The RANTANPLAN Planner: System Description

Miquel Bofill and **Joan Espasa** and **Mateu Villaret**
Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, Spain

Abstract

RANTANPLAN is a numeric planning solver that takes advantage of recent advances in SMT. It extends reduction to SAT approaches with an easy and efficient handling of numeric fluents using background theories. In this paper we describe the design choices and features of RANTANPLAN, especially, how numeric reasoning is integrated in the system. We also provide experimental results showing that RANTANPLAN is competitive with existing exact numeric planners.

Introduction

The problem of planning, in its most basic form, consists in finding a sequence of actions that allow to reach a goal state from a given initial state. Although initially considered a deduction problem, it was rapidly seen that it could be addressed by looking at it as a satisfiability (model finding) problem (Kautz and Selman 1992). Many (incomplete) heuristic methods can be found in the literature to efficiently deal with this problem, most of them oriented towards finding models. Exact methods were ruled out at the beginning due to their inefficiency. However, in (Kautz, McAllester, and Selman 1996) it was shown that modern off-the-shelf SAT solvers could be effectively used to solve planning problems. In recent years, the power of SAT technology has been leveraged to planning (Rintanen 2012), making reduction into SAT competitive with heuristic search methods.

Although a lot of work has been devoted to the encoding of plans in propositional logic, only a few works can be found in the literature on satisfiability based approaches to planning in domains that require numeric reasoning. This is probably due to the difficulty of efficiently handling at the same time numeric constraints and propositional formulas. Among the few works dealing with planning with resources are (Hoffmann 2003; Kautz and Walser 1999; Gerevini, Saetti, and Serina 2008; Hoffmann et al. 2007). There have also been some works using constraint and logic programming (Dovier, Formisano, and Pontelli 2010; Barták and Toropila 2010). However, the advances in satisfiability modulo theories (SMT) (Barrett et al. 2009) in the last years make worth considering this alternative. With RANTANPLAN we demonstrate that with SMT one can elegantly handle numeric reasoning inside any PDDL domain, thanks to the integration of various background theories with a SAT solver.

As the number of variables, and hence the search space, rapidly grows with the number of time steps considered, a key idea to improve the performance of SAT-based planners is to consider the possibility of executing several actions at the same time, i.e., the notion of parallel plans. 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. Nevertheless, in SAT-based planning, parallel plans are not intended to represent true parallelism in time, and it is usually required that a sequential plan can be built from a parallel plan in polynomial time. Two main types of parallel plans are considered: $\forall$-step plans, and $\exists$-step plans. In $\forall$-step plans, any ordering of parallel actions must result in a valid sequential plan. In $\exists$-step plans, there must exist a total ordering of parallel actions resulting in a valid sequential plan. We refer the reader to (Rintanen 2009; Rintanen, Heljanko, and Niemelä 2006) for further details. RANTANPLAN supports $\forall$ and $\exists$-step plans, using various different encodings.

To ensure that a parallel plan is sound, it is necessary that all actions proposed to be executed at the same time do not interfere. Different notions of interference have been defined, some more restrictive, some more relaxed. But, as far as we know, for efficiency reasons, potential interference between action is always determined statically, i.e., independently of any concrete state, hence in a fairly restrictive way. Moreover, very few works deal with the notion of incompatibility of actions in planning with resources, most of them with rather syntactic or limited semantic approaches (Kautz and Walser 1999; Fox and Long 2003; Gerevini, Saetti, and Serina 2008) RANTANPLAN incorporates a novel method for determining interference between actions at compile time, using an SMT solver as an oracle.

Summing up, RANTANPLAN is a numeric planner based on planning as satisfiability, which translates PDDL problems into SMT formulas. It supports various types of parallelism, using a novel notion of interference. Experimental results show that it is competitive with other exact numeric planners and strictly better in non-trivial numeric domains.

Related Work

The pioneering work of LPSAT (Wolfman and Weld 1999) on planning with resources can indeed be considered one of

the precursors of SMT, as the basic ideas of SMT (Boolean abstraction, interaction of a SAT solver with a theory solver, etc.) were already present in it.

A comparison between SAT and SMT based encodings for planning in numeric domains can be found in (Hoffmann et al. 2007). In the SAT approach, the possible values of numeric state variables is approximated, by generating a set of values $D_t(v)$ for every numeric variable v, so that every value that v can have after t time steps is contained in $D_t(v)$. These finite domains then serve as the basis for a fully Boolean encoding, where atoms represent numeric variables taking on particular values. With respect to SMT, where numeric variables and expressions are first class citizens, the authors argue that the expressivity of the SMT language comes at the price of requiring much more complex solvers than for SAT and, for this reason, their SAT-based method is very efficient in domains with tightly constrained resources, where the number of distinct values that a numeric variable can take is small.

Other approaches, related to SMT to some amount as well, have been developed more recently. In (Belouaer and Maris 2012), a set of encoding rules is defined for spatio-temporal planning, taking SMT as the target formalism. On the other hand, in (Gregory et al. 2012) a modular framework, inspired in the architecture of lazy SMT, is developed for planning with resources. We compare with some of these approaches in the experimental evaluation section.

Preliminaries

A numeric planning problem is defined as a tuple $\langle V, P, A, I, G\rangle$ where V is a set of numeric variables, P is a set of propositions (or Boolean variables), A is a set of actions, I is the initial state and G is a formula over $V \cup P$ that any goal state must satisfy.

A state is a total assignment to the variables. Actions are formalized as pairs $\langle p, e\rangle$, where p are the preconditions and e the effects. More formally, p is a set of Boolean expressions over $V \cup P$, while e is a set of (conditional) effects of the form $f \Rightarrow d$, where f is a Boolean expression over $V \cup P$ and d is a set of assignments. An assignment is a pair $\langle v, exp\rangle$, where v is a variable and exp is an expression of the corresponding type. For example, increasing a variable v by one is represented by the pair $\langle v, v+1\rangle$, indicating that $v+1$ is the value that v will hold in the next state. Unconditional effects are represented by setting $f = true$.

The active effects of an action $a = \langle p, e\rangle$ in a state s are $\cup_{f \Rightarrow d \in e}\{d \mid s \models f\}$. An action $a = \langle p, e\rangle$ is executable in a given state s if $s \models p$ and the active effects of a in state s are consistent, i.e., we do not have $exp \neq exp'$ for any variable $v \in V \cup P$ and assignments $\langle v, exp\rangle$ and $\langle v, exp'\rangle$ in the active effects.

The state resulting from executing action a in state s is denoted by $apply(a, s) = s'$. The new state s' is defined by assigning new values to the variables according to the active effects, and retaining the values of the variables that are not assigned values by any of the active effects.

A sequential plan of length n for a given planning problem $\langle V, P, A, I, G\rangle$ is a sequence of actions $a_1; a_2; \ldots; a_n$ such that $apply(a_n \ldots apply(a_2, apply(a_1, I)) \ldots) \models G$.

A parallel plan of length n can be defined similarly to a sequential plan. Instead of having a sequence of actions, we have a sequence of sets of actions $\sigma_1; \sigma_2; \ldots; \sigma_n$ such that $order(\sigma_1) \oplus order(\sigma_2) \oplus \cdots \oplus order(\sigma_n)$ is a sequential plan, where $order(\sigma_i)$ is an ordering function which transforms the set σ_i into a sequence of actions, and $\oplus$ denotes the concatenation of sequences. Actions in the same set σ_i are said to occur in parallel.

The notion of parallelism of a $\forall$-step plan is defined as the possibility of ordering the actions of each set to any total ordering, i.e., no two actions a, a' in each σ_i are interfering (e.g., executing a neither falsifies the precondition of a' nor changes any of its active effects, and vice versa).

The $\exists$-step semantics weakens the $\forall$-step requirements, by only requiring the existence of some correct ordering of the actions that results in a valid sequential plan.

In the planning as SAT approach, a planning problem is solved by considering a sequence of formulas ϕ_0, ϕ_1, ϕ_2, ..., where ϕ_i encodes the feasibility of a plan that allows to reach a goal state from the initial state in i steps. The solving procedure proceeds by testing the satisfiability of ϕ_0, ϕ_1, ϕ_2, and so on, until a satisfiable formula ϕ_n is found. It is a matter of the encoding whether one or various (non interfering) actions are executed at each step.

Framework

RANTANPLAN supports a fragment of PDDL which is close to general numeric PDDL 2.1, excluding the temporal extensions and metric optimizations. In terms of requirements, we consider typing, numeric and object fluents, equality as built-in predicate, universally quantified and negative preconditions, and conditional effects.

With respect to numeric effects, we consider $assign(x, exp)$, $increase(x, exp)$ and $decrease(x, exp)$, where exp is any closed formula over linear integer (or real) arithmetic. With respect to preconditions and conditions of numeric effects, we assume that the restrictions imposed on numeric fluents take the same form as exp.

System Architecture

The structure of the RANTANPLAN system is represented in Figure 1. It receives a PDDL instance and domain and parses it. A preprocessing step is then applied, where PDDL's `forall` constructs are expanded, static functors are identified, and precomputable substitutions and arithmetic operations are carried out.

To encode the formulas ϕ_0, ϕ_1, ϕ_2, ..., one of the two encodings described in the following sections (QF_LIA or QF_UFLIA) is carried out, transforming the PDDL problem to a pure SMT problem. Then the problem is iteratively solved, using the chosen SMT Solver as a black box.

A key aspect of the planner is the detection of interferences between parallel actions at compile time, by means of calls to a SMT Solver. In case the user demands a parallel plan, a disabling graph is computed. By *disabling graph* we refer to a directed graph, where nodes are the grounded actions from the planning problem and an edge exists from action a to action a' if the execution of a can affect a' (forbid

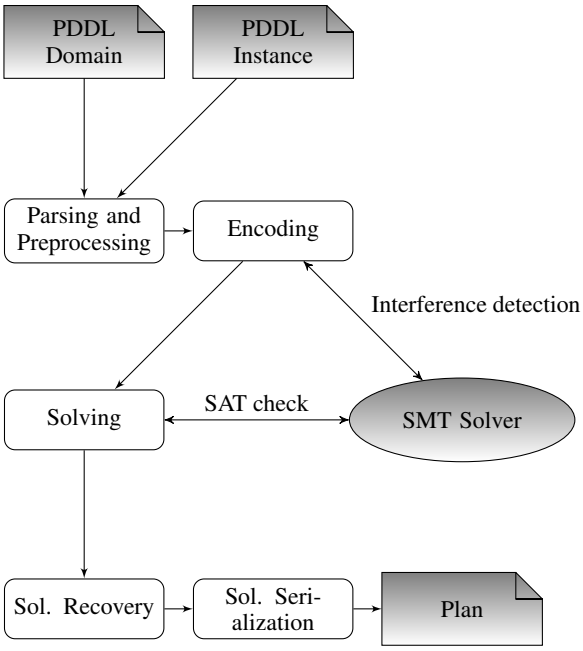

Figure 1: Basic architecture and solving process of the RANTANPLAN solver

its execution or change its active effects) (Rintanen, Heljanko, and Niemelä 2006). This graph is used, depending on the notion of parallelism chosen, to encode the necessary constraints restricting which actions can be carried out at the same time step. In particular, the solver supports:

- A sequential encoding, achieved by using an *at least one* and an *at most one* constraint on the possible actions.
- The $\forall$-step semantics, using the quadratic encoding in (Rintanen, Heljanko, and Niemelä 2006)
- The $\exists$-step semantics, with two encodings. Using the quadratic encoding in (Rintanen, Heljanko, and Niemelä 2006), and a linear-size encoding based on a fixed ordering of operators, also in (Rintanen, Heljanko, and Niemelä 2006).

The system supports solving via API or plain text file using the Yices SMT solver v1.0.38 and the Z3 4.3.2 SMT solver. Once a solution has been found, then it is finally retrieved and serialized. In the following subsections, the relevant aspects of the RANTANPLAN solver are explained in more detail.

QF_LIA Encoding

Numeric planning problems with linear integer arithmetic expressions naturally fall into the QF_LIA logic. 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. This logic has a good compromise between expressivity and performance, and is the natural choice for this problem.

We generalized Rintanen's (Rintanen 2009) encoding of planning as SAT to include numeric variables as follows.

For each time step, every ground instance of a PDDL predicate and action is mapped to a Boolean variable, and every ground instance of a PDDL function is mapped to an integer variable. For instance, a predicate stating the position of an aircraft such as `at(?a - aircraft, ?c - city)`, with three cities `c1`, `c2` and `c3`, and two planes `p1` and `p2`, will result into six ground instances `at(p1,c1)`, ..., `at(p2,c3)`, that will be mapped to six Boolean variables $at^t_{p1,c1}$, ..., $at^t_{p2,c3}$ for each time step t. Following the same example, being `at(?o - aircraft) - city` an object fluent, the mapping would result into two integer variables at_{p1}, at_{p2} with the domains being the possible cities `c1`, `c2` and `c3` (these are internally mapped into three distinct integers). Note that thanks to the SMT language, we can get a more compact encoding of states in the presence of object fluents than using a plain SAT approach. The Boolean variables resulting from actions will be used to denote what action is executed at each time step, and with which parameters. The Boolean and integer variables resulting from grounding the predicates and functions, respectively, will constitute the state variables. A superscript t is used to differentiate the variables at each time step.

Given a formula ϕ, by ϕ^t we denote the same formula ϕ where all integer variables x have been replaced by x^t. For the case of assignments, we define:

$$\begin{aligned}
\langle x, true\rangle^t &\stackrel{def}{=} x^t \\
\langle x, false\rangle^t &\stackrel{def}{=} \neg x^t \\
\langle x, k\rangle^t &\stackrel{def}{=} (x^t = k) \\
\langle x, x+k\rangle^t &\stackrel{def}{=} (x^t = x^{t-1} + k) \\
\langle x, x-k\rangle^t &\stackrel{def}{=} (x^t = x^{t-1} - k)
\end{aligned}$$

For each ground[1] action $a = \langle p, e\rangle$, we have the following constraints. First, its execution during time step t implies that its precondition is met:

$$a^t \rightarrow p^t \qquad \forall a = \langle p, e\rangle \in A \tag{1}$$

Also, each of its conditional effects will hold at the next time step if the corresponding condition holds:

$$(a^t \wedge f^t) \rightarrow d^{t+1} \quad \forall a = \langle p, e\rangle \in A, \forall f \Rightarrow d \in e \tag{2}$$

Here we view sets d of literals as conjunctions of literals. Recall also that unconditional effects will have *true* as condition f.

Second, we need explanatory axioms to express the reason of a change in state variables. For each variable x in $V \cup P$:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{a=\langle p,e\rangle \in A} \left(a^t \wedge (EPC_x(a))^t\right) \tag{3}$$

[1]By a ground action $\langle p, e\rangle$ we refer to an action where p and e are built on the state variables that result from grounding a PDDL model, as explained above.

where, given an action $a = \langle p, e \rangle$ and a variable x,

$$EPC_x(a) = \bigvee_{f \Rightarrow d \in e} \{f \mid d \text{ contains an assignment for } x\}$$

that is, the *effect precondition* for the modification of x in action a, where the empty disjunction is defined as $false$. For Boolean variables, the expression $x^t \neq x^{t+1}$ can be written as $(x^t \wedge \neg x^{t+1}) \vee (\neg x^t \wedge x^{t+1})$.

Interference Between Actions

As said in the introduction, a key concept in parallel plans is the notion of interference between actions. This issue has been carefully considered by Rintanen et al. (Rintanen, Heljanko, and Niemelä 2006) in the setting of planning as SAT. Given a disabling graph, where an edge exists from action a to action a' if the execution of a can affect a', we know for example that the simultaneous execution of all actions pertaining to a strongly connected component is not possible, as given all possible orderings of actions, all of them contain a cycle (and thus they cannot be serialized).

Note that acyclicity is a sufficient but not necessary condition for a set of actions to be executable in some order, since disabling graphs are computed independently of any state.

In (Rintanen 2009), an action a_1 is defined to *affect* another action a_2 if a_1 may prevent the execution of a_2 or change its active effects, and two actions a_1 and a_2 are considered to *interfere* if a_1 affects a_2 or a_2 affects a_1. In $\forall$-step plans, where all possible serializations must be valid, no two interfering actions can occur in parallel. In the more relaxed notion of parallelism of $\exists$-step plans, where it is only required that no action affects a later one in some total ordering, often much more parallelism is allowed in practice. For efficiency reasons, typically syntactic (rather than semantic) restrictions are imposed on parallel actions. For example, in (Rintanen 2009), where only Boolean variables are considered, $a_1 = \langle p_1, e_1 \rangle$ is determined to affect $a_2 = \langle p_2, e_2 \rangle$ if, for some variable a,

1. a is set to $true$ in d_1 for some $f_1 \Rightarrow d_1 \in e_1$, and a occurs negatively in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in e_2$, or
2. a is set to $false$ in d_1 for some $f_1 \Rightarrow d_1 \in e_1$, and a occurs positively in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in e_2$.

That is, a_1 affects a_2 if a_1 can impede the execution of a_2, or change its effects. Note that this is not a symmetric relation.

This is a fully syntactic check which can be used to establish sufficient although not necessary conditions for finding serializable parallel plans. We can observe that interference between effects is not considered. This is because, in the case two actions have contradictory effects, any formula encoding a plan with those two actions running in parallel will become unsatisfiable.

The previous approach could be naively generalized to the case of numeric variables as follows: an action $a_1 = \langle p_1, e_1 \rangle$ *affects* an action $a_2 = \langle p_2, e_2 \rangle$ if, for some variable x, x is modified in d_1 for some $f_1 \Rightarrow d_1 \in e_1$, and x occurs in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in e_2$.

Performing only syntactic checks like the previous seems too much restrictive for numeric variables, even in the case that we determine interference at compile time, i.e., independently of any concrete state. For this reason, we propose a new idea, currently submitted for review (Bofill, Espasa, and Villaret 2015), which is to use SMT technology to perform semantic checks of interference at compile time, in order to increase the amount of parallelization of numeric plans.

Our method is independent of any test suite and does not require any special purpose algorithm, as it relies on encoding the possible interference situations between pairs of actions as SMT formulas and checking their satisfiability, by calling an SMT solver, at compile time. For example, an important difference with the purely syntactic definition of interference of (Rintanen 2009) is that we include the preconditions of the actions in the semantic checks. More precisely, two actions can occur in parallel only if their preconditions can be satisfied simultaneously, regardless of the variables they contain. This way, we are able to avoid many "false positive" interference relationships.

All in all, we obtain a much more fine-grained notion of interference, that we expect will help to increase the parallelization of actions. Note that the interference relationships determined semantically will always be a subset of the interference relationships determined syntactically. Interestingly, we will be using an SMT solver both at compile time, as an oracle to predict interference relationships, and at solving time.

For efficiency reasons, to perform the interference checks we do not consider grounded actions, but the original actions in the PDDL model. Since now actions are not instantiated, we need to unify the parameters of the same type in the actions for which we check interference.

Imagine we have two actions, say `move(?d - ship ?a ?b - location)` and `dock(?e - ship ?c - location)`. We will be interested to know, for example, if the actions interfere in the case that `?d` and `?e` are the same ship. Or in the case that locations `?a` and `?c` are the same, etc. In conclusion, we must consider all different possible equality and disequality relationships between parameters of the same type, to find out in which cases one action can interfere with another action.

To accomplish this task we group all the parameters of the two considered actions by its most general declared type. Following the previous example, in total we have three parameters `a`, `b` and `c` of the type `location` and two parameters `d` and `e` of the type `ship`. Therefore, we need to check interference in the following situations:

```
a = b = c, d = e
a = b, b != c, d = e
a = c, b != c, d = e
a != b, b = c, d = e
...
```

Internally the parameters are substituted by equal or different integers according to the generated constraints, and the formulas encoding incompatibility are checked for satisfiability. These consistency checks can be done in a reasonable time with an SMT solver, and the amount of par-

allelism achieved is significantly higher than with syntactic approaches. To illustrate the situations where this notion of interference is especially accurate, consider the following example. The problem consists in transporting people between cities using planes. Each plane has a limited number of seats and a given fuel capacity. The actions on this domain are `fly`, `board`, `debark` and `refuel`. We focus on the `fly` and `board` actions. A plane can only fly if it is transporting somebody and it has enough fuel to reach its destination, and boarding is limited by seat availability:

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)

 :precondition (and (at ?a ?c1)
                (> (onboard ?a) 0)
                (>= (fuel ?a)
                    (distance ?c1 ?c2)))

 :effect (and (not (at ?a ?c1))
          (at ?a ?c2)
          (decrease (fuel ?a)
                    (distance ?c1 ?c2))))

(:action board
 :parameters (?p - person
              ?a - aircraft
              ?c - city)

 :precondition (and (at ?p ?c)
                (at ?a ?c)
                (> (seats ?a) (onboard ?a)))

 :effect (and (not (at ?p ?c))
          (in ?p ?a)
          (increase (onboard ?a) 1)))
```

The syntactic notion of interference would determine interference between `fly` and `board`, since `board` modifies the `onboard` function (number of passengers) and `fly` checks the value of this function in its precondition. On the contrary, with the semantic technique, we would find out that there is no interference at all, since it is impossible that the preconditions of `board` and `fly` were true at the same time, and after executing `board` the precondition of `fly` became false. Note that the precondition of `fly` requires `(> (onboard ?a) 0)` and the effect `(increase (onboard ?a) 1)` of `board` can never falsify `(> (onboard ?a) 0)`.

Sequential Plans

The sequential encoding allows exactly one action per time step. This is achieved by imposing an *exactly one* constraint on the action variables at each time step. We tested some well-known encodings, and we settled with the binary encoding (Frisch and Giannaros 2010) as it gave us the best performance. This encoding introduces new variables $B_1, \ldots, B_{\lceil \log_2 n \rceil}$, where $n = |A|$, and associates each variable a_i^t with a unique bit string $s_i \in \{0,1\}^{\lceil \log_2 n \rceil}$. The encoding is:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil \log_2 n \rceil} \neg a_i^t \vee \odot(i,j) \tag{4}$$

$$\bigvee_{i=1}^{n} a_i^t \tag{5}$$

where $\odot(i,j)$ is B_j if the j^{th} bit of the bit string of s_i is 1, and $\neg B_j$ otherwise. The binary encoding of the *at most one* constraint (4), introduces $\lceil \log_2 n \rceil$ new variables and $n\lceil \log_2 n \rceil$ binary clauses. Together with the *at least one* constraint (5), we obtain the desired *exactly one* constraint.

Parallel Plans

Encodings for two types of parallel plan semantics have been implemented in RANTANPLAN: $\forall$-step plans, and $\exists$-step plans.

$\forall$-step Plans The notion of parallelism of a $\forall$-step plan is defined as the possibility of ordering the actions of each time step to any total order. Therefore, at each time step t we simply add a mutex between any pair of interfering actions a_i and a_j:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ or } a_j \text{ affects } a_i \tag{6}$$

$\exists$-step Plans In $\exists$-step plans, there must exist at least a total ordering of parallel actions resulting in a valid sequential plan. RANTANPLAN implements a quadratic encoding for this purpose. It takes as ingredient an arbitrary total ordering $<$ on the actions, and the parallel execution of two actions a_i and a_j such that a_i affects a_j is forbidden only if $i < j$:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ and } i < j \tag{7}$$

The linear-size encoding described in (Rintanen, Heljanko, and Niemelä 2006), is also supported.

Since $\exists$-step plans are less restrictive than $\forall$-step plans, as they do not require that all orderings of parallel actions result in valid sequential plan, they normally allow more parallelism.

Plan Serialization

To obtain a sequential plan from the solution, for each time step with more than one action, a subgraph of the disabling graph is extracted, containing only the actions at that time step. A valid order between actions can then be computed.

Since in all implemented parallel encodings acyclicity is guaranteed between the executed actions, a reversed topological order of the subgraph is always as a valid order.

Extension: QF_UFLIA Encoding

As the previously introduced QF_LIA encodings grows considerably with the time horizon, to the point of getting unmanageable instances, we have started to develop a more compact encoding, using the theory of uninterpreted functions to express predicates, functions and actions. This encoding is reminiscent of the lifted causal encodings in (Kautz, McAllester, and Selman 1996).

In the SMT-LIB standard (Barrett, Stump, and Tinelli 2010), QF_UFLIA stands for the logic of Quantifier-Free

Boolean formulas, with Linear Integer Arithmetic constraints and Uninterpreted Functions. Uninterpreted functions have no other property than its name and arity, and are only subject to the following axiom: $x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \rightarrow f(x_1, \ldots, x_n) = f(x'_1, \ldots, x'_n)$.

The encoding goes as follows. Every defined object in the problem is mapped to an integer. For each function, predicate and action, an uninterpreted function is declared, with each parameter being declared as an integer. Also, a new integer parameter is added to each of them, representing a time step. Uninterpreted functions corresponding to predicates and actions return a Boolean value, whilst the ones for functions return an integer value. Moreover, for each action, parameter and time step, a new integer variable is defined, representing the value of that parameter in the action if executed at the corresponding time step.

For example, the Boolean function $\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t)$ determines whether action a with parameters $x^t_{a,1}, \ldots, x^t_{a,n}$ is executed at time step t. The parameter t is a constant, which is shared between all uninterpreted functions for the actions, predicates and functions in the same time step. Contrarily, $x^t_{a,1}, \ldots, x^t_{a,n}$ are variables with finite domains, and constraints are added to restrict their possible values. Regarding predicates and functions, no new variables are defined, since their arguments will be either constants or variables occurring in some action.

We remark that, in this new setting, a state is defined by the value of the uninterpreted functions corresponding to predicates and functions, for a given value of their arguments. Equations (1) and (2) of the QF_LIA encoding are generalized here as:

$$\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t) \rightarrow p^t \quad \forall a = \langle p, e \rangle \in A \qquad (8)$$

$$\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t) \wedge f^t \rightarrow d^{t+1} \quad \forall a = \langle p, e \rangle \in A, \forall \langle f, d \rangle \in e \qquad (9)$$

Note that this results in a much more compact encoding than if we restrict to QF_LIA, since here we are using variables as arguments of functions, and it is the SMT solver who is in charge of guessing the concrete values of the parameters of the executed actions. The considered set of actions A is now parametrized, and hence similar to that of PDDL, with actions like $fly(x, y, z)$, instead of grounded actions like $fly_{p1,c1,c1}$, $fly_{p1,c1,c2}$, etc. Equation (3) is generalized as:

$$\varphi_h(c_{h,1}, \ldots, c_{h,n}, t) \neq \varphi_g(c_{h,1}, \ldots, c_{h,n}, t+1) \rightarrow \bigvee_{a \in touch(g)} \Big(\varphi_a(x^t_{a,1}, \ldots, x^t_{a,m}, t) \bigwedge_{\substack{i \in 1..n, j \in 1..m \\ name(h,i) = name(a,j)}} (x^t_{a,j} = c_{h,i}) \Big) \quad \forall h \in H, \forall c_{h,1}, \ldots, c_{h,n} \in S_1 \times \cdots \times S_n \qquad (10)$$

where H is the set of predicates and functions, $touch(h)$ is the set of actions that may modify h, S_i is the domain of the i-th argument of φ_h, and $name(h, k)$ is the name in the PDDL model of the k-th argument of the functor h. To help the reader understand the formula, we provide an example. Suppose we have the following simple PDDL problem:

- objects: `A,B - truck, L1,L2,L3 - loc`
- predicate: `at(?t - truck, ?l - loc)`
- actions:
  `travel(?t - truck, ?from - loc, ?to - loc)`
  `refuel(?x - truck, ?where - loc)`
- function: `fuel(?t - truck) - number`

where `travel` has `(decrease (fuel ?t) 10)` among its effects, and `refuel` has `(increase (fuel ?x) 20)` as its only effect. Constraint (10) for the `fuel` function would be encoded into SMT at time step 0 as follows:

```
(=> (distinct (fuel A 0) (fuel A 1))
  (or (and (travel x1_0 x2_0 x3_0 0) (= x1_0 A))
      (and (refuel x4_0 x5_0 0) (= x4_0 A))))

(=> (distinct (fuel B 0) (fuel B 1))
  (or (and (travel x1_0 x2_0 x3_0 0) (= x1_0 B))
      (and (refuel x4_0 x5_0 0) (= x4_0 B))))
```

That is, we are saying that if the fuel of truck `A` (or `B`) has changed this should be because it has been the protagonist of some action implying a modification in its fuel, namely traveling or refueling.

Again, this is much more compact than its QF_LIA counterpart. With respect to the parallelism, for now this encoding only supports the sequential plan semantics, as encoding parallelism using this encoding is not straightforward. This approach is currently under development, as we obtained encouraging preliminary experimental results (Bofill, Espasa, and Villaret 2014).

Experimental Evaluation

In this section we report on experiments with RANTANPLAN using Yices (Dutertre and De Moura 2006) v1.0.38 as backend solver. All experiments have been run on 8GB Intel® Xeon® E3-1220v2 machines at 3.10 GHz.

The goal of the experiments is to evaluate if RANTANPLAN is competitive with state of the art exact numeric planners, as well as showing the benefits of having a good notion of interference.

For the sake of simplicity, only QF_LIA $\exists$-step plans are considered, using a quadratic encoding for expressing incompatibility of actions. We experimentally observed that the solver behavior was more stable when using a quadratic encoding than when using a linear encoding, probably because a linear encoding is more perturbable with the chosen ordering of actions.

We consider four distinct domains: the numeric versions of *ZenoTravel* and *Depots*, the real-life challenging *Petrobras* domain, and the crafted *Planes* domain, shown in Figure 2. All instances have been translated to make use of

```
(define (domain planes)
(:requirements :typing :fluents)
(:types  city locatable - object
         aircraft person - locatable)
(:functions
 (at       ?x - locatable) - city
 (in       ?p - person) - aircraft
 (fuel     ?a - aircraft) - number
 (seats    ?a - aircraft) - number
 (capacity ?a - aircraft) - number
 (onboard  ?a - aircraft) - number
 (distance ?c1 - city ?c2 - city) - number)

(:action board
 :parameters (?p - person
              ?a - aircraft
              ?c - city)
 :precondition (and (= (at ?p) ?c)
                 (= (at ?a) ?c)
                 (> (seats ?a) (onboard ?a)))
 :effect (and (assign (at ?p) undefined)
          (assign (in ?p) ?a)
          (increase (onboard ?a) 1)))

(:action debark
 :parameters (?p - person
              ?a - aircraft
              ?c - city)
 :precondition (and (= (in ?p) ?a)
                 (= (at ?a) ?c))
 :effect (and (assign (in ?p) undefined)
          (assign (at ?p) ?c)
          (decrease (onboard ?a) 1)))

(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :precondition (and (= (at ?a) ?c1)
                 (> (onboard ?a) 0)
                 (>= (fuel ?a)
                     (distance ?c1 ?c2)))
 :effect (and (assign (at ?a) ?c2)
          (decrease (fuel ?a)
                    (distance ?c1 ?c2)))
 )

(:action refuel
 :parameters (?a - aircraft)
 :precondition (and
                 (< (* (fuel ?a) 2) (capacity ?a))
                 (= (onboard ?a) 0))
 :effect (and (assign (fuel ?a) (capacity ?a)))))
```

Figure 2: PDDL model of the Planes domain.

object fluents, in order to obtain a compact representation in the translation to SMT. The *Planes* domain was crafted due to the limited interest of the other domains with respect to numeric interactions between actions. This new domain was derived from *ZenoTravel*, by adding some plausible numeric constraints that will help us demonstrate the goodness of the semantic approach when determining potential interference between actions.

We compare the performance of RANTANPLAN with the exact numeric planner NumReach/SAT (Hoffmann et al. 2007) using MiniSAT 2.2.0, and NumReach/SMT using Yices v1.0.38. For NumReach/SMT, we had to adapt its output so it could be used with modern SMT solvers. Moreover, since NumReach supports neither object fluents nor conditional effects, the models have been properly adapted.

Table 1 shows the results for the domains considered using the RANTANPLAN system. The **Syntactic** column shows

Depots Domain

n	Syntactic	Semantic	Time	Edges
1	4.1 (6)	2.8 (6)	31.4%	41.7%
2	32.0 (9)	18.3 (8)	42.8%	44.2%
3	166.9 (13)	108.9 (13)	34.8%	44.9%
4	438.3 (14)	323.0 (14)	26.3%	45.1%
5	TO (8)	TO (17)	-	45.1%
6	TO (-)	MO (1)	-	-
7	188.1 (10)	131.0 (10)	30.4%	44.0%
8	MO (3)	MO (10)	-	44.5%

Zenotravel Domain

n	Syntactic	Semantic	Time	Edges
1	0.0 (0)	0.0 (0)	35.3%	76.3%
2	0.1 (3)	0.0 (3)	23%	74.3%
3	0.2 (3)	0.1 (3)	34.6%	66%
4	0.3 (4)	0.1 (4)	43.5%	66.2%
5	0.5 (4)	0.3 (4)	38.9%	71.5%
6	0.8 (6)	0.5 (6)	43.5%	72.1%
7	0.8 (5)	0.4 (5)	47%	72.6%
8	2.8 (5)	1.7 (5)	38.5%	68.3%
9	26.5 (8)	31.0 (8)	-16.9%	69.6%
10	41.6 (8)	61.9 (8)	-48.7%	70.7%
11	7.1 (6)	4.5 (6)	37.2%	69%
12	105.8 (7)	95.1 (7)	10.1%	70.4%
13	1288.3 (9)	1291.5 (9)	-0.2%	72.6%
14	TO (7)	TO (7)	-	55.0%

Petrobras Domain

n	Syntactic	Semantic	Time	Edges
1	14.7 (3)	8.8 (3)	40.7%	50.4%
2	19.3 (4)	11.2 (4)	42.2%	51.8%
3	24.6 (5)	14.0 (5)	43.2%	53.2%
4	47.0 (8)	28.2 (8)	40.1%	54.5%
5	74.9 (9)	59.5 (9)	20.5%	55.8%
6	133.9 (10)	108.7 (10)	18.8%	57.1%
7	700.1 (13)	475.1 (13)	32.1%	58.3%
8	833.4 (13)	800.0 (13)	4.0%	59.5%

Planes Domain

n	Syntactic	Semantic	Time	Edges
1	1.0 (13)	0.3 (10)	71.5%	84.5%
2	6.0 (16)	1.1 (12)	81.2%	84.5%
3	49.9 (18)	8.3 (13)	83.4%	86.5%
4	431.1 (21)	40.0 (15)	90.7%	86.5%
5	117.2 (20)	27.0 (15)	77.0%	86.1%
6	1294.6 (23)	193.3 (18)	85.1%	86.1%
7	621.9 (21)	70.9 (16)	88.6%	85.8%
8	834.2 (22)	105.7 (17)	87.3%	85.8%
9	TO (23)	2889.1 (20)	-	88.0%

Table 1: Time in seconds followed by the number of parallel steps of the plan found between parentheses, for each instance. TO stands for time out and MO for memory out. Cutoff set to 3600 seconds. The Time and Edges columns show the reduction in time and edges of the disabling graph, respectively, when using the semantic approach. Instances where all approaches timed out are omitted.

the results using the generalization of the interference notion of (Rintanen 2009), described at the beginning of subsection "Interference Between Actions", additionally forbidding any two actions to occur in parallel if they modify the same numeric variable. The **Semantic** column shows the results with the lifted semantic notion of interference. In these two columns the number of parallel steps of the valid plan is found between parentheses. In case of a time out (TO) the number between parentheses is the last plan length considered.

The **Time** column shows how much faster each instance is solved with the semantic notion of interference, and the **Edges** column shows which percentage of edges of the disabling graph can be avoided thanks to this new interference notion. Note that even in instances that need the same amount of time steps, the reduction of edges in the disabling graph affects positively on the solving time. This is probably because we are reducing the number of clauses that do not contribute at all to the problem.

	Accumulated steps		Averaged reductions	
Family	Syntactic	Semantic	Time	Edges
Depots	52	51	33.1%	44.2%
ZenoTravel	68	68	22.0%	70.8%
Petrobras	65	65	30.2%	59.1%
Planes	154	116	84.3%	86.8%

Table 2: Summarized results for the domains considered using the RANTANPLAN system with the syntactic and the semantic notions of interference. For each domain we report the total number of steps of the commonly solved instances, and their averaged reductions in solving time and number of edges of the disabling graph.

Table 2 gives the number of accumulated parallel time steps used to reach a valid plan on the commonly solved instances by the two methods implemented in RANTANPLAN. The other columns show the averaged solving time reduction and disabling graph edge reduction. Note that even in domains that maintain the same number of time steps, the reduced disabling graphs make solving times notably smaller.

Table 3 shows the results for the domains considered, comparing NumReach with the semantic version of RANTANPLAN. NumReach does a good job with the *Depots* and *ZenoTravel* domains, but its performance decreases in more complex numeric domains like *Petrobras* and *Planes*, where the range of possible values for numeric fluents tends to grow.

It can be seen that on the *Planes* domain, containing only a few non-trivial numeric constraints, classical approaches (Syntactic and NumReach) tend to be overly restrictive with respect to incompatibility between actions. In most instances it can be observed an important gap between the number of time steps needed to find a valid plan by NumReach and our semantic approach. This is also generally reflected in terms of solving time.

Table 4 lists the total number of instances of each family, the number of instances solved by NumReach/SAT and NumReach/SMT and the number of instances solved by the presented semantic approach. The last two columns give the

Depots Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	0.0 (6)	1.5 (6)	2.8 (6)
2	0.5 (9)	8.4 (9)	18.3 (8)
3	5.7 (13)	43.1 (13)	108.9 (13)
4	10.1 (15)	134.7 (15)	323.0 (14)
7	2.5 (11)	35.1 (11)	131.0 (10)
8	TO (-)	362.7 (15)	MO (10)
10	4.8 (11)	101.2 (11)	MO (-)
13	2.9 (10)	96.3 (10)	TO (-)
14	25.1 (16)	1650.0 (13)	TO (-)
16	2.2 (9)	118.8 (9)	TO (-)
17	6.8 (8)	313.2 (8)	TO (-)
19	18.1 (11)	849.4 (11)	TO (-)

Zenotravel Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	0.0 (2)	0.2 (2)	0.0 (0)
2	0.0 (7)	1.5 (7)	0.0 (3)
3	0.1 (6)	3.7 (6)	0.1 (3)
4	0.0 (6)	2.4 (6)	0.1 (4)
5	0.1 (7)	7.0 (7)	0.3 (4)
6	0.0 (7)	4.2 (7)	0.5 (6)
7	0.1 (8)	9.1 (8)	0.4 (5)
8	0.4 (7)	8.1 (7)	1.7 (5)
9	0.3 (9)	18.1 (9)	31 (8)
10	0.7 (9)	24.2 (9)	61.9 (8)
11	3.5 (8)	18.4 (8)	4.5 (6)
12	3.8 (10)	99.6 (10)	95.1 (7)
13	22.2 (11)	555.6 (11)	1291.5 (9)
14	TO (-)	537.4 (9)	TO (7)

Petrobras Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	0.4 (6)	39.8 (6)	8.8 (3)
2	9.8 (9)	56.4 (6)	11.2 (4)
3	17.8 (10)	93.9 (7)	14.0 (5)
4	118.3 (11)	256.5 (9)	28.2 (8)
5	317.9 (14)	312.3 (9)	59.5 (9)
6	325.4 (14)	277.2 (9)	108.7 (10)
7	TO (-)	818.1 (11)	475.1 (13)
8	TO (-)	2753.6 (12)	800.0 (13)

Planes Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	TO (-)	36.4 (15)	0.3 (10)
2	3.3 (18)	37.9 (18)	1.1 (12)
3	TO (-)	229.9 (20)	8.3 (13)
4	4.4 (22)	632.0 (23)	40.0 (15)
5	TO (-)	768.4 (22)	27.0 (15)
6	TO (-)	1183.7 (25)	193.3 (18)
7	TO (-)	1241.2 (23)	70.9 (16)
8	5.0 (24)	1278.2 (24)	105.7 (17)
9	TO (-)	TO (-)	2889.1 (20)
12	15.5 (21)	TO (-)	TO (19)

Table 3: Time in seconds followed by the number of parallel steps of the plan found between parentheses, for each instance. TO stands for time out and MO for memory out. Cutoff set to 3600 seconds. Instances where all systems timed out are omitted.

	#	Solved instances			Accum. steps	
		N/SAT	N/SMT	Sem.	N/SMT	Sem.
Dep.	22	11	12	5	54	51
Zen.	20	13	14	13	97	68
Petr.	15	6	8	8	69	65
Plan.	12	4	8	9	170	116

Table 4: Summarized results for the domains considered using NumReach/SAT, NumReach/SMT and RANTANPLAN with the semantic notion of interference. For each domain we report the number of solved instances and their accumulated time steps of the commonly solved ones.

number of accumulated parallel time steps used to reach a valid plan on the commonly solved instances.

Note that the amount of parallelism in RANTANPLAN is notable. With respect to the number of steps, RANTANPLAN is strictly more parallel than NumReach/SAT and NumReach/SMT in nearly all instances.

The only domain where the RANTANPLAN planner is not competitive is the *Depots* domain. It is obvious that the reachability approach of NumReach is more adequate for this domain. Moreover NumReach/SAT dominates NumReach/SMT in this domain. This happens because the numeric reasoning present in the domain is nearly null: the only functions present are for controlling load limits of trucks, and thus this domain is perfectly adequate for the approach used by NumReach/SAT. The use of a Linear Integer Arithmetic solver in the RANTANPLAN planner is overkill and a leaner and more efficient approach should be taken for problems of this kind.

Conclusions and Further Work

We have presented RANTANPLAN, a new system for the setting of exact numeric planning. The planner is based on translation into SMT using a planning as satisfiability approach. It takes advantage of background theories in SMT to easily and transparently handle numeric fluents. Moreover it uses an SMT solver at compile time to detect in advance incompatibility between actions. This incompatibility results from lifting the interference notion of (Rintanen 2009) to the setting of planning with resources. We have argued why the presented approach to interference between actions with numeric fluents is better than purely syntactically based ones, and provided empirical evidence of its usefulness.

We have also shown that our system is competitive with the state of the art exact numeric planner NumReach.

As future work, it should be studied how to incorporate the concepts of relaxed $\exists$-plans (Wehrle and Rintanen 2007; Balyo 2013) to our notions of incompatibility and further develop the more compact QF_UFLIA encoding.

Acknowledgments

All authors supported by the Spanish Ministry of Economy and Competitiveness through the project HeLo (ref. TIN2012-33042). Joan Espasa also supported by UdG grant (BR 2013).

References

Balyo, T. 2013. Relaxing the Relaxed Exist-Step Parallel Planning Semantics. In *25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, 865–871. IEEE.

Barrett, C.; Sebastiani, R.; Seshia, S.; and Tinelli, C. 2009. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, volume 185. IOS Press. chapter 26, 825–885.

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

Barták, R., and Toropila, D. 2010. Solving sequential planning problems via constraint satisfaction. *Fundamenta Informaticae* 99(2):125–145.

Belouaer, L., and Maris, F. 2012. SMT Spatio-Temporal Planning. In *ICAPS 2012 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2012)*, 6–15.

Bofill, M.; Espasa, J.; and Villaret, M. 2014. Efficient SMT Encodings for the Petrobras Domain. In *Proceedings of the 13th International Workshop on Constraint Modelling and Reformulation (ModRef 2014)*, 68–84.

Bofill, M.; Espasa, J.; and Villaret, M. 2015. On Interference between Actions in Planning with Resources. (submitted).

Dovier, A.; Formisano, A.; and Pontelli, E. 2010. Multivalued action languages with constraints in CLP (FD). *Theory and Practice of Logic Programming* 10(02):167–235.

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.

Frisch, A. M., and Giannaros, P. A. 2010. SAT Encodings of the At-Most-k Constraint. Some Old, Some New, Some Fast, Some Slow. In *10th International Workshop on Constraint Modelling and Reformulation (ModRef 2010)*.

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)*. AAAI.

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.

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

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *10th European Conference on Artificial Intelligence (ECAI 92)*, 359–363. John Wiley & Sons, Inc.

Kautz, H., and Walser, J. P. 1999. State-space planning by integer optimization. In *AAAI/IAAI*, 526–533.

Kautz, H. A.; McAllester, D. A.; and Selman, B. 1996. Encoding Plans in Propositional Logic. In *Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR 96)*, 374–384. Morgan Kaufmann.

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*. IOS Press. 483–504.

Rintanen, J. 2012. Planning as Satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Wehrle, M., and Rintanen, J. 2007. Planning as Satisfiability with Relaxed Exist-Step Plans. In *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence*, volume 4830 of *LNCS*, 244–253. Springer.

Wolfman, S. A., and Weld, D. S. 1999. The LPSAT Engine & Its Application to Resource Planning. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*, 310–317. Morgan Kaufmann.