circ \sigma_{x,n} = (\sigma_{x,2} \circ \cdots \circ \sigma_{x,n}) \circ \sigma_{x_1}$, which lets us conclude that a and $h(A')$ are simply commuting.

Now, provided that a and $h(A')$ are simply commuting, in order to prove that the happening actions $h(\{a, h(A')\})$ and $h(A)$ are equivalent, we need to show that they have equivalent preconditions and effects. For preconditions, we have $Pre_{h(\{a,h(A')\})} = Pre_a \wedge Pre_{h(A')} = \wedge_{a \in A} Pre_a = Pre_{h(A)}$. For effects, we have $\sigma_{h(\{a,h(A')\})} =$

$\cup_{x\in var(S)}\{\sigma_{x,1}\circ(\sigma_{x,2}\circ\cdots\circ\sigma_{x,n})\}$ which, as seen before, is equivalent to $\cup_{x\in var(S)}\{\sigma_{x,1}\circ\sigma_{x,2}\circ\cdots\circ\sigma_{x,n}\}$. □

Lemma 6. Let a, b and c be three simply commuting actions, for a state space S modulo T. If a affects neither b nor c, then a does not affect the happening action $h(\{b,c\})$.

Proof. Let $a = \langle Pre_a, \sigma_a\rangle$, $b = \langle Pre_b, \sigma_b\rangle$ and $c = \langle Pre_c, \sigma_c\rangle$. We need to prove that

1. $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(Pre_{h(\{b,c\})}\sigma_a)$ is T-unsatisfiable,
2. a and $h(\{b,c\})$ are simply commuting, and
3. $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(x\sigma_{h(\{a,h(\{b,c\})\})} = x\sigma_{h(\{b,c\})}\sigma_a)$ is T-unsatisfiable for every variable $x \in var(S)$.

For condition 1, since $Pre_{h(\{b,c\})} = Pre_b \wedge Pre_c$, we have $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(Pre_{h(\{b,c\})}\sigma_a) = Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg((Pre_b \wedge Pre_c)\sigma_a) = (Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(Pre_b\sigma_a)) \vee (Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(Pre_c\sigma_a))$. Now assume that $Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(Pre_b\sigma_a)$ is T-satisfiable (the other case is analogous). Then $Pre_a \wedge Pre_b \wedge \neg(Pre_b\sigma_a)$ would also be T-satisfiable, contradicting that a does not affect b.

Condition 2 follows directly from Lemma 5.

For condition 3, we proceed by contradiction. Let us assume that $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(x\sigma_{h(\{a,h(\{b,c\})\})} = x\sigma_{h(\{b,c\})}\sigma_a)$ is T-satisfiable for some variable $x \in var(S)$. Then, by definition of happening action, we have $Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(x(\sigma_{x,b}\circ\sigma_{x,c}\circ\sigma_{x,a}) = x(\sigma_{x,b}\circ\sigma_{x,c})\sigma_a)$ is T-satisfiable, where $\sigma_{x,a}$, $\sigma_{x,b}$ and $\sigma_{x,c}$ are the mappings of variable x in the effects of actions a, b and c, respectively. So there exists some assignment s such that $T, s \models Pre_a$, $T, s \models Pre_b$, $T, s \models Pre_c$, and $eval^s_T(x\sigma_{x,b}\sigma_{x,c}\sigma_{x,a}) \neq eval^s_T(x\sigma_{x,b}\sigma_{x,c}\sigma_a)$. This implies the existence of some variable y different from x such that $\sigma_a[y] \neq y$. Moreover, since $\sigma_{x,b}$ and $\sigma_{x,c}$ are substitutions replacing only variable x, y must be a variable in $x\sigma_{x,b}$ or in $x\sigma_{x,c}$ and, necessarily, $eval^s_T(x\sigma_{x,b}\sigma_{x,a}) \neq eval^s_T(x\sigma_{x,b}\sigma_a)$ or $eval^s_T(x\sigma_{x,c}\sigma_{x,a}) \neq eval^s_T(x\sigma_{x,c}\sigma_a)$. But this, together with $T, s \models Pre_a$, $T, s \models Pre_b$ and $T, s \models Pre_c$, contradicts a affecting neither b nor c. □

Lemma 7. Let A be a set of actions, and a an action, for a state space S modulo T, such that the actions in $A \cup \{a\}$ are simply commuting. If a affects none of the actions in A, then a does not affect the happening action $h(A)$.

Proof. Let $A = \{a_1, \ldots, a_n\}$. We proceed by induction on the number of actions n in A. If $n = 1$ then we are trivially done, since $h(A) = a_1$ and, by assumption, a does not affect a_1. If $n \geq 2$, let $A' = A \setminus \{a_1\}$. Then a neither affects a_1 nor the happening action $h(A')$ (by the induction hypothesis). Moreover, since actions in $A \cup \{a\}$ are simply commuting, so are a, a' and $h(A')$. Then, by Lemma 6, we have that a does not affect $h(\{a', h(A')\})$ and, by Lemma 5, $h(\{a', h(A')\}) = h(A)$. □

Theorem 1. Let A be a set of non-interfering actions, for a state space S modulo T, and $s \in S$ a state such that $app_A(s)$ is defined. Then $app_A(s) = app_{a_1;\ldots;a_n}(s)$ for any total ordering $a_1 < \cdots < a_n$ of A.

Proof. By induction on the number of actions n in A. If $n = 1$ then we are trivially done. If $n \geq 2$, then let $A = \{a\} \cup A'$. Since actions in A are non-interfering, then they are simply commuting and a affects none of the actions in A'. Then, by Lemma 7, we have that a does not affect the happening action $h(A')$. Now observe that, since $app_A(s)$ is defined and $Pre_{h(A)} = \bigwedge_{a\in A} Pre_a$, both a and $h(A')$ are applicable in state s. Then, by Lemma 2, we have that $app_{\{a,h(A')\}}(s) = app_{a;h(A')}(s)$. We conclude by showing that $app_A(s) = app_{\{a,h(A')\}}(s)$ and $app_{a;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$ for any total ordering $a_1 < \cdots < a_n$ of A.

Equality $app_A(s) = app_{\{a,h(A')\}}(s)$ holds by Lemma 5. For equality $app_{a;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$, observe that $app_{a;h(A')}(s) = app_{A'}(app_a(s))$. Since actions in A' are non-interfering and $app_{A'}(app_a(s))$ is defined, by the induction hypothesis we have that $app_{A'}(app_a(s)) = app_{a_1;\ldots;a_{n-1}}(app_a(s))$ for any total ordering $a_1 < \cdots < a_{n-1}$ of A', i.e., $app_{a;h(A')}(s) = app_{a;a_1;\ldots;a_{n-1}}(s)$ for any total ordering $a_1 < \cdots < a_{n-1}$ of A'. Finally, since actions in A are non-interfering and all of them are applicable in state s, by Lemma 4 we have that $app_{a;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$ for any total ordering $a_1 < \cdots < a_n$ of A □

∃-Step Plans

Here we generalize the notion of ∃-step plan, proposed in (Dimopoulos, Nebel, and Koehler 1997) and further developed in (Rintanen, Heljanko, and Niemelä 2006), to the setting of Planning modulo Theories. Under the ∃-step semantics, it is not necessary that all actions are non-interfering as long as they can be executed it at least one order, which makes it possible increase the number of parallel actions still further.

Definition 10 (∃-Step Plan)**.** Given a set of actions A and an initial state I, for a state space S modulo T, a ∃-*step plan* for A and I is a sequence $P = \langle A_0, \ldots, A_{l-1}\rangle$ 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

1. $s_0 = I$, and
2. for all $i \in \{0, \ldots, l-1\}$ there is a total ordering $a_1 < \cdots < a_n$ of A_i, such that $app_{a_1;\ldots;a_n}(s_i)$ is defined and equals s_{i+1}.

Instead of requiring that each group A_i of actions can be ordered to any total order, as in ∀-step semantics, in ∃-step semantics it is sufficient that there is one order that maps state s_i to s_{i+1}. Note that under this semantics the successor s_{i+1} of s_i is not uniquely determined solely by A_i, as the successor depends on the implicit ordering of A_i and, hence, the definition has to make the execution $s_0, \ldots, s_l$ explicit.

Theorem 2. Let A be a set of simply commuting actions, for a state space S modulo T, such that, for some total ordering $a_1 < \cdots < a_n$ of A, if $a_i < a_j$ then a_i does not affect a_j, and let $s \in S$ be a state such that $app_A(s)$ is defined. Then $app_A(s) = app_{a_1;\ldots;a_n}(s)$.

Proof. The proof is analogous to the proof of Theorem 1, but without using Lemma 4. We proceed by induction on

the number of actions n in A. If $n = 1$ then we are trivially done. If $n \geq 2$, then let $A' = \{a_2, \ldots, a_n\}$. We have that actions in A are simply commuting and a_1 affects none of the actions in A'. Then, by Lemma 7, we have that a_1 does not affect the happening action $h(A')$. Now observe that, since $app_A(s)$ is defined and $Pre_{h(A)} = \bigwedge_{a \in A} Pre_a$, both a_1 and $h(A')$ are applicable in state s. Then, by Lemma 2, we have that $app_{\{a_1, h(A')\}}(s) = app_{a_1;h(A')}(s)$. We conclude by showing that $app_A(s) = app_{\{a_1,h(A')\}}(s)$ and $app_{a_1;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$.

Equality $app_A(s) = app_{\{a_1,h(A')\}}(s)$ holds by Lemma 5. For equality $app_{a_1;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$, observe that $app_{a_1;h(A')}(s) = app_{A'}(app_{a_1}(s))$. Since actions in A' are simply commuting and $app_{A'}(app_{a_1}(s))$ is defined, by the induction hypothesis we have that $app_{A'}(app_{a_1}(s)) = app_{a_2;\ldots;a_n}(app_{a_1}(s))$ according to the given ordering, and hence $app_{a_1;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$. □

Checking Interference with SMT

We can exactly check the proposed notion of interference, according to Definitions 3, 4 and 7, by means of checking the satisfiability of some SMT formulas at compile time.

The following example is taken from the *Planes* domain, described in the Empirical Evaluation section. The problem consists in transporting people between several cities using planes, with a limited number of seats. The considered actions are `board` and `fly`. Boarding is limited by seat availability, and a plane can only fly if it is transporting somebody. If we consider action a as

$$\begin{aligned}&\text{board\_person1\_plane1\_city1} = \\ &\quad\langle \text{seats\_plane1} > \text{onboard\_plane1} \wedge \\ &\qquad\text{at\_person1\_city1} \wedge \text{at\_plane1\_city1}, \\ &\quad\{\text{at\_person1\_city1} \mapsto \bot,\ \text{in\_person1\_plane1} \mapsto \top, \\ &\qquad\text{onboard\_plane1} \mapsto \text{onboard\_plane1} + 1\}\rangle\end{aligned}$$

and action b as

$$\begin{aligned}&\text{fly\_plane1\_city1\_city2} = \\ &\quad\langle \text{onboard\_plane1} > 0 \wedge \text{at\_plane1\_city1}, \\ &\quad\{\text{at\_plane1\_city1} \mapsto \bot,\ \text{at\_plane1\_city2} \mapsto \top\}\rangle\end{aligned}$$

then most planners, checking interference syntactically, would determine interference, since a modifies the `onboard_plane1` variable and b uses this variable in its precondition. However, according to Definition 7, it can be seen that a does not affect b, since:

1. $Pre_a \wedge Pre_b \wedge \neg(Pre_b\sigma_a)$ is T-unsatisfiable, because the precondition of b, $\text{onboard\_plane1} > 0 \wedge \text{at\_plane1\_city1}$, cannot be falsified by the effects of a, $\{\text{at\_person1\_city1} \mapsto \bot,\ \text{in\_person1\_plane1} \mapsto \top,\ \text{onboard\_plane1} \mapsto \text{onboard\_plane1} + 1\}$,
2. a and b are simply commuting, and
3. $Pre_a \wedge Pre_b \wedge \neg(x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$ is T-unsatisfiable for all variables x.

The first check can be modelled in the SMT-LIB language (Barrett, Stump, and Tinelli 2010) as follows:

```
;; declaration of problem variables.
(declare-fun at_person1_city1 () Bool)
(declare-fun at_plane1_city1 () Bool)
(declare-fun seats_plane1 () Int)
(declare-fun onboard_plane1 () Int)

;; preconditions of actions "board" and "fly"
(assert (and (> seats_plane1 onboard_plane1)
             at_person1_city1 at_plane1_city1))

(assert (and (> onboard_plane1 0)
             at_plane1_city1))

;; negated precondition of fly after board
(assert (not (and (> (+ onboard_plane1 1) 0)
                  at_plane1_city1)))
(check-sat)
```

Note that in the negated precondition of `fly`, we are replacing each variable by the term which represents its value after the execution of `board`, i.e., we replace onboard_plane1 by $\text{onboard\_plane1} + 1$. A negative answer should be obtained from the SMT solver.

The check of simply commutativity would consist in checking for all variables commonly modified by the two actions, if the effects can be commuted. In the example, there are no common variables modified by both actions. Hence, suppose we are checking whether two arbitrary assignments $\{x \mapsto exp_1\}$ and $\{x \mapsto exp_2\}$ commute. According to Definition 3, this would consist in checking whether $\neg(exp_2\{x \mapsto exp_1\} = exp_1\{x \mapsto exp_2\})$ is T-satisfiable. A negative answer would imply T-unsatisfiability of this negation and, hence, commutativity of the assignments.

The third check can be implemented analogously by means of satisfiability checks.

Encodings

In this section we propose two different encodings for *planning as SMT*, as a particular case of PMT. We generalize Rintanen's (Rintanen 2009) encoding for *planning as SAT* to include non-Boolean variables, with the idea of using an off-the-shelf SMT solver to solve the problem.

The given encodings are valid for any theory T under a 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. In the SMT-LIB standard (Barrett, Stump, and Tinelli 2010), QF_LIA stands for the logic of Quantifier-Free Boolean formulas, with Linear Integer Arithmetic constraints, and similarly QF_LRA for the case of reals. These logics have a good compromise between expressivity and performance, and so are a natural choice for numeric planning as SMT.

SMT Encoding

Let $\pi = \langle S, A, I, G\rangle$ be planning problem modulo T, for a 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 same 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

$$\{x \mapsto \top\}^t \stackrel{def}{=} x^{t+1}$$
$$\{x \mapsto \bot\}^t \stackrel{def}{=} \neg x^{t+1}$$
$$\{x \mapsto s\}^t \stackrel{def}{=} (x^{t+1} = s^t)$$

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

$$(\{x \mapsto s\} \cup \mathit{Eff})^t \stackrel{def}{=} \{x \mapsto s\}^t \wedge \mathit{Eff}^t$$
$$\emptyset^t \stackrel{def}{=} \top$$

where s is a term (either Boolean or not) and *Eff* is a set of assignments.

The constraints are as follows. For each time step t, execution of an action implies that its precondition is met:

$$a^t \rightarrow \mathit{Pre}_a^t \qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \tag{1}$$

If the action is executed, each of its effects will hold at the next time step:

$$a^t \rightarrow \mathit{Eff}_a^t \qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \tag{2}$$

Finally, we need explanatory axioms to express the reason of a change in state variables. For each variable x in $var(S)$:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{\substack{\forall a = \langle Pre_a, Eff_a \rangle \in A \\ \text{such that } \exists \{x \mapsto s\} \in Eff_a}} a^t \tag{3}$$

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.

The previous constraints are complemented as follows depending on the type of parallelism we wish:

Sequential Plans We can achieve a sequential plan by imposing an *exactly one* constraint on the action variables at each time step.

$\forall$-step Plans According to Definition 9, in $\forall$-step plans we require the possibility of ordering the set of actions planned at each time step to any total order. Therefore, for each time step t, we simply add a mutex clause for every pair of interfering actions a_i and a_j:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ or } a_j \text{ affects } a_i \tag{4}$$

$\exists$-step Plans According to Definition 10, in $\exists$-step plans there must only exist a total ordering of parallel actions resulting in a valid sequential plan. A basic form of guaranteeing this is to take an arbitrary total ordering $<$ on the actions, and forbid the parallel execution of two actions a_i and a_j such that a_i affects a_j only if $a_i < a_j$:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ and } a_i < a_j \tag{5}$$

It is not difficult to see that the encoding presented up to this point is correct for sequential plans, but it not adheres to the parallel plan semantics of Definition 6. If two actions planned at the same time modify a same variable, two different situations can arise. On the one hand, if the assignments are not equivalent, then the SMT formula encoding the planning problem will become unsatisfiable. Although this is right for the Boolean case, it is more subtle for other theories, where effects can be cumulative. For example, two assignments $\{x \mapsto x + 1\}$ and $\{x \mapsto x + 2\}$ would result into subformulas $x^{t+1} = x^t + 1$ and $x^{t+1} = x^t + 2$ which, together, are unsatisfiable. This, in practice, would rule out many parallel plans. On the other hand, if assignments were equivalent, then all but one would become redundant in the SMT formula. Then, the formula would possibly be satisfiable but, in this case, solutions would not adhere to the semantics given in Definition 6, where effects of actions planned at the same time are composed. A simple way of overcoming this problem could be to forbid the parallel execution of actions modifying a same non-Boolean variable, but this would rule out the parallelization of actions with cumulative effects. For this reason, a finer approach is described in the next section.

Chained SMT Encoding

Here we present an encoding which builds onto the former in order to add support for cumulative effects in parallel plans.

Let N be the set of non-Boolean variables from $var(S)$. For each action $a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle$ and each variable $n \in N$, let $\mathit{Eff}_{a,n}$ be the assignment $\{n \mapsto exp\} \in \mathit{Eff}_a$, or the empty set if there is no such assignment. For each $n \in N$, let $A_n = \{a \mid a \in A \wedge \mathit{Eff}_{a,n} \neq \emptyset\}$, i.e., the set of actions that modify variable n.

Constraints (2) are split and rewritten as follows, in order to take into account a possible "chain of assignments" on each variable $n \in N$. First of all, we remove the effects on variables $n \in N$ such that $|A_n| > 1$, i.e., those that are modified by more than one action.

$$a^t \rightarrow (\mathit{Eff}_a \setminus \cup_{n \in N, |A_n| > 1} \{\mathit{Eff}_{a,n}\})^t \qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \tag{6}$$

Then, for each variable $n \in N$ such that $|A_n| > 1$, and for each time step t, the following constraints are introduced, using additional variables $\zeta_{n,0}^t, \ldots, \zeta_{n,|A_n|}^t$ of the type of n, and considering an enumeration $a_1, \ldots, a_{|A_n|}$ of the actions in A_n:

$$n^t = \zeta_{n,0}^t \tag{7}$$

$$\begin{aligned} a_i^t &\rightarrow \mathit{Eff}_{a_i,n}^t \{n^{t+1} \mapsto \zeta_{n,i}^t, n^t \mapsto \zeta_{n,i-1}^t\} \\ \neg a_i^t &\rightarrow \zeta_{n,i}^t = \zeta_{n,i-1}^t \qquad \forall a_i \in a_1, \ldots, a_{|A_n|} \end{aligned} \tag{8}$$

$$n^{t+1} = \zeta_{n,|A_n|}^t \tag{9}$$

Example 6. Let $A = \{a_1, a_2\}$, with actions $a_1 = \langle \top, \{x \mapsto x + 1, y \mapsto 0\}\rangle$ and $a_2 = \langle \top, \{x \mapsto x + 2\}\rangle$. Then $A_x = \{a_1, a_2\}$ and $A_y = \{a_1\}$. For time step t, we would add variables $\zeta^t_{x,0}$, $\zeta^t_{x,1}$, $\zeta^t_{x,2}$ and the following constraints:

$$a^t_1 \rightarrow y^{t+1} = 0 \qquad a^t_2 \rightarrow \top \text{ (tautology)}$$
$$x^t = \zeta^t_{x,0}$$
$$a^t_1 \rightarrow \zeta^t_{x,1} = \zeta^t_{x,0} + 1 \qquad \neg a^t_1 \rightarrow \zeta^t_{x,1} = \zeta^t_{x,0}$$
$$a^t_2 \rightarrow \zeta^t_{x,2} = \zeta^t_{x,1} + 2 \qquad \neg a^t_2 \rightarrow \zeta^t_{x,2} = \zeta^t_{x,1}$$
$$x^{t+1} = \zeta^t_{x,2}$$

Empirical Evaluation

In this section we evaluate the impact of the proposed notion of interference on the length of parallel plans, using $\exists$-step semantics. Experiments have been performed using both syntactic an semantic checks of interference at compile time. The presented SMT encoding has been used in both cases. For the case of semantic checks, we have additionally considered the chained SMT encoding. These executions are noted as SYN, SEM, and SEM+C, respectively, in Tables 1 and 2. In syntactic checking we forbid concurrent assignment or assignment and inspection to the same numeric variable. Semantic checks are the ones we have introduced, by means of calls to a SMT solver.

Experiments have been run on 8GB Intel® Xeon® E3-1220v2 machines at 3.10 GHz, using RanTanPlan (Bofill, Espasa, and Villaret 2015) with Yices (Dutertre and De Moura 2006) v2.3.0 as back-end SMT solver, with the QF_LIA logic and a two hours timeout. For the sake of completeness, we compare the performance of our implementation with the numeric planner NumReach/SAT (Hoffmann et al. 2007) using MiniSAT 2.2.0 (column NR1), and NumReach/SMT using Yices v2.3.0 (column NR2).

Five domains are considered: the numeric versions of *ZenoTravel*, *DriverLog* and *Depots*, the real-life challenging *Petrobras* domain, and a crafted domain called *Planes*.

ZenoTravel and *DriverLog* are some of the domains in the literature with a higher numeric interaction between actions. Domains like *Rovers* or *Settlers* have been excluded because they are too big to show meaningful results with the encoding at hand and the chosen timeout.

The *Petrobras* domain models a real-life problem of resource-efficient transportation of goods from ports to petroleum platforms. It was proposed as a challenge problem at the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012). Instances were modelled from (Toropila et al. 2012).

Due to the limited numeric interactions between actions in the domains found in the literature, we additionally propose a new domain called *Planes* which is created from *ZenoTravel*, by adding some plausible numeric constraints, in order to help us demonstrate the benefits from checking interference between actions semantically.[2]

Table 1 shows the number of instances solved by each approach. Checking interference semantically and using the chained SMT encoding is best in *Petrobras*, *ZenoTravel* and *Planes*, while NumReach/SAT is best in *Depots* and *DriverLog*. The big gap in the number of solved instances in *Depots* is twofold: lack of intrinsic parallelism in the domain, and being the perfect scenario for the reachability approach of NumReach.

Domain	NR1	NR2	SYN	SEM	SEM+C
Depots	**13**	13	5	5	5
Petrobras	3	3	5	6	**7**
Planes	5	8	8	8	**8**
ZenoTravel	13	14	13	13	**15**
DriverLog	**18**	12	14	14	15
Total	**52**	50	45	46	50

Table 1: Total number of instances of each domain solved by each approach. For each domain, the approach solving more instances is marked in bold. In case of draw, the faster is marked.

Domain	NR	SYN	SEM	SEM+C
Depots (5)	54	52	52	**51**
Petrobras (3)	19	12	12	**11**
Planes (4)	90	82	62	**45**
Zenotravel (13)	97	69	69	**42**
DriverLog (12)	104	85	84	**63**
Total (37)	364	300	279	**212**

Table 2: Sum of the number of time steps of the plans found, restricted to commonly solved instances. First column shows, in parenthesis, the number of instances solved by all approaches. NumReach uses the same parallelism approach when using different background solvers, so only one column is included. The winning approach is shown in bold.

Table 2 shows the sum of the number of time steps of the plans found, for commonly solved instances. Note that the domains where our implementation solves more instances are also the ones that exhibit more gains in parallelism. Note also the significant reduction in time steps from the syntactic approach to the semantic approach with the chained SMT encoding, especially in the *Planes* domain.

Conclusion

The main contribution of this paper is the formalization of an elegant solution to the problem of determining interference between actions in PMT, which can moreover be implemented as a set of satisfiability checks of SMT formulas. We have introduced a new relaxed semantics for the parallel execution of actions, and formalized a semantic notion of interference, that are suitable for both $\forall$-step and $\exists$-step plans in the context of PMT.

We have argued why the presented proposal is better than purely syntactic ones, and provided empirical evidence of its usefulness by showing a significant improvement in parallelism in some domains. The presented semantic checks can be done independently of the underlying planning system.

[2]Detailed experiments and the Planes domain can be found at http://ima.udg.edu/Recerca/lap/rantanplan/rantanplan.html

Acknowledgements

All authors supported by the Spanish Ministry of Economy and Competitiveness through the projects TIN2012-33042 and TIN2015-66293-R. Joan Espasa also supported by UdG grant (BR 2013).

References

Barrett, C.; Stump, A.; and Tinelli, C. 2010. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`.

Bofill, M.; Espasa, J.; and Villaret, M. 2015. The RANTANPLAN Planner: System Description. In *ICAPS-15 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-15)*, 1–10.

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding Planning Problems in Nonmonotonic Logic Programs. In *Recent Advances in AI Planning, Fourth European Conference on Planning (ECP'97)*, volume 1348 of *LNCS*, 169–181.

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

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multicriteria plan quality. *Artificial Intelligence* 172(8):899–944.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning Modulo Theories: Extending the Planning Paradigm. In *Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 65–73.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Sixth International Conference on Artificial Intelligence, Planning and Scheduling (AIPS 2002)*, 303–312.

Hoffmann, J.; Gomes, C. P.; Selman, B.; and Kautz, H. A. 2007. SAT Encodings of State-Space Reachability Problems in Numeric Domains. In *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1918–1923.

Kautz, H. A., and Walser, J. P. 1999. State-space planning by integer optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, 526–533.

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

Rintanen, J. 2009. Planning and SAT. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. 483–504.

Toropila, D.; Dvorak, F.; Trunda, O.; Hanes, M.; and Barták, R. 2012. Three Approaches to Solve the Petrobras Challenge: Exploiting Planning Techniques for Solving Real-Life Logistics Problems. In *IEEE 24th International Conference on Tools with Artificial Intelligence, (ICTAI 2012)*, 191–198.