Planning as Satisfiability with Relaxed ∃-Step Plans

Martin Wehrle[1]* and Jussi Rintanen[2]**

[1] Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany
[2] National ICT Australia Ltd and Australian National University, Canberra, Australia

Abstract. Planning as satisfiability is a powerful approach to solving domain independent planning problems. In this paper, we consider a relaxed semantics for plans with parallel operator application based on ∃-step semantics. Operators can be applied in parallel if there is at least one ordering in which they can be sequentially executed. Under certain conditions, we allow them to be executed simultaneously in a state s even if not all of them are applicable in s. In this case, we guarantee that they are enabled by other operators that are applied at the same time point. We formalize the semantics of parallel plans in this setting, and propose an effective translation for STRIPS problems into the propositional logic. We finally show that this relaxed semantics yields an approach to classical planning that is sometimes much more efficient than the existing SAT-based planners.

1 Introduction

Planning as satisfiability is a leading approach to solving domain independent planning problems. An important factor in its efficiency is the notion of parallel plans. In the definition of Kautz and Selman [1], operators can be applied in parallel as long as they are mutually non-interfering. This guarantees that any total ordering on them is a valid execution and leads in all cases to the same state. Therefore, non-interfering operators need not be considered in all possible orderings, and respective propositional encodings get more compact as many intermediate states do not have to be represented explicitly.

Based on an idea of Dimopoulos et al. [2], Rintanen et al. [3] present a relaxed semantics which they call ∃-step semantics. In contrast to the standard ∀-step semantics, operators are allowed to be executed in parallel if they are sequentially executable in at least *one* ordering. This is a less restrictive condition than non-interference. As a consequence, plans usually get shorter and can be found more efficiently. Rintanen et al. present propositional encodings for general ADL problems that lead to a subclass of general ∃-step plans with the additional constraint that simultaneously applied operators in a state s have to be already applicable in s.

* Supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `http://www.avacs.org/` for more information.

** Supported by NICTA in the framework of the DPOLP project. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian National Research Council.

In this paper, we show that this concept can be further generalized for STRIPS problems. We present an encoding that allows operators to be executed in parallel in a state s if they are all either already applicable in s or enabled by some other operators that are applied at the same time point. This leads to a wider class of $\exists$-step plans in which more operators are simultaneously applicable. We call this semantics of parallelism the *relaxed $\exists$-step semantics*. In many planning domains, this leads to efficiency gains because of shorter parallel plan lengths. Furthermore, the formulae get more strongly constrained which can make them easier to solve. One main innovation in this context is the notion of the *disabling-enabling-graph* which is an extension of the disabling graph presented by Rintanen et al. [3].

The structure of this paper is as follows. Section 2 presents basic notations and the formal definition of the new semantics. In Section 3, we present a translation into the propositional logic. Section 4 shows its performance in comparison to the so far most efficient $\exists$-step encoding and to the encoding used by SATPLAN06 [4], one of the winners of the International Planning Competition 2006 for optimal deterministic planning. Finally, we discuss related work and conclude the paper.

2 Background

2.1 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 on P, i.e. an assignment $s : P \to \{T, F\}$. A literal is a formula of the form a or $\neg a$ for all $a \in P$. For a literal l we define $\overline{l}$ by $\overline{a} = \neg a$ and $\overline{\neg a} = a$. The set of all literals is denoted by $\mathcal{L} = P \cup \{\neg a \,|\, a \in P\}$. We use *operators* for expressing how the state of the world can be changed. As this work restricts to STRIPS operators, we use the term operator for STRIPS operator. An *operator* on a set of state variables P is a tuple $o = \langle p, e\rangle$ where $p \subset \mathcal{L}$ is a set of literals, the *precondition*, and $e \subset \mathcal{L}$ is a set of literals, the *effect*. The operator is *applicable* in a state s if $s \models p$ (where we identify a set of literals with their conjunction). In this case, we define $\text{app}_o(s) = s'$ as the unique state that is obtained from s by making the effect literals of o true and retaining the truth-values of the state variables not occurring in the effect. For sequences of operators $o_1, \ldots, o_n$ we define $\text{app}_{o_1;\ldots;o_n}(s)$ as $\text{app}_{o_n}(\ldots \text{app}_{o_2}(\text{app}_{o_1}(s))\ldots)$. We say that $\text{app}_{\langle p_1,e_1\rangle;\ldots;\langle p_n,e_n\rangle}(s)$ is *defined* if and only if $\text{app}_{\langle p_1,e_1\rangle;\ldots;\langle p_i,e_i\rangle}(s) \models p_{i+1}$ for $i = 0, \ldots, n-1$.

Let $\pi = \langle P, I, O, G\rangle$ be a *planning 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 π is a sequence $\sigma = o_1; \ldots; o_n$ of operators from O such that $\text{app}_\sigma(I) \models G$, i.e. applying the operators in the given order starting in the initial state is defined (the 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 are simultaneously applied. In contrast to the $\exists$-step semantics given by Rintanen et al., operators that are applied simultaneously in a state s do not necessarily have to be applicable in s.

Example 1. Let $\pi = \langle P, I, O, G\rangle$ be a planning instance with state variables $P = \{A_i \mid i \in \{1,\ldots,n\}\}$, initial state I such that $I \models \bigwedge_{i=1}^{n} \neg A_i$, goal state G such that $G = \bigwedge_{i=1}^{n} A_i$ and operators $o_1 = \langle \top, \{A_1\}\rangle$ and $o_i = \langle \{A_{i-1}\}, \{A_i\}\rangle$ for $i = 2,\ldots,n$. As there is no state in which more than one precondition is satisfied, the shortest $\exists$-step plan consists of n steps, whereas the more relaxed $\exists$-step plans can reach the goal within only one step, as $o_1,\ldots,o_n$ are applicable in parallel. This is because we allow simultaneously applied operators to be enabled by other operators, and o_{i-1} enables o_i for all $i \in \{2,\ldots,n\}$.

2.2 Planning as Satisfiability

Planning can be performed by propositional satisfiability testing as follows. Produce formulae $\Phi^0, \Phi^1, \Phi^2, \ldots$ such that Φ^i is satisfiable if there is a plan of length i. The formulae are tested for satisfiability in 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. Next we describe the general structure of such a formula Φ^i.

The state variables in a problem instance are $P = \{a_1,\ldots,a_n\}$ and the operators are $O = \{o_1,\ldots,o_m\}$. 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 Φ we denote the formula with all propositional variables superscribed with the index at time point t by Φ^t. The set of propositions where all variables are superscribed with index t is denoted by P^t.

For a planning instance π, a formula is generated to answer the following question. Is there an execution of a sequence of sets of operators taking k time points that reaches a state satisfying G from the initial state I? The structure of such a formula Φ^k is

$$I^0 \wedge R(P^0, P^1) \wedge \cdots \wedge R(P^{k-1}, P^k) \wedge G^k, \tag{1}$$

where I^0 and G^k are formulae describing the initial and goal state (the propositions are marked with 0 and k, respectively), and $R(P^t, P^{t+1})$ describes the transition relation leading from the state at time point t to the state at time point $t+1$ for all $t \in \{0,\ldots,k-1\}$. We will introduce a corresponding propositional encoding of that relation according to our semantics.

2.3 Relaxed $\exists$-Step Semantics

Rintanen et al. [3] adapted the linearization idea of Dimopoulos et al. to planning as satisfiability and showed that corresponding encodings in the propositional logic yield efficiency gains in contrast to the standard definition. As they considered general operators (i.e. operators with arbitrary formulae as preconditions and conditional effects), they additionally restricted their semantics such that the operators' preconditions have to be already satisfied in s. In the following, we present a semantics that relaxes this requirement. In contrast to the approach of Rintanen et al., we allow operators to be executed in parallel in a state s even if not all of them are already applicable in s. Such operators have to be enabled by other operators applied at the same time point.

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

1. *for all $i \in \{1, \ldots, l\}$ there is a total ordering $o_1 < \cdots < o_n$ on S_i such that $app_{o_1;\ldots;o_{j-1}}(s_{i-1}) \models p_j$ for all $o_j = \langle p_j, e_j \rangle \in S_i$, $s_i = app_{o_1;\ldots;o_n}(s_{i-1})$, and*
2. *the set $\bigcup_{\langle p,e\rangle \in S_i} e$ is consistent for all $i \in \{1, \ldots, l\}$.*

In the next section, we synthesize constraints that lead to relaxed $\exists$-step plans.

3 Propositional Encoding

For every $o = \langle p, e\rangle \in O$ there are the following axioms. First, if o is applied at a time point t, then its effects are true at the next time point. This is expressed by

$$o^t \rightarrow e^{t+1}, \tag{2}$$

where we identify the set e of literals with their conjunction. 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 from false to true, and another from true to false. The formulae are analogous, and here we only give the one for change from true to false:

$$(a^t \wedge \neg a^{t+1}) \rightarrow \bigvee \{o^t | o = \langle p, e\rangle \in O, \neg a \in e\}, \tag{3}$$

where the empty disjunction is the constant $\bot$. Finally, the preconditions of applied operators must be true, and furthermore we need axioms for restricting the parallel application of operators. These *precondition* and *parallelism* axioms are described next. We define the notion of *disabling-enabling-graphs* in order to provide compact encodings in the propositional logic. It generalizes the concept of disabling graphs introduced by Rintanen et al. for the $\exists$-step semantics.

We first state when two operators *conflict* in their preconditions, i.e. we formulate a necessary condition for being applied simultaneously according to our semantics. Informally speaking, two operators o and o' conflict if there is no reachable state s such that they can be executed sequentially in s. Therefore, they cannot be applied simultaneously in our relaxed $\exists$-step semantics.

Definition 2 (Conflict). *Let $\pi = \langle P, I, O, G\rangle$ be a planning instance and $o = \langle p, e\rangle$, $o' = \langle p', e'\rangle \in O$. If there are literals $m_1 \in p$ and $m_2 \in p'$, and*

1. *there is no reachable state[3] s such that $s \models (m_1 \wedge m_2)$, and*
2. *there is no operator $o'' = \langle p'', e''\rangle$ such that*
   (a) $m_2 \in e''$,

[3] As testing exactly whether a given state is reachable in a planning instance is already PSPACE-complete (and therefore as hard as planning itself), we approximate this test by using a subclass of 2-literal-invariants as produced by the algorithm by Rintanen [5] which are computable in polynomial time.

(b) there is no literal m such that $m \in e''$ and $\overline{m} \in p'$, and
(c) o, o'' and o'', o' have consistent effects,

then o and o' conflict in the ordering $o < o'$. *They* conflict *if they conflict in both orderings $o < o'$ and $o' < o$.*

Example 2. Let $\pi = \langle P, I, O, G \rangle$ be a planning instance with $P = \{A, B\}$, initial state $I \models A \wedge \neg B$, and operators $O = \{o_1, o_2\}$, where $o_1 = \langle \{A\}, \{\neg A, B\} \rangle$ and $o_2 = \langle \{B\}, \{A, \neg B\} \rangle$. Then o_1 and o_2 conflict as there is no reachable state s such that $s \models A \wedge B$, and there is no operator $o'' = \langle p'', e'' \rangle$ to resolve this conflict.

The motivation for using disabling-enabling-graphs is the following. The goal is to identify all sets of operators that "could be applied" simultaneously in our semantics (i.e. the operators do not conflict and have consistent effects), but there is a reachable state such that there is no ordering in which they can be sequentially applied. Such sets are subsets of a *strongly connected component (SCC)* of the disabling-enabling-graph. Strongly connected components can be identified in linear time [6].

Definition 3 (Disabling-Enabling-Graph). *Let $\pi = \langle P, I, O, G \rangle$ be a planning instance. A* disabling-enabling-graph *for π is a directed graph $G = \langle O, E \rangle$ with the following properties: For all $o = \langle p, e \rangle$ and $o' = \langle p', e' \rangle \in O$ there is an edge $(o, o') \in E$, if*

1. *there is a literal $m \in \mathcal{L}$ such that $m \in e$ and $m \in p'$ (*enabling*), or $m \in e'$ and $\overline{m} \in p$ (*disabling*),*
2. *o and o' do not conflict, and*
3. *there is a reachable state s such that $s \models e \wedge e'$ (parallel execution of o and o' leads to a reachable state).*

Example 3. Let $\pi = \langle P, I, O, G \rangle$ be a planning instance with $P = \{A, B, C\}$, initial state $I \models A \wedge B \wedge C$, and operators $O = \{o_1, o_2, o_3\}$, where $o_1 = \langle \{A\}, \{\neg B\} \rangle$, $o_2 = \langle \{B\}, \{\neg C\} \rangle$, and $o_3 = \langle \{C\}, \{\neg A\} \rangle$. In this case, the disabling-enabling-graph only contains disabling edges. There is a strongly connected component indicating that there is no ordering $<$ such that the operators can be executed according to $<$.

As the reachability of states is approximated, the disabling-enabling-graph that we use in our implementation is not necessarily minimal, but it can be computed in polynomial time in the size of the problem instance. This does not affect the correctness of our encoding as *less* operators are simultaneously applicable if the graph is not minimal. The number of edges typically increases in comparison to the disabling graph because enabling edges have to be considered as well. Moreover, the condition to be applied at some time point is relaxed, and we have to consider more operators that "could be applied" in parallel.

We impose a fixed ordering on all SCCs beforehand and allow operator execution only in that ordering. The resulting encoding is stricter than our formal definition of relaxed $\exists$-step semantics and does not always allow all the parallelism that is possible, but it leads to small formulae. For an SCC $S = \langle V, E \rangle$ with $V = \{o_1, \ldots, o_n\}$, fixed ordering $o_1 < \cdots < o_n$ and $i \in \{1, \ldots, n\}$, we denote the set of operators $\{o_{i+1}, \ldots, o_n\}$ that occur after o_i in this ordering by $Succ(S, o_i)$.

In the following, we describe the precondition axioms. In order to get a compact representation, we define for all operators and literals a corresponding set of *enabling operators*.

Definition 4 (Enabling Operators). *Let $o = \langle p, e\rangle$ be an operator, m a literal. The set $enOps(o; m)$ of* enabling operators *for o and m is defined such that $o' = \langle p', e'\rangle \in enOps(o; m)$ if and only if $o' \neq o$, $m \in e'$, o and o' have consistent effects and do not conflict in the ordering $o' < o$.*

Informally speaking, $enOps(o; m)$ contains operators that can be applied in parallel with o and have the literal m in their effect. In order to be applicable in a state s, we require for an operator $o = \langle p, e\rangle$ that for all literals $m \in p$ either $s \models m$ or there is an enabling operator $o' = \langle p', e'\rangle \in enOps(o; m)$ that is applied simultaneously. This is stated in the formula

$$o^t \rightarrow \bigwedge_{m \in p} (\overline{m}^t \rightarrow \bigvee (enOps^t(o; m) \setminus Succ^t(S, o))), \tag{4}$$

where S is the (unique) SCC in which o occurs and all propositions are labeled with time point t, and $enOps(o; m)^t$ is the set of enabling operators that are labeled with t. Operators occurring after o are ruled out because this allows us to use weaker parallelism axioms which are described next.

We guarantee that operators that are applied in parallel can be executed sequentially in at least one way. A sufficient but too strong constraint is the acyclicity of the corresponding disabling-enabling-graph. Instead, we only require that there is no cycle of operators that disable one another. This is sufficient because of our definition of the precondition axioms (4). Therefore, for SCC S and operator o in S, we require that if o is applied, then all operators in S that are disabled by o and occur after o in the fixed ordering are not applied at the same time point.

Let $S = \langle V, E\rangle$ be an SCC, $V = \{o_1, \ldots, o_n\}$, m a literal, and $o_1 < \cdots < o_n$ a fixed ordering. Let $E = \{o = \langle p, e\rangle \in V \mid \overline{m} \in e\}$ be the set of operators that falsify m and $R = \{o = \langle p, e\rangle \in V \mid m \in p\}$ the set of operators that require m to remain true. Similarly to Rintanen et al. [3], we introduce the chain-formula

$$\begin{aligned}
&\bigwedge \big\{o_i \rightarrow a_m^j \mid i < j, o_i \in E, o_j \in R, \{o_{i+1}, \ldots, o_{j-1}\} \cap R = \emptyset\big\} \\
&\cup \big\{a_m^i \rightarrow a_m^j \mid i < j, \{o_i, o_j\} \subseteq R, \{o_{i+1}, \ldots, o_{j-1}\} \cap R = \emptyset\big\} \\
&\cup \big\{a_m^j \rightarrow \neg o_j \mid o_j \in R\big\},
\end{aligned}$$

where the auxiliary variable a_m^j is true if there is an operator $o_i \in E_m$ with $o_i < o_j$ that is applied and falsifies m. The number of auxiliary variables is linear in the number of operators. The key idea behind this chain-formula is to ensure that if an operator o is applied that falsifies a literal m, then all operators occurring *after* o in the fixed ordering that require m to be true cannot be applied.

Example 4. Consider again the planning instance $\langle P, I, O, G\rangle$ with $P = \{A, B, C\}$, initial state $I \models A \wedge B \wedge C$, and operators $O = \{o_1, o_2, o_3\}$, where $o_1 = \langle\{A\}, \{\neg B\}\rangle$, $o_2 = \langle\{B\}, \{\neg C\}\rangle$, and $o_3 = \langle\{C\}, \{\neg A\}\rangle$. As we have seen in Example 3, these

operators are not serializable. For a fixed ordering $o_1 < o_2 < o_3$, the chain-encoding leads to the following parallelism axioms. For variable A, the set of operators that falsify A is $E = \{o_3\}$, the set of operators that require A to remain true is $R = \{o_1\}$. As $o_1 < o_3$, we do not need any parallelism axioms for A. For B, we get $E = \{o_1\}$ and $R = \{o_2\}$. Therefore, we have to ensure that if o_1 is applied, then o_2 cannot be applied at the same time point. Hence, we get the parallelism axioms $o_1 \rightarrow a_B^2$ and $a_B^2 \rightarrow \neg o_2$. For C, we get the axioms $o_2 \rightarrow a_C^3$ and $a_C^3 \rightarrow \neg o_3$ in a similar way.

In our experiments, we have kept the operators' ordering as they have come out of our PDDL front end.

The conjunction of the effect, frame, precondition and parallelism axioms for all operators describes a transition relation $R(P^t, P^{t+1})$ leading from the state at time point t to the successor state. In a state s, if $s \models R(P^t, P^{t+1})$, there is a total ordering $o_1 < \cdots < o_n$ on the parallel operators $o_1, \ldots, o_n$ such that $\text{app}_{o_1;\ldots;o_n}(s)$ is defined. Let *Var(t)* be the set of all variables occurring in $R(P^t, P^{t+1})$.

Theorem 1. *Let $\pi = \langle P, I, O, G\rangle$ be a planning instance, and s a state reachable from I. Let ν be a valuation of Var(t) for some t such that $\nu(a^t) = s(a^t)$ for all $a^t \in P^t$ and $\nu \models R(P^t, P^{t+1})$. Let $\{o \mid \nu \models o^t\} = \{o_1, \ldots, o_n\}$ be the set of operators that is applied at time point t. Then there is a total ordering $o_1 < \cdots < o_n$ such that $app_{o_1;\ldots;o_n}(s)$ is defined.*

Proof. (sketch) There is an ordering $o_1 < \cdots < o_n$ such that for all $o, o' \in \{o_1, \ldots, o_n\}$: If o, o' are contained in the same SCC $G_i = \langle V_i, E_i\rangle$, then $o < o'$ iff $o <_i o'$ (where $<_i$ is the ordering imposed on V_i). If o, o' are contained in different SCCs, then $o < o'$ if there is an edge from o to o'. As o, o' do not conflict (the reader may verify this) and have consistent effects for all $o, o' \in \{o_1, \ldots, o_n\}$, it can be shown by induction that $app_{o_1;\ldots;o_n}(s)$ is defined: for all $o_i = \langle p_i, e_i\rangle \in \{o_1, \ldots, o_n\}$ and for all $m \in p_i$ the following holds. There is no $o_j = \langle p_j, e_j\rangle \in \{o_1, \ldots, o_{i-1}\}$ with $\overline{m} \in e_j$. If $s \not\models m$, then there is $o_j = \langle p_j, e_j\rangle \in \{o_1, \ldots, o_{i-1}\}$ with $m \in e_j$. Therefore $app_{o_1;\ldots;o_{i-1}}(s) \models p_i$.

3.1 Optimizations

In order to speed up planning, we additionally use invariants, which are formulae that are true in all states reachable from the initial state. A restricted class of invariants can be identified in polynomial time. In our experiments, we use formulae $l_1^t \vee l_2^t$ for invariants $l_1 \vee l_2$ as produced by the algorithm by Rintanen [5].

For an operator $o = \langle p, e\rangle$ and a literal m, invariants can also be used to further reduce the number of enabling operators in $enOps(o; m)$. Consider $o' = \langle p', e'\rangle \in enOps(o; m)$. If there is an invariant $l_1 \vee l_2$ such that $\overline{l_1} \in e'$ and $\overline{l_2} \in p$, then o' cannot be used to enable o at the same time point. Hence, such operators o' are ruled out in our implementation of $enOps(o; m)$. Note that this does not affect correctness.

Furthermore, it is useful to keep the SCCs as small as possible because of the fixed ordering that we impose on their operators (the more operators we restrict to be executed in a fixed ordering, the more parallelism will possibly be lost). In order to tackle the problem of increasing SCCs, we add constraints $\neg o_1^t \vee \neg o_2^t$ for operators o_1 and o_2 that disable each other (i.e. o_1 disables o_2, and vice versa), because such pairs cannot

be executed in parallel anyway according to our semantics. The quadratical worst case size is almost never an issue in practice, but the size of the SCCs decreases because we need no edges between o_1 and o_2 in the disabling-enabling-graph.

4 Experiments

We evaluated our new relaxed $\exists$-step encoding in comparison to the so far most efficient $\exists$-step encoding [3] and to the encoding used in SATPLAN06 [4], winner of the IPC 2006 for optimal deterministic planning, on a number of benchmarks from the International Planning Competitions 2004 and 2006. We used the SAT Solver SIEGE [7] and averaged the runtimes and serial plan lengths over 50 runs. In cases where a run exceeded 1200 seconds, we took the average runtime of 5 runs. Cases in which a run exceeded 3600 seconds are indicated with a dash. The results for the SATPLAN06, the $\exists$-step and relaxed $\exists$-step encoding are compared in Table 1. We report the parallel and average serial plan length, as well as the total evaluation time of the formulae by a sequential evaluation (i.e. if Φ^{i-1} is found unsatisfiable, then test Φ^i for $i = 1, 2, 3, \ldots$ until the first satisfiable formula is found). The runtime data only include the time for solving the formulae. The generation of disabling-enabling-graphs is possible almost as efficiently as the generation of disabling graphs [3]. The results were obtained on a PC running at 2.8 GHz with 2 GB main memory, and a LINUX operating system.

In the PSR, Airport and Pathways domains we get (sometimes significantly) better runtimes due to shorter parallel plan lengths. In particular, the PSR domain makes use of this more relaxed semantics. Storage is interesting because except for one instance (storage 17) we do not get shorter plans than with $\exists$-step semantics, but in that instance the last unsatisfiable formula gets much easier to solve. A more sophisticated heuristic ordering of the SCCs would possibly yield shorter plans and better runtimes in general. Furthermore, we implemented a domain structured similarly to Example 1 in which a hoist has to fill boxes with three objects in a predefined order[4]. We obtain a significant speedup in this setting as the more relaxed $\exists$-step plans get only half as long as the corresponding $\exists$-step plans. Moreover, we did experiments for other domains but give only a short overview of the results because of lack of space. In the tested instances, we can summarize the results as follows. Philosophers is easy for all the encodings, a fraction of a second is needed to solve the largest instances. In comparison to the SATPLAN06 encoding, the runtimes of the relaxed $\exists$-step encoding are (sometimes significantly) lower in the Openstacks and Pipesworld domain, and comparable (i.e. sometimes better, sometimes worse) in TPP. In comparison to the $\exists$-step encoding, the runtimes are comparable in Pipesworld and TPP, and slightly worse in Openstacks.

The parallel plan lengths of the relaxed $\exists$-step encoding usually decreases or is equal in comparison to the other encodings. The quality of the serial plans for $\exists$-step and relaxed $\exists$-step is comparable: The average serial plans produced by the relaxed $\exists$-step encoding are equally long or at most slightly longer than those produced by the $\exists$-step encoding in most of the cases. As we do no postprocessing step to eliminate redundant operators, the serial plans produced by SATPLAN06 are usually shorter than those produced by the (relaxed) $\exists$-step encodings.

[4] Where boxes n means that n boxes have to be filled.

Table 1. Experimental results with SATPLAN06, ∃-step and relaxed ∃-step encodings

	SP06 parallel length	*SP06 serial length*	*SP06 total time*	*∃-step parallel length*	*∃-step serial length*	*∃-step total time*	*relaxed parallel length*	*relaxed serial length*	*relaxed total time*
psr 46	29	34	246.9	28	42	16.4	16	43	1.1
psr 47	23	27	11.1	21	35	0.6	12	38	0.1
psr 48	26	37	42.7	24	42	2.3	15	43	0.3
psr 49	36	47	874	34	56	132.6	22	58	8.7
airport 17	28	88	0.5	28	88	1	25	88	0.3
airport 18	31	107	3.1	31	108	6.6	26	109	0.5
airport 19	30	90	0.9	30	90	0.8	25	90	0.3
airport 20	32	115	5.4	32	116	14.0	27	115	1.2
pathways 5	9	30	0.04	9	40	0.1	7	47	0.1
pathways 6	12	55	1.2	12	69	1.3	10	69	0.9
pathways 7	13	72	38.3	13	90	7.1	11	97	8.1
pathways 8	-	-	-	-	-	-	14	134	2339
storage 15	9	25	11.5	6	22	0.09	6	23	0.1
storage 16	-	-	-	7	28	0.5	7	29	0.5
storage 17	-	-	-	8	30	78.4	7	31	0.5
storage 18	-	-	-	9	36	50.2	9	38	40.4
boxes 5	21	25	9.8	20	30	2	10	29	0.1
boxes 6	25	30	149.7	24	37	28.3	12	34	0.2
boxes 7	-	-	-	28	45	354	14	52	0.9
boxes 8	-	-	-	-	-	-	16	63	2.6

5 Related Work

Planning as satisfiability was pioneered by Kautz and Selman. They presented a first translation of a planning problem into the propositional logic [8] as well as the standard encoding for parallel operator application [1]. In a further approach [9], they combined planning as satisfiability with the GraphPlan algorithm [10] resulting in the well known BLACKBOX planner. This planner has been further developed to SATPLAN06 [4].

Dimopoulos et al. [2] noticed that the notion of parallel plans used by Blum and Furst can be relaxed to what we have called ∃-step semantics. Cayrol et al. [11] implemented this idea in the GraphPlan framework. Rintanen et al. [3] proposed propositional encodings for the ∃-step semantics for arbitrary operators that restrict parallel operator application in a state s to operators that are already applicable in s.

Van den Briel et al. [12] used a concept of parallelism that is similar to our relaxed ∃-step semantics for an integer programming approach. They, however, presented a loosely constrained approach causing an exponential number of cycle elimination constraints, which their search algorithm has to take into account explicitly. Therefore, they considered it unfeasible to generate all the required constraints for guaranteeing a total ordering of parallel actions, and instead generate the necessary constraints during search. We avoided this problem by enforcing a fixed ordering on the number of

operators of non-trivial SCCs. This may rule out some plans, but leads to small formulae which can be solved very efficiently. For our approach, we do not need specialized search algorithms, but can use arbitrary SAT solver to perform the planning task.

6 Conclusions

We have given a translation of a relaxed semantics for parallel planning into SAT and shown that it is efficient in domains of the recent International Planning Competitions if they exploit the relaxation of our semantics. In this case, we are often one order of magnitude faster than the so far most efficient encodings. Our encoding is restricted to STRIPS because the generalization does not seem to be possible for general ADL problems due to disjunctive preconditions of operators. As various planning domains and different semantics have been developed in the last years, it will be interesting for further research if there are general structures of planning domains that are well suited for certain types of semantics. In this context, there is also the question whether there are other classes of parallel plans that lead to efficient planning as satisfiability.

References

1. Kautz, H., Selman, B.: Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In: Proceedings of the 13th National Conference on Artificial Intelligence. (1996) 1194–1201
2. Dimopoulos, Y., Nebel, B., Koehler, J.: Encoding Planning Problems in Nonmonotonic Logic Programs. In: Proceedings of the 4th European Conference on Planning. (1997) 169–181
3. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search. Artificial Intelligence **170** (2006) 1031–1080
4. Kautz, H., Selman, B., Hoffmann, J.: SatPlan: Planning as Satisfiability. Abstracts of the 5th International Planning Competition (2006)
5. Rintanen, J.: A Planning Algorithm not Based on Directional Search. In: Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning. (1998) 617–624
6. Tarjan, R.E.: Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing **1** (1972) 146–160
7. Ryan, L.: Efficient Algorithms for Clause-Learning SAT Solvers. Master's thesis, Simon Fraser University (2004)
8. Kautz, H., Selman, B.: Planning as Satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence. (1992) 359–363
9. Kautz, H., Selman, B.: Unifying SAT-based and Graph-based Planning. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence. (1999) 318–325
10. Blum, A.L., Furst, M.L.: Fast Planning through Planning Graph Analysis. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence. (1995) 1636–1642
11. Cayrol, M., Régnier, P., Vidal, V.: Least Commitment in Graphplan. Artificial Intelligence **130** (2001) 85–118
12. van den Briel, M., Vossen, T., Kambhampati, S.: Reviving Integer Programming Approaches for AI Planning: A Branch-and-Cut Framework. In: Proceedings of the 15th International Conference on Automated Planning and Scheduling. (2005) 310–319