Acta Polytechnica CTU Proceedings **2**:1–7, 2015
doi:10.14311/APP.2015.1.0001

available online at http://ojs.cvut.cz/ojs/index.php/app

REINFORCED ENCODING FOR PLANNING AS SAT

Tomáš Balyo*, Roman Barták, Otakar Trunda

Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Malostranske namesti 2/25, Praha 1, Czech Republic

* corresponding author: biotomas@gmail.com

Abstract. Solving planning problems via translation to satisfiability (SAT) is one of the most successful approaches to automated planning. We propose a new encoding scheme, called Reinforced Encoding, which encodes a planning problem represented in the SAS+ formalism into SAT. The Reinforced Encoding is a combination of the transition-based SASE encoding with the classical propositional encoding. In our experiments we compare our new encoding to other known SAS+ based encodings. The results indicate, that he Reinforced encoding performs well on the benchmark problems of the 2011 International Planning Competition and can outperform all the other known encodings for several domains.

Keywords: planning, satisfiability, encoding.

1. Introduction

Planning is the problem of finding a sequence of actions – a plan, that transforms the world from an initial state to a state that satisfies some goal conditions. The world is fully-observable, deterministic and static (only the agent we make the plan for changes the world). The number of possible states of the world as well as the number of possible actions is finite, though possibly very large. We will assume that the actions are instantaneous (take a constant time) and therefore we only need to deal with their sequencing. Actions have preconditions, which specify in which states of the world they can be applied, as well as effects, which dictate how the world will be changed after the action is executed.

One of the most successful approaches to planning is encoding the planning problem into a series of satisfiability (SAT) formulas and then using a SAT solver to solve them. The method was first introduced by Kautz and Selman [1] and is still very popular and competitive. This is partly due to the power of SAT solvers, which are getting more efficient year by year. Since then many new improvements have been made to the method, such as new compact and efficient encodings [2–5], better ways of scheduling the SAT solvers [3] or modifying the SAT solver's heuristics to be more suitable for solving planning problems [6].

In this paper we present a new encoding scheme. It is inspired by the SASE transition-based encoding [2], which was the first SAT encoding based on the SAS+ planning formalism. The motivation for our work is to make the SASE encoding more robust by incorporating the strengths of older encoding schemes. We will prove the correctness of our encoding and compute an upper bound on the size of the encoded formula. In the experimental section of the paper we compare our new encoding to other SAS+ encodings on benchmark problems from the 2011 International Planning Competition (IPC) [7].

2. Preliminaries

In this section we give the basic definitions of satisfiability, and planning with parallel plans.

2.1. Satisfiability

A *Boolean variable* is a variable with two possible values *True* and *False*. A *literal* of a Boolean variable x is either x or $\neg x$ (*positive* or *negative literal*). A *clause* is a disjunction (OR) of literals. A clause with only one literal is called a *unit clause* and with two literals a *binary clause*. An implication of the form $x \Rightarrow (y_1 \vee \cdots \vee y_k)$ is equivalent to the clause $(\neg x \vee y_1 \vee \cdots \vee y_k)$. A *conjunctive normal form (*CNF*) formula* is a conjunction (AND) of clauses. A truth assignment ϕ of a formula F assigns a truth value to its variables. The assignment ϕ satisfies a positive (negative) literal if it assigns the value True (False) to its variable and ϕ satisfies a clause if it satisfies any of its literals. Finally, ϕ satisfies a CNF formula if it satisfies all of its clauses. A formula F is said to be satisfiable if there is a truth assignment ϕ that satisfies F. Such an assignment is called a *satisfying assignment*. The satisfiability problem (SAT) is to find a satisfying assignment of a given CNF formula or determine that it is unsatisfiable.

2.2. Planning

In the introduction we briefly described what planning is, in this section we give the formal definitions. We will use the multivalued SAS+ formalism [8] instead of the classical STRIPS formalism [9] based on propositional logic.

A planning task Π in the SAS+ formalism is defined as a tuple $\Pi = \{X, O, s_I, s_G\}$ where

- $X = \{x_1, \ldots, x_n\}$ is a set of multivalued variables with finite domains $\mathrm{dom}(x_i)$.

- O is a set of actions (or operators). Each action $a \in O$ is a tuple $(\text{pre}(a), \text{eff}(a))$ where $\text{pre}(a)$ is the set of preconditions of a and $\text{eff}(a)$ is the set of effects of a. Both preconditions and effects are of the form $x_i = v$ where $v \in \text{dom}(x_i)$.
- A state is a set of assignments to the state variables. Each state variable has exactly one value assigned from its respective domain. We denote by S the set of all states. $s_I \in S$ is the initial state. s_G is a partial assignment of the state variables (not all variables have assigned values) and a state $s \in S$ is a goal state if $s_G \subseteq s$.

An action a is *applicable* in the given state s if $\text{pre}(a) \subseteq s$. By $s' = \text{apply}(a, s)$ we denote the state after executing the action a in the state s, where a is applicable in s. All the assignments in s' are the same as in s except for the assignments in $\text{eff}(a)$ which replace the corresponding (same variable) assignments in s. If $P = [a_1 \dots a_k]$ is a sequence of actions, then $\text{apply}(P, s) = \text{apply}(a_k, \text{apply}(a_{k-1} \dots \text{apply}(a_2, \text{apply}(a_1, s)) \dots))$. A *sequential plan* P of *length* k for a given planning task Π is a sequence of k actions P such that $s_G \subseteq \text{apply}(P, s_I)$.

2.3. Parallel Plans

A *parallel plan* P with *makespan* k for a given planning task Π is a sequence of sets of actions (called parallel steps) $P = [A_1, \dots, A_k]$ such that $\trianglelefteq(A_1) \oplus \dots \oplus \trianglelefteq(A_k)$ is a sequential plan for Π, where $\trianglelefteq$ is an ordering function, which transforms a set of actions A_i into a sequence of actions $\trianglelefteq(A_i)$ and $\oplus$ denotes the concatenation of sequences.

Let us denote by s_j the world state in between the parallel steps A_j and A_{j+1}, which is obtained by applying the sequence $\trianglelefteq(A_j)$ on s_{j-1}, i.e., $s_j = \text{apply}(\trianglelefteq(A_j), s_{j-1})$ (except for $s_0 = s_I$). In this paper we will use the ∀-Step parallel planning semantics [10], which requires that each action $a \in A_j$ is applicable in the state s_j, the effects of all actions are applied in s_{j+1} and all possible orderings $\trianglelefteq$ of the sets A_j make valid sequential plans (hence the name ∀-Step semantics).

To ensure, that each ordering of the sets of actions in a parallel plan leads to a valid sequential plan, it is sufficient to check that the actions in each set are pairwise independent [3]. We say that two actions a_1 and a_2 are *independent* if they do not share common variables, i.e., $\text{scope}(a_1) \cap \text{scope}(a_2) = \emptyset$, where $\text{scope}(a) \subseteq X$ is a set of all state variables that appear in $\text{pre}(a)$ and $\text{eff}(a)$.

Note, that the pairwise independence of actions is a sufficient but not a necessary condition for the parallel steps in a ∀-Step semantics plan, as the following example demonstrates.

Example 1. *Let a_1 and a_2 be two actions such that $\text{pre}(a_1) = \text{pre}(a_2) = \{x = 1\}$, $\text{eff}(a_1) = \{y = 2\}$, and $\text{eff}(a_2) = \{z = 2\}$. Clearly, a_1 and a_2 are not independent (they share the variable x), however, they can be ordered arbitrarily to achieve the same changes between two given states.*

```
SP1  PlanningAsSat (Π)
SP2    k := 0
SP3    repeat
SP4      k := k + 1
SP5      F := encodeTaskWithMakespan(Π, k)
SP6    until isSatisfiable(F)
SP7    P := extractPlan(getSatAssignment(F))
SP8    return P
```

FIGURE 1. Pseudo-code of the basic planning as satisfiability algorithm.

The pairwise independence of actions in each step of a parallel plan also implies that they can be executed in parallel (at the same time).

3. Finding Plans using SAT

The basic idea of solving planning as SAT is the following [1]. We construct (by encoding the planning task) a series of SAT formulas $F_1, F_2, \dots$ such that F_i is satisfiable if there is a parallel plan of makespan $\leq i$. Then we solve them one by one starting from F_1 until we reach the first satisfiable formula F_k. From the satisfying assignment of F_k we can extract a plan of makespan k. The pseudo-code of this algorithm is presented in Figure 1

The method was first introduced by Kautz and Selman [1] and is still very popular and competitive. This is partly due to the power of SAT solvers, which are getting more efficient year by year. Since then many new improvements have been made to the method, such as new compact and efficient encodings [2–5], better ways of scheduling the SAT solvers [3] or modifying the SAT solver's heuristics to be more suitable for solving planning problems [6]. Clever ways of solver scheduling [3] can significantly improve the performance of the planning algorithm at the cost of possibly longer-makespan plans. Nevertheless, we will use the basic one-by-one scheduling since we are interested only in comparing the properties of encodings, i.e., the construction of the formulas. In the following section we will describe how a formula encoding a planning task can be constructed using our new Reinforced encoding.

4. Reinforced Encoding

Our goal is (given a planning task $\Pi = \{X, O, s_I, s_G\}$ and an integer k) to construct a CNF formula F_k such that F_k is satisfiable only if there is a parallel plan of at most k steps for Π. We also want to construct F_k in a way, that in the case it is satisfiable, we can easily extract a plan from its satisfying assignment. Before we describe the formula, we need to introduce the notion of transitions [2].

A *transition* represents a change of a state variable $x \in X$ from one value to another from its domain $\mathrm{dom}(x)$ or from an arbitrary value to a specific value. There are the following three kinds of transitions.

- An *active* transition changes the value of the variable x from d to e such that $d \neq e$, $\{d, e\} \subseteq \mathrm{dom}(x)$, it is denoted by $\delta_{x:\, d \to e}$. An action a has an active transition $\delta_{x:\, d \to e}$ if $(x = d) \in \mathrm{pre}(a)$ and $(x = e) \in \mathrm{eff}(a)$.
- A *prevailing* transition conserves the value of the variable x (if it was d, then it remains d, $d \in \mathrm{dom}(x)$), it is denoted by $\delta_{x:\, d \to d}$. An action a has a prevailing transition $\delta_{x:\, d \to d}$ if $(x = d) \in \mathrm{pre}(a)$ and there is no assignment related to x in $\mathrm{eff}(a)$.
- A *mechanical* transition changes the value of the variable x from any value to the value d ($d \in \mathrm{dom}(x)$), it is denoted by $\delta_{x:\, * \to d}$. An action a has a mechanical transition $\delta_{x:\, * \to d}$ if $(x = d) \in \mathrm{eff}(a)$ and there is no assignment related to x in $\mathrm{pre}(a)$.

Example 2. *The action a with preconditions $\mathrm{pre}(a) = \{x = 1, y = 3\}$ and effects $\mathrm{eff}(a) = \{y = 1, z = 2\}$ has one active transition ($\delta_{y:\, 3 \to 1}$), one prevailing transition ($\delta_{x:\, 1 \to 1}$), and one mechanical transition ($\delta_{z:\, * \to 2}$).*

The *transition set* of an action a is the set of all transitions that a has, it is denoted by Δ_a. By Δ_p we will mean the set of all possible prevailing transitions of a planning task, i.e., $\Delta_p = \{\delta_{x:\, d \to d} \mid x \in X, d \in \mathrm{dom}(x)\}$. The set of all transitions Δ is the union of all the prevailing transitions and the transition sets of all the actions $\Delta = \Delta_p \cup \{\Delta_a \mid a \in O\}$. By $\Delta_x \subseteq \Delta$ where $x \in X$ we will denote the set of all transitions related to the variable x.

The constructed formula F_k will have the following three kinds of Boolean variables.

- *Action variables* a_i^t indicating whether the i-th action is used in the t-th step. We will have one such variable for each action from the description of the planning task and for each of the k parallel steps.
- *Assignment variables* $b_{x=v}^t$ indicating whether the value of the variable x is equal to v in the end of the t-th step (after applying the actions of the t-th step). We will have one such Boolean variable for each state variable $x \in X$ and each value $v \in \mathrm{dom}(x)$ for each of the k parallel steps.
- *Transition variables* c_δ^t (or $c_{x:\, d \to e}^t$ where $\delta = \delta_{x:\, d \to e}$) indicating whether the transition δ occurred during the t-th step. We will have one such variable for each $\delta \in \Delta$ for each of the k parallel steps.

Now we are ready to define the clauses contained in F_k.

The following set of binary clauses will enforce, that at most one value is assigned to each state variable $x \in X$.

$$\begin{gathered}(\neg b_{x=v_i}^t \vee \neg b_{x=v_j}^t) \\ \forall x \in X,\ v_i \neq v_j,\ \{v_i, v_j\} \subseteq \mathrm{dom}(x),\ \forall t \in \{1, \ldots, k\}\end{gathered} \tag{1}$$

The following three kinds of clauses connect the assignment variables with the transition variables. The first set of clauses ensures that each transition $\delta_{x:\, d \to e}$ (including prevailing transitions $\delta_{x:\, e \to e}$ and mechanical transitions $\delta_{x:\, * \to e}$) implies that $x = e$ at the end of each step.

$$\begin{gathered}(\neg c_{\delta_{x:\, d \to e}}^t \vee b_{x=e}^t) \\ \forall \delta_{x:\, d \to e} \in \Delta, \forall t \in \{1, \ldots, k\}\end{gathered} \tag{2}$$

Similarly, we need to add clauses for each transition $\delta_{x:\, d \to e}$ (except for mechanical transitions) to enforce that $x = d$ holds at the end of the previous step, except for the first step, where we explicitly disable all the transitions that are not compatible with the initial state (using the clauses from equation 8).

$$\begin{gathered}(\neg c_{\delta_{x:\, d \to e}}^t \vee b_{x=d}^{t-1}) \\ \forall \delta_{x:\, d \to e} \in \Delta, d \neq *, \forall t \in \{2, \ldots, k\}\end{gathered} \tag{3}$$

The third kind of clauses is needed to guarantee, that if a variable x has the value v then there is a transition which changes the value of x to v.

$$\begin{gathered}(\neg b_{x=v}^t \vee c_{\delta_1}^t \vee \cdots \vee c_{\delta_m}^t) \\ \forall x \in X, v \in \mathrm{dom}(x), \delta_1, \ldots, \delta_m \text{ transform } x \text{ to } v, \\ \forall t \in \{1, \ldots, k\}\end{gathered} \tag{4}$$

Next we describe the clauses that connect the action variables with the transition variables. If an action a is selected, then all the transitions in its transition set Δ_a must be selected as well. This implication is expressed via the following clauses.

$$\begin{gathered}(\neg a^t \vee c_\delta^t) \\ \forall a \in O, \forall \delta \in \Delta_a, \forall t \in \{1, \ldots, k\}\end{gathered} \tag{5}$$

Also we need to make sure, that transitions (except for prevailing transitions) cannot happen without actions that have them in their transition sets. The following set of clauses will ensure this.

$$\begin{gathered}(\neg c_\delta^t \vee a_{s_1}^t \vee \cdots \vee a_{s_m}^t) \\ \forall \delta \in (\Delta \setminus \Delta_p), \mathrm{support}(\delta) = \{s_1, \ldots, s_m\}, \\ \forall t \in \{1, \ldots, k\}\end{gathered} \tag{6}$$

By $\mathrm{support}(\delta)$ we mean the set of indices of actions that have δ in their transition set, i.e., $\mathrm{support}(\delta) = \{i \mid a_i \in O;\ \delta \in \Delta_{a_i}\}$.

Next we need to deal with the interfering actions inside a parallel step. As discussed earlier, it is sufficient to ensure, that only pair-wise independent actions are together in each parallel step. We will achieve this by disabling all pairs of non-independent (interfering) actions. To extrude interfering actions from the parallel

steps we will add binary clauses for all the interfering action pairs.

$$(\neg a_i^t \vee \neg a_j^t)$$
$$\forall a_i, a_j \in O,\ a_i, a_j \text{ not independent},$$
$$\forall t \in \{1, \ldots, k\}$$

There might be a plenty of interfering action pairs producing a lot of clauses. But if we look carefully at the clauses we have already described, we can see, that most of the interfering actions cannot occur together anyway as we will show via the following notion of compatible actions.

Two sets of conditions (assignments) are compatible if they assign the same values to the variables they share. Two actions a_1 and a_2 are *compatible* if the preconditions of a_1 are compatible with the preconditions of a_2 and also the effects of a_1 are compatible with the effects of a_2.

Due to the clauses that enforce, that actions imply their transitions (5) and their connection to assignment variables (2 and 3) together with the clauses that forbid a state variable to have more than one value 1, actions that are not compatible cannot be in a parallel step together. Therefore it is enough to suppress compatible interfering action pairs.

$$(\neg a_i^t \vee \neg a_j^t)$$
$$\forall a_i, a_j \in O,\ a_i, a_j \text{ compatible and not independent},$$
$$\forall t \in \{1, \ldots, k\} \tag{7}$$

Lastly, we add the clauses that enforce the initial state to hold in the beginning and the goal conditions to be satisfied in the end. As for the initial state, we will disable all the transitions that are not compatible with the initial state, i.e., if a variable x has the value d in the initial state, then all the transitions that change x from a value other than d are disabled by using a unit clause. Note, that mechanical transitions are always compatible with the initial state (or any other state) and therefore no mechanical transition is disabled.

$$(\neg c^1_{\delta_{x:\ d \to e}})$$
$$\forall \delta_{x:\ d \to e} \in \Delta, (x = d) \notin s_I \tag{8}$$

To encode the goal conditions we will use unit clauses with assignment variables. Fore each goal condition $(x = v) \in s_G$ we will have a unit clause $(b^k_{x=v})$ which forces the value of x to be v after the last parallel step.

$$(b^k_{x=v})$$
$$\forall (x = v) \in s_G \tag{9}$$

The formula F_k for the Reinforced encoding is a conjunction of the clauses defined in equations 1, 2, 3, 4, 5, 6, 7, 8, and 9. A ∀-step parallel plan can be extracted from any satisfying assignment of F_k in the following way. Let ϕ be a satisfying assignment of F_k. P_ϕ is a sequence of action sets such that its t-th set contains those actions $a_i \in O$ for which $\phi(a_i^t) = True$.

4.1. Correctness

In this subsection the prove the correctness of our encoding, i.e., the following proposition.

Proposition 1. *If the formula F_k obtained using the Reinforced encoding of the planning task Π is satisfied by a truth assignment ϕ then P_ϕ is a valid ∀-Step parallel plan of makespan k for the planning task Π.*

Proof. The requirements for the action sets given by the ∀-Step semantics are clearly satisfied:

- the preconditions of actions in each parallel step are satisfied due to 2, 3, and 5
- the effects are propagated also due to 2, 3, and 5
- the actions can be ordered arbitrarily thanks to 7

It remains to prove that $P^S_\phi = [\trianglelefteq(A_1) \oplus \cdots \oplus \trianglelefteq(A_k)]$ is a valid (sequential) plan for Π, where $\oplus$ denotes the concatenation of sequences and $\trianglelefteq$ is an arbitrary ordering of an action set.

Let us observe, that the transitions of the state variables are consistent at each step, i.e., exactly one transition is allowed for each state variable (due to 2, 3, and 1) and a non-prevailing transition cannot happen without an action that has it (thanks to 6). Prevailing transitions do not have to be supported by any actions since they are used to preserve the values of the variables that are not changed in the given step by any actions. Furthermore, all the transitions between two neighboring parallel steps must be compatible due to 2, 3, 4 and 1. Note, that it may happen, that a variable x has no value assigned in the end of a step t (all the $b^t_{x=v}, v \in \mathrm{dom}(x)$ are False) and no transition related to the variable is selected (all the $c^t_\delta, \delta \in \Delta_x$ are False). However, this can only occur for variables that are not used in the goal conditions or by actions that appear in the t-th step or later.

Since the action variables imply the proper transition variables thanks to 5, the actions must be applicable if their action variable is True and also the transition connected to the action must happen.

Thanks to 8 only transitions compatible with the initial state can happen in the first step and because of 2 and 9 only transitions that change the variables to their goal values are allowed in the last step. This fact together with the consistency of the transitions during all the k steps implies the validity of P^S_ϕ for the planning task Π. ∎

The reversed implication, which is that if P_ϕ is a valid ∀-Step parallel plan of makespan k, then ϕ satisfies F_k, does not hold. This is because P_ϕ may contain a non-independent pair of actions in one of its steps and still be a valid ∀-Step plan (see Example 1). Such a pair of actions would make one of the clauses of type 7 unsatisfied.

4.2. Size of The Encoded Formula

The size of the formulas will of course depend on the parameters of the planning task being encoded. We will use the following quantitative properties of a planning task $\Pi = (X, O, s_I, s_G)$ to compute the upper bounds.

- n - The number of actions ($n = |O|$).
- v - The number of state variables ($v = |X|$).
- d - The maximum domain size of any state variable ($d = \max_{x \in X}\{|\operatorname{dom}(x)|\}$).
- p - The maximum number of preconditions or effects an action has ($p = \max_{a \in O}\{|\operatorname{pre}(a)|, |\operatorname{eff}(a)|\}$)

Typically, the number of actions is much higher than the other parameters. From these values we can compute the following upper bounds related to the planning task.

- The number of assignments is at most vd.
- The number of transitions is at most $v(d^2 + d)$ since there are at most $d(d-1)$ active, d prevailing, and d mechanical transitions for each variable.

From these bounds it is apparent, that F_k (a formula for makespan k) has at most kn action variables, kvd assignment variables, and $kv(d^2 + d)$ transition variables. Therefore the total number of Boolean variables in F_k is $k(n + vd(d + 2))$.

Now let us compute an upper bound on the number of clauses in F_k. We will count separately the number of unit clauses (clauses with one literal), binary clauses (clauses with two literals), and Horn clauses (clauses with at most one positive literal). The formula F_k obtained by the Reinforced encoding is the conjunction of the clauses defined in equations 1, 2, 3, 4, 5, 6, 7, 8, and 9

- There are at most kvd^2 clauses of the type 1 – one for each step and variable and two different values from its domain. These clauses are binary and Horn.
- There are at most $kv(d^2 + d)$ clauses of the both type 2 and type 3 – one for each step and transition. These clauses are binary and Horn.
- There are at most kvd clauses of the type 4 – one for each step and assignment.
- There are at most $2knp$ clauses of the type 5 – one for each step, action and each of its transitions (there are at most $2p$ transitions connected to each action). These clauses are binary and Horn.
- There are at most $kv(d^2 + d)$ clauses of the type 6 – one for each step and transition.
- There are at most kn^2 clauses of the type 7 – one for each step, and each pair of compatible interfering actions (at most each pair of actions). These clauses are binary and Horn.
- There are at most $v(d^2 + d)$ clauses of type 8 – one for each transition that is not compatible with the initial state (at most all the transitions). These are unit clauses.
- There are at most v clauses of the type 9 – one for each goal condition. These clauses are unit.

In total we have $k(n^2 + 2np + 4vd^2 + 4vd) + vd^2 + vd + v$ clauses, from which $vd^2 + vd + v$ are unit clauses and $k(n^2 + 2np + 3vd^2 + 2vd)$ are both binary and Horn clauses.

5. Experimental Evaluation

To evaluate the performance of our new Reinforced encoding, we compared it with three other SAS+ based encodings of planning as SAT. We ran experiments with a 30 minutes time limit using the following four encodings.

- Reinforced Encoding (Reinf). A Java implementation of our new Reinforced encoding as described in the previous section.
- Direct Encoding (Dir). We implemented a simple encoding based on the historically first encoding of planning as SAT [1]. We adapted it for the SAS+ formalism. This encoding is similar to our Reinforced encoding but uses only action and assignment variables.
- SASE Encoding (SASE). Our Java implementation[1] of the transition-based SASE encoding [2]. This encoding uses only action and transition variables.
- $R^2\exists$-Step Encoding ($R^2\exists$). The original Java implementation of the $R^2\exists$-Step encoding [5]. This encoding differs from the previous three encoding significantly since it uses a different parallel planning semantics. The $R^2\exists$-Step encoding allows more actions inside the parallel steps, therefore it often finds plans with much lower makespans. Lower makespan indicates that fewer SAT solver calls are required to find a plan, however, it does not say anything about its length, i.e., the total number of actions it contains.

5.1. Experimental Setting

To compare the performance of the encodings we created a simple script, which iteratively constructed and solved the formulas for time steps $1, 2, \ldots$ until a satisfiable formula was reached (see Figure 1). For each encoding we used the same SAT solver – Lingeling[11] (version ats).

The time limit was 30 minutes for the SAT solving part, i.e., the total time the SAT solver could spend solving the formulas $F_1, F_2, \ldots$ for each problem instance was 30 minutes. The time required for the generation of $F_1, F_2, \ldots$ is usually negligible compared to the time required to solve them and therefore we will ignore it. Hence the overall planning time could exceed the given time limit for a problem instance.

[1] The original SASE implementation cannot be used since it does not support the format of the latest benchmark problems

Domain	Dir	SASE	Reinf	$R^2\exists$
barman	4	4	4	**8**
elevators	**20**	**20**	**20**	**20**
floortile	16	11	**18**	**18**
nomystery	**20**	10	**20**	6
openstacks	0	0	0	**15**
parcprinter	**20**	**20**	**20**	**20**
parking	0	0	0	0
pegsol	10	6	10	**19**
scanalyzer	14	12	**15**	9
sokoban	**2**	**2**	**2**	**2**
tidybot	**2**	**2**	**2**	**2**
transport	16	17	**18**	13
visitall	12	9	10	**20**
woodworking	**20**	**20**	**20**	**20**
Total	156	133	159	**172**

TABLE 1. The number of problems (out of 20) in each domain that the encodings solved within the time limit (30 minutes for SAT solving).

Domain	Dir	SASE	Reinf	$R^2\exists$
barman	121	121	121	84
elevators	190	190	190	**85**
floortile	302	181	344	169
nomystery	347	119	347	30
openstacks	-	-	-	93
parcprinter	261	261	261	**30**
parking	-	-	-	-
pegsol	222	131	222	158
scanalyzer	83	61	95	17
sokoban	60	60	60	**27**
tidybot	14	15	15	**6**
transport	221	242	262	55
visitall	223	110	146	34
woodworking	68	68	68	**33**

TABLE 2. The sum of makespans of the plans found within the time limit for each domain. Lower makespan means fewer SAT solver calls, it does not indicate better plan quality. Domains with the same number of solved problems for each encoding are highlighted.

The experiments were run on a computer with Intel i7 920 CPU @ 2.67 GHz processor and 6 GB of memory.

The benchmark problems of the IPC are organized into domains. Each domain contains 20 problems and there are 14 domains which results in a total of 280 problems. The benchmark problems are provided in the PDDL format, however, the encodings require input in the SAS+ format. We used Helmert's translation tool, which is a part of the Fast Downward planning system [12], to obtain the SAS+ files from the PDDL files. The translation is very fast requiring only a few seconds for all domains.

5.2. Experimental Results

The number of solved instances in presented in Table 1. Looking at the results from the perspective of the domains, we can observe, that the elevators, parcprinter, and woodworking domains are entirely solved by every encoding. On the other hand, the parking domain is so difficult that not even a single problem is solved by any of the encodings. The openstacks domain is very difficult for all but the $R^2\exists$-Step encoding. The sokoban and tidybot domains are also very hard for all of the encodings, only two of the twenty problems are solved by each encoding.

If we compare the encodings, we can observe that the $R^2\exists$-Step encoding has the highest total number of solved instances followed by our new Reinforced encoding. As for the individual domains, the $R^2\exists$-Step encoding solves strictly more problems than the other encodings in four cases. The Reinforced encoding achieves this for three domains, while the Direct and SASE encoding cannot outperform the other encodings in any of the domains. The Reinforced encoding solves the same number of problems as any other encoding in seven cases. Except for the visitall domain, the Reinforced encoding is never worse than the Direct or SASE encoding.

Looking at the makespans of found plans displayed in Table 2 we can observe that the makespans for the $R^2\exists$-Step plans are indeed significantly lower than the makespans of plans found by the other three encodings. As expected, in the cases when the Direct, SASE, and Reinforced encodings solve all the problems (or solve the same problems) their total makespans are identical. This is due to the fact that these three encodings use the same $\forall$-Step parallel planning semantics.

The times required to solve the problems are presented in Table 3. If we look at the results for the domains, where each encoding solved the same number of problems, i.e., the highlighted domains, we can notice, that except for the parcprinter and sokoban problems, the runtime of the $R^2\exists$-Step encoding is much higher than the runtime of the other methods. If we also look at Table 2, which contains the total makespan of the found plans, we can deduce, that lower makespan, i.e., fewer SAT calls does not necessarily mean faster planning, especially not in the case of the easy domains. Nevertheless, for the domains, where $R^2\exists$-Step significantly outperformed the other methods – openstacks, pegsol, and visitall, the makespans are much lower than the makespans of the other methods, despite the fact, that they solved fewer problems.

Domain	Dir	SASE	Reinf	$R^2\exists$
barman	2041.44	1549.31	1680.53	2999.30
elevators	**33.96**	55.72	38.10	288.29
floortile	7327.15	2083.57	1206.18	1380.63
nomystery	3798.45	1377.23	1894.65	1927.40
openstacks	-	-	-	2679.68
parcprinter	10.07	28.25	12.15	**4.24**
parking	-	-	-	-
pegsol	3796.11	2096.72	4971.97	830.72
scanalyzer	1401.21	831.80	1435.19	1118.89
sokoban	592.03	1337.87	857.04	**550.18**
tidybot	75.68	**74.02**	118.09	480.85
transport	1404.23	3554.90	3203.62	7418.01
visitall	2380.76	683.25	726.53	14.24
woodworking	2.57	**2.04**	3.84	138.61

TABLE 3. The time in seconds required to solve all the problems that were solved within the time limit. The presented time is the sum of times the SAT solver alone required, formula generation time is not included. Domains with the same number of solved problems for each encoding are highlighted.

6. CONCLUSION

In this paper we have introduced a new encoding of a planning problem represented in the SAS+ formalism into SAT. Our new encoding performs well on the benchmark problems of the 2011 International Planning Competition. It can strictly outperform all the other evaluated SAS+ encodings in three domains and solve the same number of problems as any other encoding for seven domains out of fourteen. On the remaining four domains our encoding is outperformed by the $R^2\exists$-Step encoding which uses a different parallel planning semantics. As for future work, we believe that the Reinforced encoding can be improved by decreasing the number of its clauses by using a more compact way of encoding of the action interference constraints.

ACKNOWLEDGEMENTS

The research is supported by the Czech Science Foundation under the contract P103/10/1287 and by the Grant Agency of Charles University under contracts no. 600112 and no. 390214. This research was also supported by the SVV project number 260 104.

REFERENCES

[1] H. A. Kautz, B. Selman. Planning as satisfiability. In *ECAI*, pp. 359–363. 1992. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.9443.

[2] R. Huang, Y. Chen, W. Zhang. A novel transition based encoding scheme for planning as satisfiability. In M. Fox, D. Poole (eds.), *AAAI*. AAAI Press, 2010. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.182.9561.

[3] J. Rintanen, K. Heljanko, I. Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artif Intell* **170**(12-13):1031–1080, 2006. DOI:10.1016/j.artint.2006.08.002.

[4] N. Robinson, C. Gretton, D. N. Pham, A. Sattar. Sat-based parallel planning using a split representation of actions. In A. Gerevini, A. E. Howe, A. Cesta, I. Refanidis (eds.), *ICAPS*. AAAI, 2009. http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/732.

[5] T. Balyo. Relaxing the relaxed exist-step parallel planning semantics. In *ICTAI*, pp. 865–871. IEEE, 2013. DOI:10.1109/ICTAI.2013.131.

[6] J. Rintanen. Planning as satisfiability: Heuristics. *Artif Intell* **193**:45–86, 2012. DOI:10.1016/j.artint.2012.08.001.

[7] C. L. López, S. J. Celorrio, Á. G. Olaya. The deterministic part of the seventh international planning competition. *Artificial Intelligence* 2015. DOI:10.1016/j.artint.2015.01.004.

[8] C. Bäckström, B. Nebel. Complexity results for sas+ planning. *Computational Intelligence* **11**:625–656, 1995. DOI:10.1111/j.1467-8640.1995.tb00052.x.

[9] R. Fikes, N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artif Intell* **2**(3/4):189–208, 1971. DOI:10.1016/0004-3702(71)90010-5.

[10] A. Blum, M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence* **90**(1-2):281–300, 1997. DOI:10.1016/S0004-3702(96)00047-1.

[11] A. Biere. Lingeling and plingeling home page, 2015. http://fmv.jku.at/lingeling/.

[12] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)* **26**:191–246, 2006. DOI:10.1613/jair.1705.