Efficient SMT Encodings for the Petrobras Domain

Miquel Bofill, Joan Espasa, and Mateu Villaret

Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, Spain
{miquel.bofill,joan.espasa,mateu.villaret}@udg.edu

Abstract. Reformulation into SAT is one of the main approaches to deterministic planning. The main idea is to successively check the satisfiability of each of the propositional formulas in a sequence ϕ_0, ϕ_1, ϕ_2, ... where ϕ_i encodes the feasibility of a plan of length i. For real-life applications, resources such as time, distances, or money must often be considered. Therefore, in the context of plannning with resources, a natural choice is to consider satisfiability modulo theories (SMT) instead of plain SAT. In this paper we consider the application of off-the-shelf SMT solvers to the Petrobras domain, an abstraction of a real-life problem of resource-efficient transportation of goods from ports to petroleum platforms. We consider different encodings into SMT and compare with state of the art alternative approaches.

1 Introduction and Preliminaries

The problem of planning, in its most basic form, consists in finding a sequence of actions that will 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 better addressed by looking at it as a satisfiability (model finding) problem [14]. 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 [15] it was shown that modern off-the-shelf SAT solvers could be effectively used to solve planning problems. In the last years, the power of SAT technology has been leveraged to planning [17], making reduction into SAT state of the art for deterministic planning.

In the deterministic planning problem actions are formalized as pairs $\langle p, e\rangle$, where p and e are sets of literals denoting the precondition and the effects of the action, respectively. A finite set of Boolean variables determines the state at each moment, and an action $\langle p, e\rangle$ is executable in state s if $s \models p$. The successor state s' (which can be seen either as a set of literals or as a valuation function) is defined by $s' \models e$ and $s'(a) = s(a)$ for all variables a not occurring in e. A solution to the planning problem consists in a sequence of actions that allow to reach a goal state from a given initial state.

To solve the planning problem by reduction into SAT, a sequence of formulas ϕ_0, ϕ_1, ϕ_2, ... is considered, where ϕ_i encodes the feasibility of a plan that allows to reach the goal state from the initial state in i steps, i.e., by executing i actions sequentially. 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. Then, a plan of length n can be extracted from the assignment satisfying ϕ_n. The basic idea of the encoding [16,17] for ϕ_i is to introduce, for each state variable a and time point $t \in \{0, \ldots, i-1\}$, the formula

$$a_{t+1} \rightarrow (a_t \vee o_1^a \vee \cdots \vee o_n^a)$$

where $o_1^a, \ldots, o_n^a$ are all the actions (operators) that have a as effect, for explaining the possible reasons of the truth of a_{t+1}, and

$$\neg a_{t+1} \rightarrow (\neg a_t \vee o_1^{\neg a} \vee \cdots \vee o_m^{\neg a})$$

where $o_1^{\neg a} \vee \cdots \vee o_m^{\neg a}$ are all the actions that have $\neg a$ as effect, for explaining the possible reasons of the falsity of a_{t+1}. Many-valued variables must be represented in terms of several Boolean ones. In this setting, binary constraints of the form $\neg a_{1,t} \vee \neg a_{2,t}$, stating that a many-valued variable a can only have one of the two values 1 and 2 at time t, are redundant but often crucial for efficiency.

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. However, 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 [16,18] for further details.

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. However, the advances in satisfiability modulo theories (SMT) [1] in the last years make worth considering this alternative.

One of the first approaches to planning with resources, the one of LPSAT [20], 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 there. A comparison between SAT and SMT based encodings for planning in numeric domains can be found in [12]: 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. The authors argue that the expressivity of the SMT language comes at the price of requiring much more complex solvers than SAT solvers, and their 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 [4], a set of encoding rules is defined for spatio-temporal planning, with SMT as target formalism. In [11] a modular framework, inspired in the work of SMT, is developed for planning with resources.

In this paper we consider the application of off-the-shelf SMT solvers to the Petrobras domain, an abstraction of a real-life problem of resource-efficient transportation of goods from ports to petroleum platforms, posed as a challenge at the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012).

The rest of the paper is structured as follows. In Section 2 the Petrobras problem is presented and modelled. In Section 3 different SMT encodings for this problem are considered. Section 4 is devoted to experimental evaluation. Conclusions are given in Section 5

2 The Petrobras Problem

The Petrobras domain was posed as a challenge problem at the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012). This domain[1] is an interesting real-life problem, that lies in the border between scheduling and planning.

Generically speaking, the problem is described as the need to transport various cargos of goods and tools from two ports to various platforms located in the ocean at various distances. The strips are divided in two parts: Rio de Janeiro and Santos. The basic elements and agents of the problem are: ports, platforms, waiting areas, cargo items and vessels. The actions that can be performed are:

Sail Navigates a ship from one location to another.
Dock Docks a vessel in a port or platform.
Undock Undocks a vessel in a port or platform.
Load Loads a cargo item into the ship.
Unload Unloads a cargo item from the ship to a platform or port.
Refuel Refuels a ship at a refueling location (a port or any specified platform).

Although the proposal gives various optimization criteria, we only consider the satisfiability of the problem, minimizing the plan length.

[1] `http://icaps12.icaps-conference.org/ickeps/petrobrasdomain.html`

2.1 Modelling

We model the problem with the Planning Domain Definition Language (PDDL). PDDL [10] is the language used in the International Planning Competition, and the most commonly supported input definition language of modern planning systems.

Natural Model We focus on encoding the first scenario proposed by Petrobras, where time constraints like speed, refueling rates, and docking costs are not considered. We use the PDDL language facilities to compactly encode some of the problem features, like refueling or sailing using conditional preconditions and effects.

```
(:types strip port waiting_area platform - location
        ship cargo - object )

(:predicates
    (at_ ?sh - object ?where - location)
    (can_refuel ?l - location)
    (docked ?sh - ship ?where - location)
    (loaded ?c - cargo ?sh - ship)
    (is_waiting_area ?l - location) )

(:functions
    (current_fuel ?sh - ship) - number
    (current_load ?sh - ship) - number
    (fuel_capacity ?sh - ship) - number
    (load_capacity ?sh - ship) - number

    (current_docking_capacity ?p - location) - number
    (total_docking_capacity ?p - location) - number

    (distance ?from - location ?to - location) - number

    (weight ?c - cargo) - number
    (total_fuel_used) - number )
```

Fig. 1. Preamble for the Petrobras domain, in PDDL.

Figure 1 depicts the types, predicates and functions used. The following actions are defined using those elements: `sail`, `load`, `unload`, `refuel_at_port`, and `refuel_at_platform`. One can intuitively model the actions, taking into account the following design decisions:

– The `sail` action (Figure 2) can only be performed if the ship is not docked, and it has enough fuel for that trip. Note that the consumed fuel depends on the current ship load.

– To perform the `load` and `unload` actions, the ship must be docked where the load is, and it must have enough free space. The current load of the ship must be modified accordingly.
– For the `dock` and `undock` operations, the position and docking capacities must be checked.

As time constraints are simplified, the refueling amount is set to either 100 or 200 liters per action, and conditional effects are used to ensure that the total fuel capacity of the vessels is not exceeded.

Model with Unconditional Constraints As the NumReach solver of [12] does not support conditional effects, like the ones introduced with the `when` keyword in Figure 2, we model some parts of the problem in a different way. This will allow to make a fair comparison between our software and NumReach. In order to make the minimum changes from the natural model, we do not change any predicate or function. We only make the following changes: We split the `sail` action in two, namely `sail_empty` and `sail_full`, with the (unconditional) effects corresponding to the case where `current_load` is zero or not, respectively (see Figure 2). The `refueling_at_port` and `refueling_at_platform` actions also make use of conditional effects, so they also need to be split in two. Note that although this apparently seems a minor change, it may cause many new variables to appear in the final encoding, as more actions are present.

```
(:action sail

    :parameters (?sh - ship ?from - location ?to - location)

    :precondition (and (at_ ?sh ?from)
       (not docked ?sh ?from)
       (imply (= (current_load ?sh) 0)
        (>= (current_fuel ?sh) (/ (distance ?from ?to) 5)))
       (imply (not (= (current_load ?sh) 0))
        (>= (current_fuel ?sh) (/ (distance ?from ?to) 3))))

    :effect (and (at_ ?sh ?to)
       (not (at_ ?sh ?from))
       (when (= (current_load ?sh) 0) (and
        (decrease (current_fuel ?sh) (/ (distance ?from ?to) 5))
        (increase (total_fuel_used) (/ (distance ?from ?to) 5))))
       (when (not (= (current_load ?sh) 0)) (and
        (decrease (current_fuel ?sh) (/ (distance ?from ?to) 3))
        (increase (total_fuel_used) (/ (distance ?from ?to) 3)))))
)
```

Fig. 2. Example action in the natural model, containing conditional effects.

3 SMT Encodings

We reformulate the *natural* and *unconditional*[2] PDDL models of the Petrobras problem to SMT. We apply a preprocessing step where we identify constant function values and simplify arithmetic constraints that operate on them. For example, function values like `(/ (distance ?from ?to)` in Figure 2 can be replaced by a numeric constant since `?from` and `?to` are given locations, and distances between locations are constant.

3.1 QF_LIA Encoding

In the SMT-LIB standard [2], 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. The reformulation of our PDDL models goes 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 example, the action `dock(?sh - ship, ?loc - location)`, with three locations `P1`, `P2` and `P3`, and two ships `ship1` and `ship2`, will result into six ground instances `dock(ship1,P1)`, ..., `dock(ship2,P3)`, that will be mapped to six Boolean variables $dock^t_{ship1,P1}$, ..., $dock^t_{ship2,P3}$ for each time step t. 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. Again, a superscript t is used to differentiate the variables at each time step.

Below we describe the constraints that we impose on these variables, following the notation and ideas in [16]: O is the set of all actions o of the form $\langle p, e\rangle$, where p is the action precondition,[3] and e is a set of conditional action effects of the form $\langle f, d\rangle$ (where f is a condition, and d is a set of literals that must be set to true when f holds). We define $EPC_l(o) = \bigvee\{f \mid \langle f, d\rangle \in e, l \in d\}$, that is, the *effect precondition* for literal l in action o.

Since apart from Boolean variables we also deal with integer variables, we assume that we can have literals like $x \geq k$ in preconditions and like $x = x^{prev} + k$ in effects, where x^{prev} denotes the previous value of x. Moreover, given a formula ϕ, by ϕ^t we denote the same formula where all integer variables x have been replaced by x^t, and x^{prev} by x^{t-1}, and analogously for Boolean variables.

For each ground[4] action $o = \langle p, e\rangle$, we have the following constraints. First, its execution during time step t implies that its precondition is met.

[2] For the sake of brevity, we refer to the model with unconditional constraints as the *unconditional model*.

[3] Here preconditions can be general propositional formulas, not just sets of literals.

[4] 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.

$$o^t \implies p^t \qquad \forall o \in O \tag{1}$$

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

$$o^t \wedge f^t \implies d^{t+1} \qquad \forall o \in O, \forall \langle f, d \rangle \in e \tag{2}$$

Here we view sets d of literals as conjunctions of literals. Note that unconditional effects will have $\top$ as condition f.

Second, we need explanatory axioms to express the reason of a change in the value of Boolean state variables.

$$a^t \wedge \neg a^{t+1} \implies ((o_1^t \wedge EPC_{\neg a}^t(o_1)) \vee \cdots \vee (o_m^t \wedge EPC_{\neg a}^t(o_m))) \tag{3}$$

$$\neg a^t \wedge a^{t+1} \implies ((o_1^t \wedge EPC_{a}^t(o_1)) \vee \cdots \vee (o_m^t \wedge EPC_{a}^t(o_m))) \tag{4}$$

Constraints (3) and (4) are generalized for numeric state variables as follows.

$$x^t \neq x^{t+1} \implies ((o_1^t \wedge EPC_{mod(x,o_1)}^t(o_1)) \vee \cdots \vee (o_m^t \wedge EPC_{mod(x,o_m)}^t(o_m))) \tag{5}$$

where $mod(x, o)$ is an arithmetic literal, either $x = x^{prev} + k$, $x = x^{prev} - k$, or $x = k$, resulting from the translation of the PDDL expression `increase(x,k)`, `decrease(x,k)` or `assign(x,k)`, respectively, occurring in the effects of the PDDL action corresponding to o.

3.2 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 (see [9]) as it gave us the best performance. The encoding introduces new variables $B_1, \ldots, B_{\lceil \log_2 n \rceil}$, where $n = |O|$, and associates each variable o_i^t with a unique bit string $s_i \in \{0, 1\}^{\lceil \log_2 n \rceil}$. The encoding is:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil \log_2 n \rceil} \neg o_i^t \vee \odot(i, j) \tag{6}$$

$$\bigvee_{i=1}^{n} o_i^t \tag{7}$$

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 (6), introduces $\lceil \log_2 n \rceil$ new variables and $n\lceil \log_2 n \rceil$ binary clauses. Together with the `at-least-one` constraint (7), we obtain the desired `exactly-one` constraint.

3.3 Parallel Plans

A parallel plan follows the same idea as a sequential plan, but with the exception that at each time step a set of actions can be executed, instead of only one. Two types of parallel plans have been encoded: $\forall$-step plans, and $\exists$-step plans.

We have found a good compromise between the number of added clauses and the parallelism obtained in the Petrobras domain using the notion of affectation defined in [16]. For the Boolean case, an action $o = \langle p, e \rangle$ is considered to affect $o' = \langle p', e' \rangle$ if, for some Boolean state variable a, either of the following holds:

$$\begin{gathered} a \in d \text{ for some } \langle f, d \rangle \in e \wedge \big(a \text{ occurs in } f' \text{ for some } \langle f', d' \rangle \in e' \\ \vee\, a \text{ occurs negatively in } p'\big) \end{gathered} \tag{8}$$

$$\begin{gathered} \neg a \in d \text{ for some } \langle f, d \rangle \in e \wedge \big(a \text{ occurs in } f' \text{ for some } \langle f', d' \rangle \in e' \\ \vee\, a \text{ occurs positively in } p'\big) \end{gathered} \tag{9}$$

That is, o affects o' if o can impede the execution of o', or change its effects. For integer variables x the constraints become:

$$\begin{gathered} mod(x, o) \in d \text{ for some } \langle f, d \rangle \in e \\ \wedge \big(x \text{ occurs in } f' \text{ for some } \langle f', d' \rangle \in e' \vee x \text{ occurs in } p'\big) \end{gathered} \tag{10}$$

where $mod(x, o)$, as in (5), is an arithmetic literal of the form $x = x^{prev} + k$, $x = x^{prev} - k$ or $x = k$.

These precomputed affectations will be used to forbid parallel executions of incompatible actions. Notice that affectation is not a symmetric relation.

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

$$\neg(o_i^t \wedge o_j^t) \text{ if } o_i \text{ affects } o_j \text{ or } o_j \text{ affects } o_i \tag{11}$$

∃-step Plans In ∃-step plans, a fixed (arbitrary) total ordering on the actions is imposed beforehand, and the parallel execution of two actions o_i and o_j such that o_i affects o_j is forbidden only if $i < j$:

$$\neg(o_i^t \wedge o_j^t) \text{ if } o_i \text{ affects } o_j \text{ and } i < j \tag{12}$$

Since ∃-step plans are less restrictive than ∀-step plans, as they do not require that all orderings of parallel actions result in valid sequential plan, they sometimes allow more parallelism.

3.4 QF_UFLIA Encoding

In the SMT-LIB standard [2], 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. In other words, they are only subject to the following axiom schema of consistency: $x_1 = x_1' \wedge \cdots \wedge x_n = x_n' \implies f(x_1, \ldots, x_n) = f(x_1', \ldots, x_n')$.

As the previously introduced QF_LIA encodings grows considerably with time, to the point of getting unmanageable instances, we developed 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 [15].

Every defined object (ship, port, cargo, ...) 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, 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 $\varphi_o(x_{o,1}^t, \ldots, x_{o,n}^t, t)$ determines whether action o with parameters $x_{o,1}^t, \ldots, x_{o,n}^t$ is executed at time step t. The parameter t is a constant representing the time step. It is shared between all uninterpreted functions for the actions, predicates and functions in the same time step. Contrarily, $x_{o,1}^t, \ldots, x_{o,n}^t$ 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:

$$\varphi_o(x_{o,1}^t, \ldots, x_{o,n}^t, t) \implies p^t \qquad \forall o = \langle p, e\rangle \in O \tag{13}$$

$$\varphi_o(x_{o,1}^t, \ldots, x_{o,n}^t, t) \wedge f^t \implies d^{t+1} \qquad \forall o = \langle p, e\rangle \in O, \forall \langle f, d\rangle \in e \tag{14}$$

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 O is now parameterized, and hence similar to that of PDDL, with actions like $dock(x, y)$, $sail(x, y, z)$, etc. instead of grounded actions like $dock_{ship1,P1}$, $dock_{ship1,P2}$, etc.

Equations (3), (4) and (5) are generalized as:

$$\begin{gathered}\varphi_g(c_{g,1},\ldots,c_{g,n},t) \neq \varphi_g(c_{g,1},\ldots,c_{g,n},t+1) \implies \\ \bigvee_{o \in touch(g)} \Big(\varphi_o(x^t_{o,1},\ldots,x^t_{o,m},t) \bigwedge_{\substack{i \in 1..n, j \in 1..m \\ name(g,i) = name(o,j)}} (x^t_{o,j} = c_{g,i})\Big) \\ \forall g \in G \quad \forall c_{g,1},\ldots,c_{g,n} \in S_1 \times \cdots \times S_n\end{gathered} \tag{15}$$

where G is the set of predicates and functions, $touch(g)$ is the set of actions that may modify g, S_i is the domain of the i-th argument of φ_g, 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.

Example 1. Suppose we have the following simple PDDL problem:

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

where the action `travel` has `(decrease (fuel ?t) 10)` among its effects, and the action `refuel` has `(increase (fuel ?x) 20)` as its only effect. The constraint described in (15) 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.

4 Experimental Evaluation

The experiments were run on a cluster of machines, running the CentOS operating System, equipped with Intel® Xeon® E3-1220v2 Processors at 3.10 GHz with Turbo Boost disabled, and 8GB of main memory.

To test our encodings, we created a very similar set of benchmarks to the ones used in [19]. It consists of 4 groups of generated instances, with an increasing number of cargo items, ranging from 1 to 15. Every cargo is assigned randomly to one of the two ports, and each ship is randomly docked in one of them. The groups differ in the number of ships and in the total fuel capacity of each ship:

- Group A: 3 ships with 600 liters of fuel capacity.
- Group B: 10 ships with 600 liters of fuel capacity.
- Group C: 10 ships with 800 liters of fuel capacity.
- Group D: 10 ships with 1000 liters of fuel capacity.

We compare the performance of our approach to that of NumReach [12], which approximates the reachable domains of the numeric state variables. That is, it generates 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 $D_t(v)$. Then a SAT encoding is generated, by introducing a Boolean variable $a_{v,c,t}$ for every t, v and $c \in D_t(v)$. As we will see, this method is very sensitive to the size of $D_k(v)$.

The input language of the NumReach solver is PDDL, and it has two strategies for solving: NumReach/SAT and NumReach/SMT. NumReach/SMT is similar to NumReach/SAT, except for the encoding of the numeric variables. NumReach uses a different backend solver for each one. For the SAT approach, it uses MiniSAT or ZChaff, but we only used the latest version of MiniSAT (2.2.0) [7] in the experiments, as we couldn't find any modern version of ZChaff. For the SMT backend, it was not possible to use any modern version of a SMT solver, and we had to restrict MathSAT 3. This is because NumReach generates the SMT instances in a file format which is different from SMT-LIB and not known by modern SMT solvers. We also could not find any documentation on the format used, and writing a converter would be costly. For this reasons and due to the poor observed performance, we do not include the results for NumReach/SMT.

During the experiments with NumReach, we found out that MiniSAT dedicated most of its solving time into simplificating the formula. So we decided to execute the same experiments in two ways: instructing MiniSAT not to simplify the input formula, and with the default options. In the tables of results we refer to both solving options as SAT and SAT w/o pre, respectively.

For each instance, we made executions for the three QF_LIA encodings: sequential, $\forall$-step and $\exists$-step, with two SMT solvers via API: Yices-1.0.38 [6] and Z3-4.3.1 [5]. The results depicted are from the Yices executions, as although it wasn't always faster, it solved more instances than Z3. The executions were made through the APIs, because when we tried to use plain files we found that the generated files for the biggest instances were too big for the solvers, spanning some gigabytes.

We do not report results for the QF_UFLIA encoding, as they are comparable to that of the sequential QF_LIA encoding and, moreover, the extension of the QF_UFLIA encoding to parallel plans is a non trivial task, as the encoding of the explanatory axioms relies on the premise that only one action is executed. Consequently we leave it as future work.

Nevertheless, it is worth noting that the QF_UFLIA encoding, which is much more compact, allowed us to run the solvers by feeding them the formula from a file.

Table 1. Execution times for group A in seconds, and number of time steps checked.

Instance	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
A1	2.73	(5)	45.27	(5)	22.95	(5)	105.77	(6)	**1.20**	(6)
A2	45.67	(13)	76.37	(8)	40.51	(8)	780.93	(9)	**13.65**	(9)
A3	TO	(17)	105.44	(10)	55.29	(9)	1231.98	(10)	**26.28**	(10)
A4	TO	(17)	1727.77	(13)	2674.75	(12)	2067.24	(11)	**137.47**	(11)
A5	TO	(19)	TO	(13)	TO	(13)	2992.79	(12)	**243.45**	(12)
A6	TO	(19)	TO	(14)	TO	(13)	TO	(13)	**933.22**	(14)
A7	TO	(19)	TO	(14)	TO	(13)	TO	(13)	**431.90**	(13)
A8	TO	(20)	TO	(14)	TO	(14)	TO	(13)	TO	(14)
A9	TO	(18)	TO	(14)	TO	(13)	TO	(12)	TO	(14)
A10	TO	(19)	TO	(15)	TO	(14)	TO	(12)	TO	(12)
A11	TO	(20)	TO	(14)	TO	(14)	TO	(12)	TO	(12)
A12	TO	(20)	TO	(14)	TO	(15)	TO	(12)	TO	(12)
A13	TO	(20)	TO	(15)	TO	(15)	TO	(12)	TO	(12)
A14	TO	(19)	TO	(14)	TO	(14)	TO	(12)	TO	(12)
A15	TO	(21)	TO	(15)	TO	(15)	TO	(12)	TO	(12)

Tables 1 to 4 show the execution time in seconds and number of time steps checked for each group of instances. TO denotes that the solver could not find a plan in the given time of one hour, and the fastest solver for each instance is highlighted. Between parenthesis there is the last time step checked by the solver (which corresponds to the length of the shortest plan found for the solved instances). Note that it is not clear for us what notion of affectation or parallelism is NumReach using, so the plan lengths between correct solutions given by NumReach and our encoding for the same instance may differ. Table 5 summarizes how many instances each solving approach could finish in the given time, and among those in how many it was the fastest.

If we look at the NumReach/SAT executions, all the instances have a better solving time without simplificating the input formula. But, although we observe an speedup of more than one order of magnitude on most of the solved instances, only a few more instances can be solved without preprocessing, due to the combinatorial explosion. This indicates that the problem is inherently hard.

After analyzing the computed affectations between actions, we could see that the problem is highly parallel in the number of ships. Ships can operate inde-

Table 2. Execution times for group B in seconds, and number of time steps checked.

Instance	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
B1	11.47	(5)	154.01	(5)	79.20	(5)	366.29	(6)	**9.02**	(6)
B2	154.53	(10)	158.88	(5)	82.80	(5)	544.15	(6)	**12.76**	(6)
B3	TO	(12)	162.81	(5)	86.02	(5)	1344.00	(7)	**35.97**	(7)
B4	TO	(13)	168.30	(5)	**89.74**	(5)	2761.95	(8)	151.22	(8)
B5	TO	(13)	173.05	(5)	**93.77**	(5)	2864.55	(8)	167.65	(8)
B6	TO	(14)	178.21	(5)	**96.92**	(5)	2952.88	(8)	173.83	(8)
B7	TO	(14)	183.16	(5)	**101.81**	(5)	TO	(10)	TO	(10)
B8	TO	(14)	300.96	(7)	**189.73**	(7)	TO	(9)	TO	(9)
B9	TO	(14)	TO	(8)	TO	(7)	TO	(9)	TO	(9)
B10	TO	(15)	748.626	(8)	**358.57**	(7)	TO	(9)	TO	(9)
B11	TO	(14)	TO	(8)	TO	(8)	TO	(9)	TO	(9)
B12	TO	(14)	TO	(8)	TO	(8)	TO	(9)	TO	(9)
B13	TO	(14)	TO	(8)	TO	(8)	TO	(9)	TO	(9)
B14	TO	(16)	TO	(9)	TO	(8)	TO	(9)	TO	(9)
B15	TO	(14)	TO	(9)	TO	(8)	TO	(9)	TO	(9)

Table 3. Execution times for group C in seconds, and number of time steps checked.

Instance	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
C1	11.58	(5)	154.46	(5)	79.05	(5)	510.45	(6)	**14.78**	(6)
C2	149.59	(10)	159.05	(5)	81.59	(5)	744.29	(6)	**21.80**	(6)
C3	TO	(13)	163.17	(5)	86.21	(5)	1826.95	(7)	**71.88**	(7)
C4	TO	(13)	168.05	(5)	**89.76**	(5)	TO	(9)	TO	(9)
C5	TO	(13)	173.13	(5)	**93.44**	(5)	TO	(8)	TO	(9)
C6	TO	(13)	178.25	(5)	**97.28**	(5)	TO	(8)	TO	(8)
C7	TO	(14)	183.30	(5)	**101.06**	(5)	TO	(8)	TO	(8)
C8	TO	(14)	298.40	(7)	**168.80**	(7)	TO	(8)	TO	(8)
C9	TO	(14)	TO	(8)	TO	(7)	TO	(8)	TO	(8)
C10	TO	(14)	758.50	(8)	**351.54**	(7)	TO	(8)	TO	(8)
C11	TO	(14)	TO	(8)	TO	(8)	TO	(8)	TO	(8)
C12	TO	(14)	TO	(8)	TO	(8)	TO	(8)	TO	(8)
C13	TO	(14)	TO	(8)	TO	(8)	TO	(8)	TO	(8)
C14	TO	(17)	TO	(9)	TO	(8)	TO	(8)	TO	(8)
C15	TO	(14)	TO	(9)	TO	(8)	TO	(8)	TO	(8)

pendently, with the only limitation of the docking space. This can be seen for example in instance D7: in 4 time steps, 7 cargo items are transported from the port of origin to its destination. The difference of 2 time steps between D7 and D8 is caused only by the docking space capacities.

Note also that ∃-step plans are easier to find than ∀-step plans in this domain. However, contrarily to what could be expected, in most of the cases they are not shorter. This is due to the nature of the domain: as said, ships can operate independently and hence, in many cases, requiring parallel actions to result in a

Table 4. Execution times for group D in seconds, and number of time steps checked.

Instance	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
D1	**11.58**	(5)	154.47	(5)	78.80	(5)	1097.26	(6)	24.71	(6)
D2	139.56	(10)	158.66	(5)	81.80	(5)	1777.23	(6)	**37.44**	(6)
D3	TO	(12)	163.36	(5)	**85.86**	(5)	TO	(7)	144.29	(7)
D4	TO	(13)	168.01	(5)	**89.39**	(5)	TO	(7)	TO	(7)
D5	TO	(13)	173.36	(5)	**93.17**	(5)	TO	(7)	TO	(7)
D6	TO	(13)	177.89	(5)	**97.26**	(5)	TO	(7)	TO	(7)
D7	TO	(14)	182.66	(5)	**100.92**	(5)	TO	(7)	TO	(7)
D8	TO	(14)	302.91	(7)	**240.33**	(7)	TO	(7)	TO	(7)
D9	TO	(14)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D10	TO	(14)	762.78	(8)	**333.21**	(7)	TO	(7)	TO	(7)
D11	TO	(15)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D12	TO	(14)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D13	TO	(14)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D14	TO	(17)	TO	(9)	TO	(8)	TO	(7)	TO	(7)
D15	TO	(15)	TO	(9)	TO	(8)	TO	(7)	TO	(7)

Table 5. Summary of the results.

60 instances	QF_LIA encoding			NumReach	
	Sequential	∀-step	∃-step	SAT	SAT w/o pre
Total solved	8	31	31	16	19
Faster instances	1	0	19	0	14

valid plan if putting them to any total order, is not stronger than requiring this for some fixed order.

On the other hand, under our approach, with the *natural* model the solver found a solution for 7 more instances than with the *unconditional* model. However the natural model could not be compared with NumReach (recall that it does not support conditional effects), so the results shown in Tables 1 to 5 are from the unconditional model.

Intuitively, a higher ship fuel capacity should make the problem easier, as less actions will be necessary as ships will need to refuel less often. Instead, it is interesting to note that for NumReach/SAT the groups C and D become the hardest instances. This is because with the higher numbers, state-space exploration seems to grow too large to be manageable, as we suspected.

Other recent works have provided efficient solutions to the Petrobras challenge proposal. In [19,3] various heuristic (incomplete) solvers are used to solve the Petrobras challenge under different optimization criteria. Since our natural PDDL model is very close to one of the proposed models in [19], we mimicked the generated benchmarks and compared the results. In Table 6 we can see that, with the unconditional model and the QF_LIA encoding, only 3 instances less are solved than with SGPlan [13]. But if we consider the natural model, also with the QF_LIA encoding, 7 more instances are solved, outperforming SGPlan.

Table 6. Number of instances solved by each approximation.

Instance set	Unconditional model			Natural model			SGPlan
	Sequential	∀-step	∃-step	Sequential	∀-step	∃-step	
Group A	2	4	4	3	5	5	6
Group B	2	9	9	2	11	11	6
Group C	2	9	9	2	11	11	10
Group D	2	9	9	2	11	11	12
Total	8	31	31	9	38	38	34

5 Conclusions and Future Work

In this paper we have presented several SMT encodings for the Petrobras challenge. The proposed encodings make use of SMT to tightly integrate arithmetic into the problem, where other approximations rely into making state-space exploration on the numerical variables, or loosely integrate external solvers for evaluating arithmetic constraints. Our approximation seems to be competitive with other exact and complete methods for planning with resources on this problem, and also with some incomplete (heuristic) ones. In particular, we have obtained better results than NumReach [12] and similar results to SGPlan [13]. We have seen that the method of [12], which is based on approximating the reachable domains of numeric variables, is very sensitive to the number of distinct possible values, and it is not well-suited for this real real-life problem.

Although SAT and SMT solvers have generic preprocessing steps to simplify the input formulas, we observed that for MiniSAT those were harmful for this problem. Nevertheless, it would be interesting to consider some more ad hoc preprocessing steps to help reduce the search space, based on reachability analysis, as well as operator splitting and factoring [15,8].

We have also observed a promising performance using a sequential encoding with uninterpreted functions. This encoding is more compact, and it retains most of the problem original structure. It remains to be seen if a parallelized version of this encoding could lead to better results than the encoding without functions. To the best of our knowledge, there are no works using parallelized encodings with uninterpreted functions. It should hence be studied how to generalize the standard parallel encodings to this setting.

Acknowledgements

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).

The authors acknowledge Jörg Hoffmann for giving them access to the NumReach software.

References

1. Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, volume 185, chapter 26, pages 825–885. IOS Press, February 2009.
2. Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), 2010. `http://www.SMT-LIB.org`.
3. Roman Barták and Neng-Fa Zhou. On Modeling Planning Problems: Experience from the Petrobras Challenge. In *Advances in Soft Computing and Its Applications - 12th Mexican International Conference on Artificial Intelligence (MICAI 2013, Part II)*, volume 8266 of *LNCS*, pages 466–477. Springer, 2013.
4. Lamia Belouaer and Frédéric Maris. SMT Spatio-Temporal Planning. In *ICAPS 2012 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2012)*, pages 6–15, 2012.
5. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
6. Bruno Dutertre and Leonardo De Moura. The Yices SMT Solver. Technical report, Computer Science Laboratory, SRI International, 2006. Available at `http://yices.csl.sri.com`.
7. Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
8. Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-Compilation of Planning Problems. In *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 1169–1177. Morgan Kaufmann, 1997.
9. Alan M. Frisch and Paul A. Giannaros. 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)*, 2010. `http://www.it.uu.se/research/group/astra/ModRef10/programme.html`.
10. Alfonso Gerevini and Derek Long. Preferences and soft constraints in PDDL3. In *ICAPS workshop on planning with preferences and soft constraints*, pages 46–53, 2006.
11. Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. Planning Modulo Theories: Extending the Planning Paradigm. In *Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI, 2012.
12. Jörg Hoffmann, Carla P. Gomes, Bart Selman, and Henry A. Kautz. SAT Encodings of State-Space Reachability Problems in Numeric Domains. In *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1918–1923, 2007.
13. Chih-Wei Hsu and Benjamin W. Wah. The SGPlan Planning System in IPC-6, 2008. `http://wah.cse.cuhk.edu.hk/wah/Wah/papers/C168/C168.pdf`.
14. Henry Kautz and Bart Selman. Planning As Satisfiability. In *10th European Conference on Artificial Intelligence (ECAI 92)*, pages 359–363. John Wiley & Sons, Inc., 1992.
15. Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding Plans in Propositional Logic. In *Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR 96)*, pages 374–384. Morgan Kaufmann, 1996.

16. Jussi Rintanen. Planning and SAT. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 483–504. IOS Press, 2009.
17. Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.
18. Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.
19. Daniel Toropila, Filip Dvorak, Otakar Trunda, Martin Hanes, and Roman Barták. 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)*, volume 1, pages 191–198. IEEE, 2012.
20. Steven A. Wolfman and Daniel S. Weld. The LPSAT Engine & Its Application to Resource Planning. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*, pages 310–317. Morgan Kaufmann, 1999.