

# Parallel Encodings of Classical Planning as Satisfiability

Jussi Rintanen<sup>1</sup>, Keijo Heljanko<sup>2</sup>, and Ilkka Niemelä<sup>2</sup>

<sup>1</sup> Albert-Ludwigs-Universität Freiburg  
Institut für Informatik, Georges-Khler-Allee  
79110 Freiburg im Breisgau  
Germany

<sup>2</sup> Helsinki University of Technology  
Laboratory for Theoretical Computer Science  
P. O. Box 5400, FI-02015 HUT  
Finland

**Abstract.** We consider a number of semantics for plans with parallel operator application. The standard semantics used most often in earlier work requires that parallel operators are independent and can therefore be executed in any order. We consider a more relaxed definition of parallel plans, first proposed by Dimopoulos et al., as well as normal forms for parallel plans that require every operator to be executed as early as possible. We formalize the semantics of parallel plans emerging in this setting, and propose effective translations of these semantics into the propositional logic. And finally we show that one of the semantics yields an approach to classical planning that is sometimes much more efficient than the existing SAT-based planners.

## 1 Introduction

Satisfiability planning [6] is a leading approach to solving difficult planning problems. An important factor in its efficiency is the notion of parallel plans [6, 2]. The standard parallel encoding, the *state-based encoding* [6], allows the simultaneous execution of a set of operators as long as the operators are mutually non-interfering. This condition guarantees that any total ordering on the simultaneous operators is a valid execution and in all cases leads to the same state. We call this semantics of parallelism *the step semantics*. Two benefits of this form of parallelism in planning as satisfiability are that, first, it is unnecessary to consider all possible orderings of a set of non-interfering operators, and second, less clauses and propositional variables are needed as the values of the state variables in the intermediate states need not be represented.

In this paper we formalize two more refined parallel semantics for AI planning and present efficient encodings of them in the propositional logic. Both of the semantics are known from earlier research but the first, *process semantics*, has not been considered in connection with planning, and the second, *1-linearization semantics*, has not been given efficient encodings in SAT/CSP before. With our new encoding this semantics dramatically outperforms the other semantics and encodings given earlier. Our main innovations here are the definition of *disabling graphs* and the use of the strong components (or strongly connected components SCCs) of these graphs to derive very efficient encodings.

The two semantics considered in this paper are orthogonal refinements of the step semantics. The process semantics is stricter than the step semantics in that it requires all actions to be taken as early as possible. Process semantics was first introduced for Petri nets; for an overview see [1]. Heljanko [5] has applied this semantics to the deadlock detection of 1-safe Petri nets and has shown that it leads to big efficiency gains on many types of problems in bounded model-checking.

The idea of the 1-linearization semantics was proposed by Dimopoulos et al. [4]. They pointed out that it is not necessary to require that all parallel operators are non-interfering as long as the parallel operators can be executed in at least one order. They also showed how blocks worlds problems can be modified to satisfy this condition and that the reduction in the number of time points improves runtimes. Until now the application of 1-linearization in satisfiability planning had been hampered by the cubic size of the obvious encodings. We give more compact encodings for this semantics and show that this often leads to dramatic improvements in planning efficiency. Before the developments reported in this paper, this semantics had never been used in an automated planner that is based on a declarative language like the propositional logic.

The structure of this paper is as follows. In Section 4 we discuss the standard step semantics of parallel plans and its encoding in the propositional logic. Section 5 introduces the underlying ideas of the process semantics and discusses its representation in the propositional logic. In Section 6 we present the 1-linearization refinement to step semantics and its encoding. Section 7 evaluates the advantages of the different semantics in terms of some planning problems. Section 8 discusses related work.

## 2 Notation

We consider planning in a setting where the states of the world are represented in terms of a set  $P$  of Boolean state variables that take the value *true* or *false*. Each *state* is a valuation of  $P$ , that is, an assignment  $s : P \rightarrow \{T, F\}$ . We use *operators* for expressing how the state of the world can be changed.

**Definition 1.** An operator on a set of state variables  $P$  is a triple  $\langle p, e, c \rangle$  where

1.  $p$  is a propositional formula on  $P$  (the precondition),
2.  $e$  is a set of literals on  $P$  (unconditional effects), and
3.  $c$  is a set of pairs  $f \triangleright d$  (conditional effects) where  $f$  is a propositional formula on  $P$  and  $d$  is a set of literals on  $P$ .

For an operator  $\langle p, e, c \rangle$  its *active effects* in state  $s$  are

$$e \cup \bigcup \{d \mid f \triangleright d \in c, s \models f\}.$$

The operator is *applicable* in  $s$  if  $s \models p$  and its set of active effects in  $s$  is consistent (does not contain both  $p$  and  $\neg p$  for any  $p \in P$ .) If this is the case, then we define  $\text{app}_o(s) = s'$  as the unique state that is obtained from  $s$  by making the active effects of  $o$  true and retaining the truth-values of the state variables not occurring in the active effects. For sequences  $o_1; o_2; \dots; o_n$  of operators we define  $\text{app}_{o_1; o_2; \dots; o_n}(s)$  as

$\text{app}_{o_n}(\dots \text{app}_{o_2}(\text{app}_{o_1}(s)) \dots)$ . For sets  $S$  of operators and states  $s$  we define  $\text{app}_S(s)$ : the result of simultaneously applying all operators  $o \in S$ . We require that  $\text{app}_o(s)$  is defined for every  $o \in S$  and that the set of active effects of all operators in  $S$  is consistent. Different semantics of parallelism impose further restrictions on sets  $S$ .

Let  $\pi = \langle P, I, O, G \rangle$  be a *problem instance*, consisting of a set  $P$  of state variables, a state  $I$  on  $P$  (the initial state), a set  $O$  of operators on  $P$ , and a formula  $G$  on  $P$  (the goal formula). A (sequential) *plan* for  $\pi$  is a sequence  $\sigma = o_1; \dots; o_n$  of operators from  $O$  such that  $\text{app}_\sigma(I) \models G$ , that is, applying the operators in the given order starting in the initial state is defined (precondition of every operator is true when the operator is applied) and produces a state that satisfies the goal formula.

In the rest of this paper we also consider plans that are sequences of *sets of operators*, so that at each execution step all operators in the set are simultaneously applied. The different semantics discussed in the next sections impose further constraints on these sets.

### 3 Planning as Satisfiability

Planning can be performed by propositional satisfiability testing as follows. Produce formulae  $\phi_0, \phi_1, \phi_2, \dots$  such that  $\phi_i$  is satisfiable if there is a plan of length  $i$ . The formulae are tested for satisfiability in the order of increasing plan length, and from the first satisfying assignment that is found a plan is constructed. Length  $i$  of a plan means that there are  $i$  time points in which a *set* of operators is applied simultaneously. There are alternative semantics for this kind of parallel plans and their encodings in the propositional logic differ only in axioms restricting simultaneous application of operators. Next we describe the part of the encodings shared by all the semantics.

The state variables in a problem instance are  $P = \{a^1, \dots, a^n\}$  and the operators are  $O = \{o^1, \dots, o^m\}$ . For a state variable  $a$  we have the propositional variable  $a_t$  that expresses the truth-value of  $a$  at time point  $t$ . Similarly, for an operator  $o$  we have  $o_t$  for expressing whether  $o$  is applied at  $t$ . For formulae  $\phi$  we denote the formula with all propositional variables subscripted with the index to a time point  $t$  by  $\phi_t$ .

A formula is generated to answer the following question. Is there an execution of a sequence of sets of operators taking  $l$  time points that reaches a state satisfying  $G$  from the initial state  $I$ ? The formula is conjunction of  $I_0$  (formula describing the initial state with propositions marked with time point 0),  $G_l$ , and the formulae described below, instantiated with all  $t \in \{0, \dots, l-1\}$ .

First, for every  $o = \langle p, e, c \rangle \in O$  there are the following axioms. The precondition  $p$  has to be true when the operator is applied.

$$o_t \rightarrow p_t \tag{1}$$

If  $o$  is applied, then its (unconditional) effects  $e$  are true at the next time point.

$$o_t \rightarrow e_{t+1} \tag{2}$$

Here we view the set  $e$  of literals as a conjunction of literals. For every  $f \triangleright d \in c$  the effects  $d$  will be true if the antecedent  $f$  is true at the preceding time point.

$$(o_t \wedge f_t) \rightarrow d_{t+1} \tag{3}$$

Second, the value of a state variable does not change if no operator that changes it is applied. Hence for every state variable  $a$  we have two formulae, one expressing the conditions for the change of  $a$  to false from true, and another from true to false. The formulae are analogous, and here we only give the one for change from true to false:

$$(a_t \wedge \neg a_{t+1}) \rightarrow ((o_t^1 \wedge \phi_t^1) \vee \dots \vee (o_t^m \wedge \phi_t^m)) \quad (4)$$

where  $\phi^i$  expresses the condition under which operator  $o^i$  changes  $a$  from true to false. So let  $o^i = \langle p, e, c \rangle$ . If  $a$  is a negative effect in  $e$  then simply  $\phi^i = \top$ . Otherwise, the change takes place if one of the conditional effects is active. Let  $f^1 \triangleright d^1, \dots, f^k \triangleright d^k$  be the conditional effects with  $a$  as a negative effect in  $d^j$ . Here  $k \geq 0$ . Then  $\phi^i = f^1 \vee \dots \vee f^k$ . The empty disjunction with  $k = 0$  is the constant false  $\perp$ .

Finally, we need axioms for restricting the parallel application of operators: we will describe them in the next sections for each semantics. The resulting set of formulae is satisfiable if and only if there is an operator sequence taking  $l$  time points that reaches a goal state from the initial state.

In addition to the above axioms, which are necessary to guarantee that the set of satisfying assignments exactly corresponds to the set of plans with  $l$  time points, it is often useful to add further constraints that do not affect the set of satisfying assignments but instead help in pruning the set of incomplete solutions need to be looked at, and thereby speed up plan search. The most important type of such constraints for many planning problems is invariants, which are formulae that are true in all states reachable from the initial state. Typically, one uses only a restricted class of invariants that are efficient (polynomial time) to identify. There are efficient algorithms for finding many invariants that are 2-literal clauses [8, 2]. In the experiments in Section 7 we use formulae  $l_t \vee l'_t$  for invariants  $l \vee l'$  as produced by the algorithm by Rintanen [8].

## 4 Step Semantics

In this section we formally present a semantics that generalizes the semantics used in most works on parallel plans, for example Kautz and Selman [6]. Practical implementations of satisfiability planning approximate this semantics as described in Section 4.1.

For defining parallel plans under step semantics, we need to define when operators interfere in a way that makes their simultaneous application unwanted.

**Definition 2 (Interference).** Operators  $o_1 = \langle p_1, e_1, c_1 \rangle$  and  $o_2 = \langle p_2, e_2, c_2 \rangle$  interfere in state  $s$  if

1.  $s \models p_1 \wedge p_2$ ,
2. the set  $e_1 \cup \{d \mid f \triangleright d \in c_1, s \models f\} \cup e_2 \cup \{d \mid f \triangleright d \in c_2, s \models f\}$  is consistent, and
3. the operators are not applicable in both orders or applying them in different orders leads to different results, that is, at least one of the following holds:
  - (a)  $app_{o_1}(s) \not\models p_2$ ,
  - (b)  $app_{o_2}(s) \not\models p_1$ , or
  - (c) active effects of  $o_2$  are different in  $s$  and in  $app_{o_1}(s)$  or active effects of  $o_1$  are different in  $s$  and in  $app_{o_2}(s)$ .

The first two conditions are the *applicability conditions* for parallel operators: preconditions have to be satisfied and the effects may not contradict each other. The third condition says that interference is the impossibility to interpret parallel application as application in any order,  $o_1$  followed by  $o_2$ , or  $o_2$  followed by  $o_1$ , leading to the same state in both cases: one execution order is impossible or the resulting states may be different.

The conditions guarantee that two operators may be exchanged in any total ordering of a set of pairwise non-interfering operators without changing the state that is reached.

**Definition 3 (Step plans).** For a set of operators  $O$  and an initial state  $I$ , a plan is a sequence  $T = S_1, \dots, S_l$  of sets of operators such that there is a sequence of states  $s_0, \dots, s_l$  (the execution of  $T$ ) such that

1.  $s_0 = I$ ,
2.  $s_{i-1} \models p$  for all  $i \in \{1, \dots, l\}$  and  $\langle p, e, c \rangle \in S_i$ ,
3.  $\bigcup_{\langle p, e, c \rangle \in S_i} (e \cup \bigcup \{d \mid f \triangleright d \in c, s_{i-1} \models f\})$  is consistent for every  $i \in \{1, \dots, l\}$ ,
4.  $s_i = \text{app}_{S_i}(s_{i-1})$  for all  $i \in \{1, \dots, l\}$ , and
5. for all  $i \in \{1, \dots, l\}$  and  $o, o' \in S_i$  and  $S \subseteq S_i \setminus \{o, o'\}$ ,  $o$  and  $o'$  are applicable in  $\text{app}_S(s_{i-1})$  and do not interfere in  $\text{app}_S(s_{i-1})$ .

*Example 1.* Consider  $S_1 = \{o_1, o_2, o_3\}$  where  $o_1 = \langle p \vee \neg p, \{q\}, \emptyset \rangle$ ,  $o_2 = \langle p \vee \neg p, \emptyset, \{q \triangleright p\} \rangle$ , and  $o_3 = \langle p \vee \neg p, \emptyset, \{p \triangleright r\} \rangle$ .  $S_1$  cannot be the first step of a step plan starting from an initial state  $I$  in which  $p, q$  and  $r$  are false because the fifth condition of step plans is not satisfied: there is  $S = \{o_1\} \subseteq S_1$  such that  $o_2$  and  $o_3$  interfere in  $\text{app}_S(I)$ , because applying  $o_3$  in  $\text{app}_S(I)$  has no effect but applying  $o_3$  in  $\text{app}_{o_2}(\text{app}_S(I))$  makes  $r$  true.

**Lemma 1.** Let  $T = S_1, \dots, S_k, \dots, S_l$  be a step plan. Let  $T' = S_1, \dots, S_k^0, S_k^1, \dots, S_l$  be the step plan obtained from  $T$  by splitting the step  $S_k$  into two steps  $S_k^0$  and  $S_k^1$  such that  $S_k = S_k^0 \cup S_k^1$  and  $S_k^0 \cap S_k^1 = \emptyset$ .

If  $s_0, \dots, s_k, \dots, s_l$  is the execution of  $T$  then  $s_0, \dots, s'_k, s_k, \dots, s_l$  for some  $s'_k$  is the execution of  $T'$ .

*Proof.* So  $s'_k = \text{app}_{S_k^0}(s_{k-1})$  and  $s_k = \text{app}_{S_k}(s_{k-1})$  and we have to prove that  $\text{app}_{S_k^1}(s'_k) = s_k$  and operators in  $S_k^1$  are applicable in  $s'_k$ . For the first we will show that the active effects of every operator in  $S_k^1$  are the same in  $s_{k-1}$  and in  $s'_k$ , and hence the changes from  $s_{k-1}$  to  $s_k$  are the same in both plans. Let  $o^1, \dots, o^z$  be the operators in  $S_k^0$  and let  $T_i = \{o^1, \dots, o^i\}$  for every  $i \in \{0, \dots, z\}$ . We show by induction that the active effects of every operator in  $S_k^1$  are the same in  $s_{k-1}$  and  $\text{app}_{T_i}(s_{k-1})$  and that every operator in  $S_k^1$  is applicable in  $\text{app}_{T_i}(s_{k-1})$ , from which the claim follows as  $s'_k = \text{app}_{T_z}(s_{k-1})$ .

Base case  $i = 0$ : Immediate because  $T_0 = \emptyset$ .

Inductive case  $i \geq 1$ : By the induction hypothesis the active effects of every operator  $o \in S_k^1$  are the same in  $s_{k-1}$  and in  $\text{app}_{T_{i-1}}(s_{k-1})$  and  $o$  is applicable in  $\text{app}_{T_{i-1}}(s_{k-1})$ . In  $\text{app}_{T_i}(s_{k-1})$  additionally the operator  $o^i$  has been applied. We have to show that this operator application does not affect the set of active effects of  $o$  nor does it disable  $o$ . By the definition of step plans, the operators  $o$  and  $o^i$  do not interfere in  $\text{app}_{T_{i-1}}(s_{k-1})$ , and

hence the active effects of  $o$  are the same in  $\text{app}_{T_{i-1}}(s_{k-1})$  and in  $\text{app}_{T_{i-1} \cup \{o^i\}}(s_{k-1})$  and  $o^i$  does not disable  $o$ . This completes the induction and the proof.

**Theorem 1.** *Let  $T = S_1, \dots, S_k, \dots, S_l$  be a step plan. Then any  $\sigma = o_1^1; \dots; o_{n_1}^1; o_2^2; \dots; o_{n_2}^2; \dots; o_1^l; \dots; o_{n_l}^l$  such that for every  $i \in \{1, \dots, l\}$  the sequence  $o_1^i; \dots; o_{n_i}^i$  is a total ordering of  $S_i$ , is a plan, and its execution leads to the same final state as that of  $T$ .*

*Proof.* First all empty steps are removed from the step plan. By Lemma 1 non-singleton steps can be split repeatedly to smaller non-empty steps until every step is singleton and the desired ordering is obtained. The resulting plan is a sequential plan.

#### 4.1 Encoding in the Propositional Logic

Definition 3 provides a notion of step plans that attempts to maximize parallelism. Most works on satisfiability planning [6] use a simple syntactic condition for guaranteeing non-interference, approximating Condition 3 in Definition 2. For example, the simultaneous application of two operators is forbidden always when a state variable affected by one operator occurs in the precondition or in the antecedents of conditional effects of the other. In our experiments in Section 7 we include the formula  $\neg o_t \vee \neg o'_t$  for any such pair of operators  $o$  and  $o'$  with mutually non-contradicting effects and mutually non-contradicting preconditions to guarantee that we get step plans. There are  $O(m^2)$  such constraints for  $m$  operators.

### 5 Process Semantics

The idea of process semantics is that we only consider those step plans that fulfill the following condition. There is no operator  $o$  applied at time  $t + 1$  with  $t \geq 0$  such that the sequence of sets of operators obtained by moving  $o$  from time  $t + 1$  to time  $t$  would be a step plan according to Definition 3.

The important property of process semantics is that even though the additional condition reduces the number of acceptable plans, whenever there is a plan with  $t$  time steps under step semantics, there is also a plan with  $t$  time steps under process semantics. A plan satisfying the process condition is obtained from a step plan by repeatedly moving operators violating the condition one time point earlier. As a result of this procedure a step plan satisfying the process condition is obtained which is greedy in the sense that it applies each operator of the original step plan as early as possible.

As an example consider a set  $S$  in which no two operators interfere nor have contradicting effects and are applicable in state  $s$ . If we have time points 0 and 1, we can apply each operator alternatively at 0 or at 1. The resulting state at time point 2 will be the same in all cases. So, under step semantics the number of equivalent plans on two time points is  $2^{|S|}$ . Process semantics says that no operator that is applicable at 0 may be applied later than at 0. Under process semantics there is only one plan instead of  $2^{|S|}$ .

## 5.1 Encoding in the Propositional Logic

The encoding of process semantics extends the encoding of step semantics, so we take all axioms for the latter and have further axioms specific to process semantics.

The axioms for process semantics deny the application of an operator  $o$  at time  $t + 1$  if it can be shown that moving  $o$  to time  $t$  would also be a valid step plan according to Definition 3. The idea is that if an operator  $o$  is delayed to be applied at time  $t + 1$  then there must be some operator  $o'$  applied at time  $t$  which is the reason why  $o$  cannot be moved to time  $t$ . More precisely, (i)  $o'$  may have enabled  $o$ , (ii)  $o'$  may have conflicting effects with  $o$ , or (iii) moving  $o$  to time  $t$  might make it interfere with  $o'$  according to Condition 5 of Definition 3.

We cautiously approximate the process semantics using simple syntactic conditions to compute for each operator  $o$  the set of operators  $\{o^1, \dots, o^n\}$  which are the potential reasons why  $o$  cannot be moved one time point earlier. Then we have the following axioms guaranteeing that the application of  $o$  is delayed only when there is a reason for this.

$$o_{t+1} \rightarrow (o_t^1 \vee o_t^2 \vee \dots \vee o_t^n)$$

These disjunctions may be long and it may be useful to use only the shortest of these constraints, as they are most likely to help speed up plan search. In the experiments reported later, we used only constraints with 6 literals or less. We tried the full set of process axioms, but the high number of long disjunctions led to poor performance.

## 6 1-Linearization Semantics

Dimopoulos et al. [4] adapted the idea of satisfiability planning to answer set programming and presented an interesting idea. The requirement that parallel operators are executable in any order can be relaxed, only requiring that *one* ordering is executable. They called this idea *post-serializability* and showed how to transform operators for blocks world problems to make them post-serializable. The resulting nonmonotonic logic programs were shown to be more efficient due to a shorter parallel plan length. Rintanen [8] implemented this idea in a constraint-based planner and Cayrol et al. [3] in the GraphPlan framework.

We will present a semantics and general-purpose domain-independent translations of this more relaxed semantics into the propositional logic. Our approach does not require transforming the problem. Instead, we synthesize constraints that guarantee that the operators applied simultaneously can be ordered to an executable plan.

**Definition 4 (1-linearization plans).** For a set of operators  $O$  and an initial state  $I$ , a 1-linearization plan is a sequence  $T = S_1, \dots, S_l$  of sets of operators such that there is a sequence of states  $s_0, \dots, s_l$  (the execution of  $T$ ) such that

1.  $s_0 = I$ ,
2.  $s_{i-1} \models p$  for all  $i \in \{1, \dots, l\}$  and  $\langle p, e, c \rangle \in S_i$ ,
3. the set  $\bigcup_{\langle p, e, c \rangle \in S_i} (e \cup \bigcup \{d \mid f \triangleright d \in c, s_{i-1} \models f\})$  is consistent for every  $i \in \{1, \dots, l\}$ ,
4.  $s_i = \text{app}_{S_i}(s_{i-1})$  for all  $i \in \{1, \dots, l\}$ , and

5. for every  $i \in \{1, \dots, l\}$  there is a total ordering  $o_1 < o_2 < \dots < o_n$  of  $S_i$  such that for all operators  $o_j = \langle p_j, e_j, c_j \rangle \in S_i$
- (a)  $\text{app}_{o_1; o_2; \dots; o_{j-1}}(s_{i-1}) \models p_j$ , and
  - (b) for all  $f \triangleright d \in c_j$ ,  $s_{i-1} \models f$  if and only if  $\text{app}_{o_1; o_2; \dots; o_{j-1}}(s_{i-1}) \models f$ .

The difference to step semantics is that we have replaced the non-interference condition with a weaker condition. From an implementations point of view, the main difficulty here is finding appropriate total orderings  $<$ .

**Theorem 2.** (i) Each step plan is a 1-linearization plan and (ii) for every 1-linearization plan  $T$  there is a step plan whose execution leads to the same final state as that of  $T$ .

*Proof.* (Sketch) (i) Consider a step plan  $T = S_1, \dots, S_l$ . By Theorem 1 for every  $i \in \{1, \dots, l\}$  any total ordering of the operators in  $S_i$  can be used to satisfy Condition 5 in Definition 4. Hence,  $T$  is a 1-linearization plan. (ii) For a 1-linearization plan  $T = S_1, \dots, S_l$ , a step plan whose execution leads to the same final state as that of  $T$  can be obtained as follows:  $\{o_1^1\}, \dots, \{o_{n_1}^1\}, \dots, \{o_1^l\}, \dots, \{o_{n_l}^l\}$  where for every  $i \in \{1, \dots, l\}$ , the sequence  $\{o_1^i\}, \dots, \{o_{n_i}^i\}$  is a total ordering of  $S_i$  given by Condition 5 of Definition 4.

## 6.1 Encoding in the Propositional Logic

Given the precondition and effect axioms (1), (2) and (3), we have to guarantee that there is a total ordering of the operators so that no operator application disables the operators that will be applied later, and no operator application changes the set of active (conditional) effects of later operators. The encodings we give are stricter than our formal definition of 1-linearization semantics and do not always allow all the parallelism that is possible. Next we define the notion of *disabling graphs* in order to provide compact and effective encodings of 1-linearization semantics in the propositional logic.

The motivation for using disabling graphs is the following. Define a *circularly disabled set* as a set of operators that is applicable in some state without the effects contradicting each other and that cannot be totally ordered into a sequential plan so that no operator disables or changes the active effects of a later operator. Now any set-inclusion minimal circularly disabled set is a subset of an SCC of the disabling graph. We may allow the simultaneous application of a set of operators from the same SCC if the subgraph of the disabling graph induced by those operators does not contain a cycle.<sup>3</sup>

**Definition 5 (Disabling graph).** A graph  $\langle O, E \rangle$  is a disabling graph of a set of operators  $O$  where  $E \subseteq O \times O$  is the set of directed edges so that  $\langle o_1, o_2 \rangle \in E$  if for  $o_1 = \langle p_1, e_1, c_1 \rangle$  and  $o_2 = \langle p_2, e_2, c_2 \rangle$  there is a state  $s$ <sup>4</sup> such that

1.  $s \models p_1 \wedge p_2$ , and

<sup>3</sup> In step semantics simultaneous application is allowed if the subgraph does not have *any* edges.

<sup>4</sup> Clearly, this could be restricted to states that are reachable from the initial state, but testing reachability is PSPACE-hard. Instead, one can use some subclass of invariants computable in polynomial time to ignore some of the unreachable states, like we have done in our implementation of disabling graphs.

2.  $F_1 \cup F_2$  is consistent where  $F_1 = e_1 \cup \bigcup\{d \mid f \triangleright d \in c_1, s \models f\}$  and  $F_2 = e_2 \cup \bigcup\{d \mid f \triangleright d \in c_2, s \models f\}$ , and
3. applying  $o_1$  may make  $o_2$  inapplicable or may change the active effects of  $o_2$ :
  - (a)  $\text{app}_{o_1}(s) \not\models p_2$ , or
  - (b) there is  $f \triangleright d \in c_2$  such that either  $s \models f$  and  $\text{app}_{o_1}(s) \not\models f$ , or  $s \not\models f$  and  $\text{app}_{o_1}(s) \models f$ .

For a given set of operators there are typically several disabling graphs because the graph obtained by adding an edge to a disabling graph is also a disabling graph. In the experiments in Section 7 we use disabling graphs that are not necessarily minimal but can be computed in polynomial time. For STRIPS operators they are minimal.

Our disabling graphs are related to the definition of preconditions-effects graphs of Dimopoulos et al. [4], but they often have many less edges and much smaller SCCs. For example, in the well-known logistics problems all the strong components have 1 or  $n + 1$  operators, where  $n$  is the number of airplanes<sup>5</sup>. This means that encoding the constraints that guarantee that simultaneous operators indeed can be linearized will be rather efficient, as only cycles of rather small length have to be considered. Notice that operators in different strong components cannot be part of the same cycle; therefore constraints on their simultaneous application are not needed. Hence when every strong component has cardinality 1, no constraints whatsoever are needed.

Next we discuss two ways of synthesizing constraints that guarantee that simultaneous operators can be ordered to a valid totally ordered plan.

## 6.2 General $O(n^3)$ Encoding

We can exactly test that the intersection of one SCC and a set of simultaneous operators do not form a cycle. The next encoding allows the maximum parallelism with respect to a given disabling graph, but it is expensive in terms of formula size.

Let  $o^i$  and  $o^{i'}$  belong to the same SCC of the disabling graph and let there be an edge from  $o^i$  to  $o^{i'}$ . We use auxiliary propositions  $c^{i,j}$  for all operators with indices  $i$  and  $j$ , indicating that there is a set of applied operators  $o^i, o^1, o^2, \dots, o^n, o^j$  such that every operator disables or changes the effects of its immediate successor in the sequence. Then we have the formulae  $(o_t^i \wedge c_t^{i',j}) \rightarrow c_t^{i,j}$  for all  $i, i'$  and  $j$  such that  $i \neq i' \neq j \neq i$ . Further we have formulae  $\neg(o_t^i \wedge c_t^{i',i})$  for preventing the completion of a cycle.

The size of the encoding is cubic and the number of new propositional variables is quadratic in the number of operators in an SCC. Some problems have SCCs of dozens or hundreds of operators, and this  $O(n^3)$  means that there are thousands or millions of formulae, which often makes this encoding impractical.

## 6.3 Fixed Ordering

The simplest and possibly the most effective encoding does not allow all the parallelism that allowed by the preceding encoding, but it leads to small formulae. With this

<sup>5</sup> The refinement to disabling graphs involving invariants in the preceding footnote makes all SCCs for Logistics singleton sets.

encoding the number of constraints on parallel application *is smaller* than with the less permissive step semantics, as the set of constraints on parallelism is a subset of the constraints for step semantics. One therefore receives two benefits simultaneously: possibly much shorter parallel plans and formulae with a smaller size / time points ratio.

The idea is to impose beforehand an (arbitrary) ordering on the operators  $o^1, \dots, o^n$  in an SCC and to disallow parallel application of two operators  $o^i$  and  $o^j$  such that  $o^i$  may disable or change the effects of  $o^j$  only if  $i < j$ . Hence, in comparison to step semantics, part of the parallelism axioms on operators within one SCC are left out. In comparison to step semantics, the total reduction in the number of constraints can be significant because none of the inter-SCC parallelism constraints are needed.

This is the encoding we have very successfully applied to a wide range of planning problems, as discussed in the next section. Selecting the ordering carefully may increase parallelism. In our experiments we order the operators in the order they happen to come out of our PDDL front-end. Better orderings could be produced by heuristic methods.

## 7 Experiments

We evaluate the different semantics on a number of benchmarks from the AIPS planning competitions. In addition to the Logistics, Depots and Satellite benchmarks reported here, we also test Driver, Zeno, Freecell, Schedule and Mystery, but do not report runtimes because of lack of space. On Freecell and Schedule 1-linearization does not decrease plan length and runtimes are comparable to step semantics. Process semantics fares worse than step semantics on Schedule. Mystery is trivial for 1-linearization semantics. Runtime differences with Driver and Zeno are like with Logistics.

In Tables 1 and 2 we present the name of the problem instance and the runtimes for the formulae corresponding to the highest number(s) of time points without a plan (truth value F) and the first satisfiable formula corresponding to a plan (truth value T). Runtimes for 1-linearization semantics are reported on their own lines because its shortest plan lengths differ from the other semantics.

For the experiments we use a 3.6 GHz Intel Xeon processor with 512 KB internal cache and the Siege SAT solver version 3 by Ryan of the Simon Fraser University. Because Siege uses randomization, the runtimes for a given formula vary across executions. We run Siege 40 times on each formula and report the average. When only some of the runs finish within a time limit of 3 to 4 minutes (we terminate every 60 seconds those processes that have consumed over 180 seconds of CPU) we report the average time  $t$  of the finished runs as  $> n$ . This roughly means that the average runtime on Siege exceeds  $n$  seconds. A dash — indicates that none of the runs finished.

The best runtimes are usually obtained with the 1-linearization semantics. It is often one or two orders of magnitude faster. This usually goes back to the shorter plan length: formulae are smaller and easier to evaluate.

Contrary to our expectations, process semantics usually does not provide an advantage over step semantics although there are often far fewer potential plans to consider. When showing the inexistence of plans of certain length, the additional constraints could provide a big advantage, similarly to symmetry-breaking constraints. In a few

instance	len	val	1-lin	step	proc
satell-15	4	F	8.86		
satell-15	5	T	1.65		
satell-15	7	F		24.15	21.91
satell-15	8	T		3.61	3.50
satell-16	3	F	2.27		
satell-16	4	T	3.91		
satell-16	5	F		9.17	8.24
satell-16	6	?		-	-
satell-16	7	T		6.57	6.85
satell-17	3	F	0.22		
satell-17	4	T	2.48		
satell-17	5	F		1.12	1.31
satell-17	6	T		1.86	1.96
satell-18	4	F	0.06		
satell-18	5	T	0.23		
satell-18	7	F		0.24	0.27
satell-18	8	T		0.48	0.57
satell-19	6	F	46.26		
satell-19	7	T	25.78		
satell-19	10	F		> 225.50	-
satell-19	11	?		-	-
satell-19	12	T		> 170.69	-

instance	len	val	1-lin	step	proc
log-23-0	8	F	0.56		
log-23-0	9	T	2.93		
log-23-0	14	F		37.69	27.86
log-23-0	15	?		-	-
log-23-0	16	T		> 139.10	> 132.64
log-23-1	8	F	1.70		
log-23-1	9	T	0.57		
log-23-1	14	F		48.34	44.12
log-23-1	15	T		> 66.07	> 76.50
log-24-0	8	F	0.40		
log-24-0	9	T	3.33		
log-24-0	14	F		35.93	13.77
log-24-0	15	?		-	-
log-24-0	16	T		> 108.92	> 99.44
log-24-1	9	F	9.66		
log-24-1	10	T	2.61		
log-24-1	15	F		> 101.29	> 112.07
log-24-1	16	?		-	-
log-24-1	17	T		> 131.52	> 119.01

**Table 1.** Runtimes of Satellite and Logistics problems in seconds

cases, like the last or second to last unsatisfiable formula for log-24-0, process constraints do halve the runtimes. The reason for the ineffectiveness of process semantics may lie in these benchmarks: many operators may prevent earlier application of an operator, and this results in long clauses that figure only very late in the search.

## 8 Related Work

The BLACKBOX planner of Kautz and Selman [7] is the best-known planner that implements the satisfiability planning paradigm. Its GraphPlan-based encoding is similar to our step semantics encoding, but less compact for example because of its use of GraphPlan’s NO-OPs. Comparison of sizes and evaluation times (Sieve V3) between our step semantics encoding and BLACKBOX’s encoding is given in Table 3. Corresponding 1-linearization encodings are, per time point, between 94 per cent (depot-11) and 69 per cent (logistics-23-1) of the step encoding sizes.

The above data suggest that the efficiency of BLACKBOX encodings is either roughly comparable or lower than our basic encoding for step semantics, and hence sometimes much lower than our 1-linearization encoding.

instance	len	val	1-lin	step	proc
depot-10	7	F	0.01		
depot-10	8	T	0.02		
depot-10	9	F		0.28	0.30
depot-10	10	T		0.29	0.34
depot-11	13	F	0.04		
depot-11	14	T	0.44		
depot-11	17	F		69.56	70.29
depot-11	18	?		-	-
depot-11	19	?		-	-
depot-11	20	T		> 154.43	> 157.28
depot-12	19	F	0.24		
depot-12	20	T	> 143.73		
depot-12	21	F		142.12	> 140.75
depot-12	22	?		-	-
depot-13	7	F	0.01		
depot-13	8	T	0.01		
depot-13	8	F		0.01	0.01
depot-13	9	T		0.04	0.05
depot-14	9	F	0.05		
depot-14	10	T	0.11		
depot-14	11	F		1.25	1.28
depot-14	12	T		2.97	2.89

**Table 2.** Runtimes of Depot problems in seconds

## 9 Conclusions

We have given translations of semantics for parallel planning into SAT and shown that one of them, for 1-linearization semantics, is very efficient, often being one or two orders of magnitude faster than previous encodings. This semantics is superior because with our encoding the number of time steps and parallelism constraints is small. Interestingly, the process semantics, a refinement of the standard step semantics that imposes a further condition on plans, usually did not improve planning efficiency in our tests.

The 1-linearization encoding combined with novel strategies for finding satisfiable formulae that correspond to plans [9] sometimes lead to a substantial improvement in efficiency for satisfiability planning.

## Acknowledgments

We thank Lawrence Ryan for his assistance with Siege. Part of the authors gratefully acknowledge the financial support of the Academy of Finland (project 53695 and grant for research work abroad), FET project ADVANCE contract No IST-1999-29082, and EPSRC grant 93346/01.

instance	len	val	size in MB		runtime in secs	
			step	BB	step	BB
depot-11-8765	17	F	6.9	57.0	69.56	331.60
depot-11-8765	20	T	8.3	85.6	207.88	> 1200
logistics-23-1	14	F	4.8	20.9	48.34	71.97
logistics-23-1	15	T	5.2	25.6	99.31	115.16
driver-4-4-8	10	F	5.5	35.0	1.26	0.53
driver-4-4-8	11	T	6.1	53.2	5.56	21.00

**Table 3.** Sizes and runtimes of our step encoding and BLACKBOX's GraphPlan-based encoding

## References

1. Best, E., Devillers, R.: Sequential and concurrent behavior in Petri net theory. *Theoretical Computer Science* **55** (1987) 87–136
2. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. *Artificial Intelligence* **90** (1997) 281–300
3. Cayrol, M., Régnier, P., Vidal, V.: Least commitment in Graphplan. *Artificial Intelligence* **130** (2001) 85–118
4. Dimopoulos, Y., Nebel, B., Koehler, J.: Encoding planning problems in nonmonotonic logic programs. In Steel, S., Alami, R., eds.: *Recent Advances in AI Planning. Fourth European Conference on Planning (ECP'97)*. Number 1348 in *Lecture Notes in Computer Science*, Springer-Verlag (1997) 169–181
5. Heljanko, K.: Bounded reachability checking with process semantics. In: *Proceedings of the 12th International Conference on Concurrency Theory (Concur'2001)*. Volume 2154 of *Lecture Notes in Computer Science*, Springer-Verlag (2001) 218–232
6. Kautz, H., Selman, B.: Pushing the envelope: planning, propositional logic, and stochastic search. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, Menlo Park, California, AAAI Press (1996) 1194–1201
7. Kautz, H., Selman, B.: Unifying SAT-based and graph-based planning. In Dean, T., ed.: *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers (1999) 318–325
8. Rintanen, J.: A planning algorithm not based on directional search. In Cohn, A.G., Schubert, L.K., Shapiro, S.C., eds.: *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, Morgan Kaufmann Publishers (1998) 617–624
9. Rintanen, J.: Evaluation strategies for planning as satisfiability. In Saitta, L., ed.: *Proceedings of the 16th European Conference on Artificial Intelligence*, IOS Press (2004) to appear.