Habilitationsschrift

Automated Planning: Algorithms and Complexity

vorgelegt von

Dr. Jussi Rintanen

Albert-Ludwigs-Universität Freiburg
Institut für Informatik

Juli 2005

Contents

Chapter 1

Introduction

Planning in Artificial Intelligence is a formalization of *decision making* about the *actions* to be taken. Consider an intelligent robot. The robot is a computational mechanism that takes input through its sensors that allow the robot to *observe* its environment and to build *a representation* of its immediate surroundings and parts of the world it has observed earlier. For a robot to be useful it has to be able to *act*. A robot acts through its *effectors* which are devices that allow the robot to move itself and other objects in its immediate surroundings. A robot resembling a human being has hands and feet, or their muscles, as effectors.

At an abstract level, a robot is a mechanism that maps its observations, which are obtained through the sensors, to actions which are performed by means of the effectors. Planning is the decision making needed in producing a sequence of actions given a sequence of observations. The more complicated the environment and the tasks of the robot are, the more intelligent the robot has to be. For genuine intelligence it is important that the robot is able to plan its actions also in challenging situations.

We view planning as *an algorithmic problem*. Given a formalization of the world and its dynamics an algorithm is used in finding a plan that satisfies the objectives the agent has.

In the simplest cases when the world and the actions are deterministic and the initial state of the world is known, planning can be viewed as finding a path in the transition graph describing the effects of different actions to the state of the world. The use of conventional s-t-reachability algorithms, however, is often not practical because of the potentially very high number of states. Indeed, the representation of the world and the actions typically used in AI planning is based on state variables and operators describing the effects of the actions, and consequently the size of the transition graph may be exponential in the size of the problem description. Most work on AI planning concentrates on utilizing the regularities of the problem description to avoid the explicit construction of the transition graph.

Planning is complicated by different kinds of incompleteness the knowledge of the agent has about the current and the future states of the world. First, the unpredictability of the effects of some actions may make it difficult to achieve the desired goals. The unpredictability of the actions and the world is known as *nondeterminism*: when a given action is taken, the successor state of the current world state is not determined uniquely by the action, and hence there may be several possible successor states. Nondeterminism may be due to inherent unpredictability of the world or simply a product of the incompleteness of the knowledge about the dynamics of the world. We do not further distinguish between these two types of nondeterminism.

Second, the current state of the world might not be uniquely known to the agent. The agent may

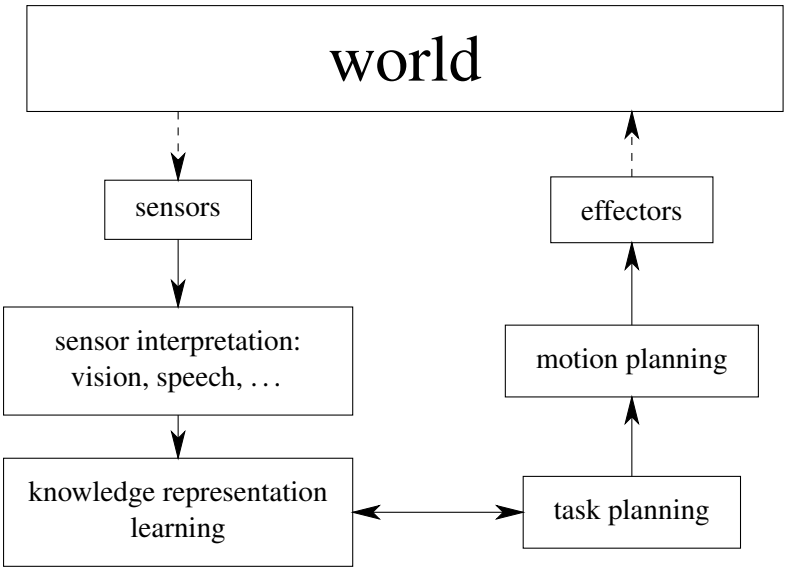


Figure 1.1: Software architecture of an intelligent robot

be capable of sensing and observing some aspects of the world, but other aspects of the world are inaccessible to it. This is known as *partial observability*. Partial observability makes it necessary to formalize the knowledge an agent has. The knowledge determines which states of the world the agent considers as possible current states, and possibly also assigns different degrees of probability to the possible current states.

Efficient and general planning algorithms are needed for building intelligent systems that are able to adjust to unforeseen situations quickly and robustly and choose the appropriate actions that lead to achieving the goals.

The most general formalizations of planning are intractable. The term *domain-independent planning* refers to a class of problems expressible in a general problem description language and to algorithms for solving any problem in the class.

The most commonly used languages used in domain-independent planning, for example the PDDL language [Ghallab *et al.*, 1998], are based on state variables and operators. Languages that use state variables are very powerful and many NP-hard problems, including the satisfiability problem of the classical propositional logic and the halting problems of some classes of resource-bounded Turing machines, are easily expressible as planning problems, which makes domain-independent planning in its general forms intractable. All known algorithms for most types of planning problems require exponential time in the worst case.

Intractability does not necessarily imply high runtimes of algorithms in *typical* or *average* cases. The research problems that arises in this setting include identifying the sources of computational difficulty in different classes of planning problems, finding techniques for tackling computational difficulty, and devising algorithms that are efficient for as wide classes of planning problems as possible.

1.1 Contributions of this work

This work makes contributions to several areas in AI planning: analysis of the computational complexity of different types of planning problems, algorithms for planning under the assumption of determinism and one initial state, and algorithms for planning under incomplete information.

The non-probabilistic planning problems with incomplete information addressed by this work can be seen as discrete non-probabilistic relatives of the policy construction problems of Markov decision processes (MDPs) [Howard, 1960; Puterman, 1994] and partially observable MDPs (POMDPs) [Smallwood and Sondik, 1973; Sondik, 1978]. As we ignore probabilities, we cannot speak about expected rewards and expected costs as in the MDP and POMDP frameworks, which makes it difficult to quantify the value of a plan in terms of typical behavior. However, contrary to the POMDP policy construction problem that is unsolvable when policies with at least a given value are sought for [Madani *et al.*, 2003], planning problems addressed in this work are solvable.

A main difference to conventional MDP and POMDP formulations in our work as in most of AI planning is that problem instances are represented in terms of a succinct state-variable based representation that allows exponentially more concise description of problem instances that the enumerative representation of transition relations and transition probabilities. Similar approach can also been used in connection with POMDPs [Mundhenk *et al.*, 2000].

A practically important subclass of planning problems is those with deterministic actions and one initial state. This subclass is known as *classical planning*. Because of the lack of uncertainty about the initial state and the effects of actions, the state of the world after every action can be predicted unambiguously. For this special case it suffices to consider only plans that are sequences of actions.

The results of this work cover some of the main approaches that have been presented for classical planning. We have generalized some of the earlier techniques used in connection with heuristic state-space search, and present a range of new techniques for plan search by means of algorithms for the satisfiability problem of the classical propositional logic.

The contributions of this work are as follows.

Complexity analysis of non-probabilistic planning with partial observability

1. The complexity results in Chapter 4.

   A part of the results have been published [Rintanen, 2004a].

   The main result establishes the 2-EXP-completeness of general non-probabilistic planning with partial observability. We have analyzed the decision problem of whether a plan exists. The corresponding decision problem for partially observable Markov Decision Processes of whether there is a policy with at least a given value is not solvable [Madani *et al.*, 2003]. It has been argued that conditional planning is not feasible because the associated plans are big, and robotic control and other applications should use algorithms that only decide one action ahead. However, the 2-EXP-hardness of plan existence directly also implies the 2-EXP-hardness of deciding whether taking a given action is safe in the sense that it does not make the goals unreachable. Hence deciding about the first action to be taken is computationally just as difficult as deciding general plan existence. Of course, it may be a practical consideration that a complete plan for reaching a goal does not have to be represented explicitly.

Earlier research has identified the complexity of probabilistic planning problems with different restriction to observability and different horizon lengths [Littman, 1997; Mundhenk *et al.*, 2000], the complexity of the nondeterministic planning problem without observability [Haslum and Jonsson, 2000], and the complexity of probabilistic and nondeterministic planning problems under different observability assumptions assuming plans of polynomial size and/or polynomial execution length [Littman *et al.*, 1998; Rintanen, 1999; Baral *et al.*, 2000; Turner, 2002].

Other related works include the analyses of computational complexity of controller and plan synthesis in a temporal logic framework, possibly under observability restrictions [Pnueli and Rosner, 1989; Vardi, 1995; Kupferman and Vardi, 1997; Giacomo and Vardi, 2000; Calvanese *et al.*, 2002]. In many of the problems analyzed in these works the complexity of the satisfiability problem of the underlying temporal logic dominates the complexity of the controller and plan synthesis problems. For example, the CTL$*$ synthesis problem considered by Kupferman and Vardi [1997] is 2-EXP-complete, which is also the complexity of the satisfiability problem of CTL$*$.

Algorithms for non-probabilistic planning with partial observability

1. The framework for non-probabilistic planning with partial observability in Section 5.4.

   A part of the results have been published [Rintanen, 2005].

   We have presented a representation of dynamic-programming type backward search for planning with partial observability in the discrete belief space without probabilities. Although efficient algorithms for non-probabilistic fully observable planning with a dynamic-programming character exist [Cimatti *et al.*, 2003], recent work on non-probabilistic conditional planning [Bertoli *et al.*, 2001b] had resorted to forward search because of the conceptual simplicity and seemingly because of the conceptual difficulty of backward search in the belief space. Backward search approaches in general are more effective in avoiding repeatedly solving the same subproblems.

   Related works include heuristic search algorithms solving partially observable planning problems [Bonet and Geffner, 2000; Bertoli *et al.*, 2001b; 2001a], and algorithms for solving MDPs and POMDPs [Howard, 1960; Sondik, 1971; Smallwood and Sondik, 1973]. The works on MDPs and POMDPs are based on dynamic-programming type updates similar to our work, and our work could be viewed as a discrete counterpart of the POMDP framework of Sondik and Smallwood.

2. Representation of non-probabilistic planning with nondeterminism and partial observability as evaluation of quantified Boolean formulae. The approach is very general but in Section 5.3.1 we present the basic case without observability only.

   A part of the results have been published [Rintanen, 1999].

   The basic insight is that even under plan size restrictions plan search for problems with nondeterminism or observability restrictions cannot in general be usefully viewed as a satisfiability problem as in the deterministic one-initial state case [Kautz and Selman, 1992; 1996]. The problem has a quantification structure requiring the use of QBF or some other generalization of the propositional satisfiability problem: *there is* a plan such that *for all* possible eventualities *there is* an execution of the plan leading to a goal state. This naturally

leads to quantified Boolean formulae with three quantifiers $\exists\forall\exists$. Since the plan and the set of eventualities unambiguously determine the execution which can be found in polynomial time these QBF correspond to decision problems on the second level of the polynomial hierarchy: is there a polynomial-size plan that always reaches a goal state with a polynomially long execution.

Our work introduced the use of quantified Boolean formulae QBF in solving advanced reasoning tasks in AI and about transition systems. Many follow-up works exist. This includes the use of a satisfiability algorithm for testing whether a given candidate plan found by an ad hoc search algorithm is a valid plan [Castellini *et al.*, 2003] and complexity analyses of planning under partial observability based on QBF [Turner, 2002].

3. The definition of a family of distance heuristics for planning with partial observability in Section 5.3.2. Our definition of n-distances generalizes most of the heuristics defined earlier for partially observable problems

   A part of the results have been published [Rintanen, 2004b].

   Earlier heuristics include the cardinality of belief states [Bertoli *et al.*, 2001a] and the distances/costs of the fully observable problem [Bonet and Geffner, 2000]. Recent work has concentrated on planning graph based [Blum and Furst, 1997] implementation techniques for approximating the distances obtained by relaxing the problem to a corresponding fully observable problem [Bryce and Kambhampati, 2004]. Smith and Weld [1998] gave a heuristic that can be viewed as a specialized planning graph based approximation of our n-distances but restricted to deterministic problems.

4. The algorithm for planning with full observability based on the procedure *prune* in Section 4.3.1. This algorithm generalizes an algorithm presented by Cimatti et al. [2003] for solving the special case of the problem restricted to reaching a given goal state. These algorithms are loosely related to the value iteration algorithm for Markov decision processes. Since the MDP model uses expected rewards as the criterion for comparing plans and not a designated set of goal states that must be reached the non-probabilistic problem we solve is different from the MDP problem. Simulation of goal states by rewards in the MDP model is not accurate in the sense that a plan for an MDP that maximizes rewards that are only obtained in goal states may fail to reach a goal state with a non-zero probability.

Algorithms for deterministic (classical) planning

1. Generalization of a range of techniques introduced for solving the deterministic planning problem to a richer plan definition language that includes conditional effects, including the generalization of some of the main distance heuristics [Bonet and Geffner, 2001; Hoffmann and Nebel, 2001] (Section 3.3) and the introduction of a regression operator for operators with conditional effects (Section 3.1.2).

   Practically all earlier works on these topics were restricted to the class of STRIPS operators which have a conjunction of (positive) literals as a precondition and effects (assignments of values to state variables) that are executed unconditionally. Our work shows how a small and relatively simple set of concepts suffices for representing many of the important techniques uniformly and concisely.

2. The algorithm for computing invariants in Section 3.5.

   An early variant of the algorithm has been published [Rintanen, 1998].

   Invariant in planning have been used for pruning search trees. This differs from the applications of invariants in computer-aided verification where invariants are desirable properties of a transition system [Bensalem *et al.*, 1996].

   The computation of mutexes in Blum and Furst's [1997] planning graphs is closely related to my invariant algorithm but there are also major differences. Similarly to my invariant algorithm Blum and Furst's construction essentially performs an approximative reachability analysis that gives upper bounds on the set of states that are reachable in a given number of time steps. However, Blum and Furst restrict to a narrow class of operators with unconditional effects only, and their construction allows the application of several operators in one time point. For this reason the intermediate stages in my invariant computation give a lower bound on the number of actions needed to reach a state while Blum and Furst's construction does not. Hence the former yields an *admissible heuristic* for guiding a heuristic search algorithm for planning and the latter does not.

   Other works that introduce algorithms for computing invariants include [Gerevini and Schubert, 1998; Fox and Long, 1999]. In comparison to these works our algorithm is conceptually simpler and equally or more powerful.

3. The analysis of different notions of partially-ordered plans in Section 3.7.

   The best known notion of parallel or partially-ordered plans stems from the research on partial-order planning [Sacerdoti, 1975; McAllester and Rosenblitt, 1991]. If two operators have no mutual ordering constraint they can be ordered in any order. Partial orderings in this context are related to partial-order reduction techniques used in computer-aided verification [Godefroid, 1991; Valmari, 1991].

   A slightly more restrictive notion of partially-ordered plans have been used in the Graphplan algorithm [Blum and Furst, 1997] and in planning by satisfiability [Kautz and Selman, 1996; 1999] in which the partial orders are required to be *modular*. A strict partial order $<$ on a countable set is modular if all the elements i can be assigned an integer $f(i)$ so that $i < j$ iff $f(i) < f(j)$.

   The motivation for the use of partially ordered plans in the Graphplan algorithm and in planning by satisfiability is that planning is more efficient if less time points are needed for the plan. For satisfiability algorithms a smaller number of time points directly leads to smaller formulae and less propositional variables, and hence to a smaller search space.

   A more relaxed notion of partially-ordered plans was proposed by Dimopoulos et al. [1997].

   We have analyzed the computational complexity of these two notions of partially-ordered plans and given results indicating the boundary between more restricted tractable and intractable definitions of partially-ordered plans.

   These results are also applicable to *bounded model-checking* [Biere *et al.*, 1999] which is an extension of planning by satisfiability.

4. Asymptotically optimal encodings of partially ordered plans in the classical propositional logic in Section 3.8.

These results and those of Section 3.7 are joint work with Keijo Heljanko and Ilkka Niemelä and have been published as a technical report [Rintanen *et al.*, 2005] and submitted for journal publication.

Earlier encodings of partially-ordered plans [Kautz and Selman, 1996; 1999] in a declarative language like the propositional logic had a quadratic size. In our work we have given asymptotically optimal linear-size encodings of the main tractable notions of partially-ordered plans. Our encodings make planning as satisfiability practical to many problems that were earlier difficult to solve because of high formula size or high runtimes.

5. Two new algorithms for controlling a plan search process based on looking for plans of a given length in Section 3.9.

   A part of the results have been published [Rintanen, 2004c].

   All previous work on planning by satisfiability [Kautz and Selman, 1992; 1996; 1999] and related approaches [Blum and Furst, 1997] were based on testing the existence of a plan of a given length, and if it was shown that no plans of that length exists, the plan length was increased. This procedure guarantees that the first plan that is found has the minimum length.

   In practice the procedure is very inefficient if the objective is to find any plan, not necessarily the optimal one, because proving that plans of length $n - 1$ do not exist is typically much more expensive than finding a plan of length n, where n is the length of the shortest plan. Furthermore, planning by satisfiability [Kautz and Selman, 1992; 1996] and related approaches [Blum and Furst, 1997] have almost exclusively been used in connection with plan notions for which the plan length parameter does not correspond to the most natural plan size criterion, the number of operators in a plan, but to a less directly useful measure of the number of time points in a plan. Hence the procedure that guarantees optimality with respect to number of time points does not in general guarantee the smallest number of operators.

   The algorithms we have proposed allow trading optimality with respect to the number of time points to improved runtimes to find a plan. The algorithms attempt to find plans for different numbers of time points simultaneously and in many cases avoids the very expensive proofs that plans of certain lengths do not exist. Our algorithm and experiments demonstrate that much of the perceived superiority of heuristic state-space search over planning as satisfiability on planning benchmarks when optimality guarantees are not required is due to the fact that planners based on satisfiability testing were giving optimality guarantees or proofs of close-to-optimality whereas the heuristic state-space planners did not.

Chapter 2

Preliminaries

In this chapter we define the formal machinery needed in the rest of the work for describing different planning problems and algorithms. We give the basic definitions related to the classical propositional logic, theory of computational complexity, and the definition of the transition system model that is the basis of most work on planning.

2.1 Transition systems

We define transition systems in which states are atomic objects and actions are represented as binary relations on the set of states.

Definition 2.1 *A* transition system *is a 5-tuple* $\Pi = \langle S, I, O, G, P \rangle$ *where*

1. S *is a finite set of states,*
2. $I \subseteq S$ *is the set of initial states,*
3. O *is a finite set of actions* $o \subseteq S \times S$*,*
4. $G \subseteq S$ *is the set of goal states, and*
5. $P = (C_1, \ldots, C_n)$ *is a partition of* S *to non-empty classes of observationally indistinguishable states satisfying* $\bigcup\{C_1, \ldots, C_n\} = S$ *and* $C_i \cap C_j = \emptyset$ *for all* i, j *such that* $1 \leq i < j \leq n$*.*

Making an observation tells which set C_i the current state belongs to. Distinguishing states within a given C_i is not possible by observations. If two states are observationally distinguishable then plan execution can proceed differently for them.

The number n of components in the partition P determines different classes of planning problems with respect to observability restrictions. If $n = |S|$ then every state is observationally distinguishable from every other state. This is called *full observability*. If $n = 1$ then no observations are possible and the transition system is *unobservable*. The general case $n \in \{1, \ldots, |S|\}$ is called *partial observability*.

An action o is *applicable* in states for which it associates at least one successor state. We define *images* of states as $img_o(s) = \{s' \in S | sos'\}$ and (weak) *preimages* of states as $preimg_o(s') = \{s \in S | sos'\}$. Generalization to sets of states is $img_o(T) = \bigcup_{s \in T} img_o(s)$ and $preimg_o(T) =$

$\bigcup_{s \in T} \mathit{preimg}_o(s)$. For sequences $o_1, \ldots, o_n$ of actions $\mathit{img}_{o_1;\ldots;o_n}(T) = \mathit{img}_{o_n}(\cdots \mathit{img}_{o_1}(T) \cdots)$ and $\mathit{preimg}_{o_1;\ldots;o_n}(T) = \mathit{preimg}_{o_1}(\cdots \mathit{preimg}_{o_n}(T) \cdots)$. The *strong preimage* of a set T of states is the set of states for which all successor states are in T, defined as $\mathit{spreimg}_o(T) = \{s \in S | s' \in T, sos', \mathit{img}_o(s) \subseteq T\}$.

Lemma 2.2 *Images, strong preimages and weak preimages of sets of states are related to each other as follows. Let o be any action and S and S' any sets of states.*

1. $\mathit{spreimg}_o(T) \subseteq \mathit{preimg}_o(T)$
2. $\mathit{img}_o(\mathit{spreimg}_o(T)) \subseteq T$
3. *If* $T \subseteq T'$ *then* $\mathit{img}_o(T) \subseteq \mathit{img}_o(T')$.
4. $\mathit{preimg}_o(T \cup T') = \mathit{preimg}_o(T) \cup \mathit{preimg}_o(T')$.
5. $s' \in \mathit{img}_o(s)$ *if and only if* $s \in \mathit{preimg}_o(s)$.

Proof:

1. $\mathit{spreimg}_o(T) = \{s \in S | s' \in T, sos', \mathit{img}_o(s) \subseteq T\} \subseteq \{s \in S | s' \in T, sos'\} = \bigcup_{s' \in T} \{s \in S | sos'\} = \bigcup_{s' \in T} \mathit{preimg}_o(s') = \mathit{preimg}_o(T)$.
2. Take any $s' \in \mathit{img}_o(\mathit{spreimg}_o(T))$. Hence there is $s \in \mathit{spreimg}_o(T)$ so that sos'. As $s \in \mathit{spreimg}_o(T)$, $\mathit{img}_o(s) \subseteq T$. Since $s' \in \mathit{img}_o(s)$, $s' \in T$.
3. Assume $T \subseteq T'$ and $s' \in \mathit{img}_o(T)$. Hence sos' for some $s \in T$ by definition of images. Hence sos' for some $s \in T'$ because $T \subseteq T'$. Hence $s' \in \mathit{img}_o(T')$ by definition of images.
4. $\mathit{preimg}_o(T \cup T') = \bigcup_{s' \in T \cup T'} \{s \in S | sos'\} = \bigcup_{s' \in T} \{s \in S | sos'\} \cup \bigcup_{s' \in T'} \{s \in S | sos'\} = \mathit{preimg}_o(T) \cup \mathit{preimg}_o(T')$
5. $s' \in \mathit{img}_o(s)$ iff sos' iff $s \in \mathit{preimg}_o(s)$.

□

2.1.1 Deterministic transition systems

Transition systems which we use in Chapter 3 have only one initial state and deterministic actions. For this subclass observability is irrelevant because the state of the transition system after a given sequence of actions can be predicted exactly. We use a simpler formalization of them.

Definition 2.3 *A* deterministic transition system *is a 4-tuple* $\Pi = \langle S, I, O, G \rangle$ *where*

1. S *is a finite set of states,*
2. $I \in S$ *is the initial state,*
3. O *is a finite set of actions* $o \subseteq S \times S$ *that are partial functions, and*
4. $G \subseteq S$ *is the set of goal states.*

That the actions are partial functions means that for any $s \in S$ and $o \in O$ there is at most one state s' such that sos'. We denote the unique successor state s' of a state s in which operator o is applicable by $s' = app_o(s)$. For sequences $o_1; \ldots; o_n$ of operators we define $app_{o_1;\ldots;o_n}(s)$ as $app_{o_n}(\cdots app_{o_1}(s)\cdots)$.

2.2 Classical propositional logic

Let A be a set of propositional variables (atomic propositions). We define the set of propositional formulae inductively as follows.

1. For all $a \in A$, a is a propositional formula.
2. If ϕ is a propositional formula, then so is $\neg\phi$.
3. If ϕ and ϕ' are propositional formulae, then so is $\phi \vee \phi'$.
4. If ϕ and ϕ' are propositional formulae, then so is $\phi \wedge \phi'$.
5. The symbols $\bot$ and $\top$, respectively denoting truth-values false and true, are propositional formulae.

The symbols $\wedge$, $\vee$ and $\neg$ are *connectives* respectively denoting the *conjunction*, *disjunction* and *negation*. We define the implication $\phi \rightarrow \phi'$ as an abbreviation for $\neg\phi \vee \phi'$, and the equivalence $\phi \leftrightarrow \phi'$ as an abbreviation for $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

A valuation of A is a function $v : A \rightarrow \{0, 1\}$ where 0 denotes false and 1 denotes true. Valuations are also known as *assignments* or *models*. For propositional variables $a \in A$ we define $v \models a$ if and only if $v(a) = 1$. A valuation of the propositional variables in A can be extended to a valuation of all propositional formulae over A as follows.

1. $v \models \neg\phi$ if and only if $v \not\models \phi$
2. $v \models \phi \vee \phi'$ if and only if $v \models \phi$ or $v \models \phi'$
3. $v \models \phi \wedge \phi'$ if and only if $v \models \phi$ and $v \models \phi'$
4. $v \models \top$
5. $v \not\models \bot$

Computing the truth-value of a formula under a given valuation of propositional variables is polynomial time in the size of the formula by the obvious recursive procedure.

A propositional formula ϕ is *satisfiable* (*consistent*) if there is at least one valuation v so that $v \models \phi$. Otherwise it is *unsatisfiable* (*inconsistent*). A finite set F of formulae is satisfiable if $\bigwedge_{\phi \in F} \phi$ is. A propositional formula ϕ is *valid* or a *tautology* if $v \models \phi$ for all valuations v. We denote this by $\models \phi$. A propositional formula ϕ is *a logical consequence* of a propositional formula ϕ', written $\phi' \models \phi$, if $v \models \phi$ for all valuations v such that $v \models \phi'$. A propositional formula that is a proposition variable a or a negated propositional variable $\neg a$ for some $a \in A$ is *a literal*. A formula that is a disjunction of literals is *a clause*.

A formula ϕ is in *negation normal form* (NNF) if all occurrences of negations are directly in front of propositional variables. Any formula can be transformed to negation normal form by

applications of the De Morgan rules $\neg(\phi \vee \phi') \equiv \neg\phi \wedge \neg\phi'$ and $\neg(\phi \wedge \phi') \equiv \neg\phi \vee \neg\phi'$, the double negation rule $\neg\neg\phi \equiv \phi$. A formula ϕ is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions of literals. A formula ϕ is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctions of literals. Any formula in CNF or in DNF is also in NNF.

2.2.1 Quantified Boolean formulae

There is an extension of the satisfiability and validity problems of the classical propositional logic with quantification over the truth-values of propositional variables. *Quantified Boolean formulae* (QBF) are like propositional formulae but there are two new syntactic rules for the quantifiers.

6. If ϕ is a formula and $a \in A$, then $\forall a\phi$ is a formula.

7. If ϕ is a formula and $a \in A$, then $\exists a\phi$ is a formula.

Further, there is the requirement that every variable is quantified, that is, every occurrence of $a \in A$ in a QBF is in the scope of either $\exists a$ or $\forall a$.

Define $\phi[\psi/x]$ as the formula obtained from ϕ by replacing occurrences of the propositional variable x by ψ.

We define the truth-value of QBF by reducing them to ordinary propositional formulae without occurrences of propositional variables. The atomic formulae in these formulae are the constants $\top$ and $\bot$. The truth-value of these formulae is independent of the valuation, and is recursively computed by the Boolean functions associated with the connectives $\vee$, $\wedge$ and $\neg$.

Definition 2.4 (Truth of QBF) *A formula $\exists x\phi$ is true if and only if $\phi[\top/x] \vee \phi[\bot/x]$ is true. (Equivalently, if $\phi[\top/x]$ is true or $\phi[\bot/x]$ is true.)*

A formula $\forall x\phi$ is true if and only if $\phi[\top/x] \wedge \phi[\bot/x]$ is true. (Equivalently, if $\phi[\top/x]$ is true and $\phi[\bot/x]$ is true.)

A formula ϕ with an empty prefix (and consequently without occurrences of propositional variables) is true if and only if ϕ is satisfiable (equivalently, valid: for formulae without propositional variables validity coincides with satisfiability.)

Example 2.5 The formulae $\forall x\exists y(x \leftrightarrow y)$ and $\exists x\exists y(x \wedge y)$ are true.

The formulae $\exists x\forall y(x \leftrightarrow y)$ and $\forall x\forall y(x \vee y)$ are false. ■

Note that a QBF with only existential quantifiers is true if and only if the formula without the quantifiers is satisfiable. Similarly, truth of QBF with only universal quantifiers coincides with the validity of the corresponding formulae without quantifiers.

Changing the order of two consecutive variables quantified by the same quantifier does not affect the truth-value of the formula. It is often useful to ignore the ordering in these cases and to view each quantifier as quantifying a set of formulae, for example $\exists x_1 x_2 \forall y_1 y_2 \phi$.

Quantified Boolean formulae are interesting because evaluating their truth-value is PSPACE-complete [Meyer and Stockmeyer, 1972], and many computational problems that presumably cannot be translated into the satisfiability problem of the propositional logic in polynomial time (assuming that NP≠PSPACE) can be efficiently translated into QBF.

2.3 Succinct transition systems

It is often more natural to represent the states of a transition system as valuations of state variables instead of enumeratively as in Section 2.1. The binary relations that correspond to actions can often be represented compactly in terms of the changes the actions cause to the values of state variables.

We represent states in terms of a set A of Boolean state variables which take the values *true* or *false*. Each *state* is a valuation of A (a function $s : A \to \{0, 1\}$.)

Definition 2.6 *Let A be a set of state variables. An* operator *is a pair $\langle c, e\rangle$ where c is a propositional formula over A (the* precondition*), and e is an* effect *over A. Effects over A are recursively defined as follows.*

1. *a and $\neg a$ for state variables $a \in A$ are effects over A.*
2. *$e_1 \wedge \cdots \wedge e_n$ is an effect over A if $e_1, \ldots, e_n$ are effects over A (the special case with $n = 0$ is the empty effect $\top$).*
3. *$c \rhd e$ is an effect over A if c is a formula over A and e is an effect over A.*
4. *$e_1 | \cdots | e_n$ is an effect over A if $e_1, \ldots, e_n$ for $n \geq 2$ are effects over A.*

The compound effects $e_1 \wedge \cdots \wedge e_n$ denote executing all the effects $e_1, \ldots, e_n$ simultaneously. In conditional effects $c \rhd e$ the effect e is executed if c is true in the current state. The effects $e_1 | \cdots | e_n$ denote nondeterministic choice between the effects $e_1, \ldots, e_n$. Exactly one of these effects is chosen randomly.

Operators describe a binary relation on the set of states as follows.

Definition 2.7 (Operator application) *Let $\langle c, e\rangle$ be an operator over A. Let s be a state (a valuation of A). The operator is* applicable in *s if $s \models c$ and every set $E \in [e]_s$ is consistent. The set $[e]_s$ is recursively defined as follows.*

1. *$[a]_s = \{\{a\}\}$ and $[\neg a]_s = \{\{\neg a\}\}$ for $a \in A$.*
2. *$[e_1 \wedge \cdots \wedge e_n]_s = \{\bigcup_{i=1}^{n} E_i | E_1 \in [e_1]_s, \ldots, E_n \in [e_n]_s\}$.*
3. *$[c' \rhd e]_s = [e]_s$ if $s \models c'$ and $[c' \rhd e]_s = \{\emptyset\}$ otherwise.*
4. *$[e_1 | \cdots | e_n]_s \;=\; [e_1]_s \cup \cdots \cup [e_n]_s$.*

An operator $\langle c, e\rangle$ induces a binary relation $R\langle c, e\rangle$ on states as follows: states s and s' are related by $R\langle c, e\rangle$ if $s \models c$ and s' is obtained from s by making the literals in some $E \in [e]_s$ true and retaining the values of state variables not occurring in E.

We call the sets in $[e]_s$ the sets of alternative *active effects* of e. For an operator $o = \langle c, e\rangle$ we also define $[o]_s = [e]_s$. We define images and preimages for operators o in terms of $R(o)$, for instance by $\mathit{preimg}_o(s) = \mathit{preimg}_{R(o)}(s)$.

Definition 2.8 *A* succinct transition system *is a 5-tuple $\Pi = \langle A, I, O, G, V\rangle$ where*

1. *A is a finite set of state variables,*

2. *I is a formula over A describing the initial states,*

3. *O is a finite set of operators over A,*

4. *G is a formula over A describing the goal states, and*

5. *$V \subseteq A$ is the set of observable state variables.*

Succinct transition systems with $V = A$ are *fully observable*, and succinct transition systems with $V = \emptyset$ are *unobservable*. Without restrictions on V the succinct transition systems are *partially observable*.

We can associate a transition system with every succinct transition system.

Definition 2.9 *Given a succinct transition system $\Pi = \langle A, I, O, G, V\rangle$, define the transition system $F(\Pi) = \langle S, I', O', G', P\rangle$ where*

1. *S is the set of all Boolean valuations of A,*

2. *$I' = \{s \in S | s \models I\}$,*

3. *$O' = \{R(o) | o \in O\}$,*

4. *$G' = \{s \in S | s \models G\}$, and*

5. *$P = (C_1, \ldots, C_n)$ where $v_1, \ldots, v_n$ for $n = 2^{|V|}$ are all the Boolean valuations of V and $C_i = \{s \in S | s(a) = v_i(a)$ for all $a \in V\}$ for all $i \in \{1, \ldots, n\}$.*

The transition system may have a size that is exponential in the size of the succinct transition system. However, the construction takes only polynomial time in the size of the transition system.

2.3.1 Deterministic succinct transition systems

A deterministic operator has no occurrences of | in the effect. Further, in this special case the definition of operator application is slightly simpler.

Definition 2.10 (Operator application) *Let $\langle c, e\rangle$ be a deterministic operator over A. Let s be a state (a valuation of A). The operator is* applicable in *s if $s \models c$ and the set $[e]_s^{det}$ is consistent. The set $[e]_s^{det}$ is recursively defined as follows.*

1. *$[a]_s^{det} = \{a\}$ and $[\neg a]_s^{det} = \{\neg a\}$ for $a \in A$.*

2. *$[e_1 \wedge \cdots \wedge e_n]_s^{det} = \bigcup_{i=1}^{n} [e_i]_s^{det}$.*

3. *$[c' \rhd e]_s^{det} = [e]_s^{det}$ if $s \models c'$ and $[c' \rhd e]_s^{det} = \emptyset$ otherwise.*

A deterministic operator $\langle c, e\rangle$ induces a partial function $R\langle c, e\rangle$ on states as follows: two states s and s' are related by $R\langle c, e\rangle$ if $s \models c$ and s' is obtained from s by making the literals in $[e]_s^{det}$ true and retaining the truth-values of state variables not occurring in $[e]_s^{det}$.

We define $app_o(s) = s'$ by $sR(o)s'$ and $app_{o_1;\ldots;o_n}(s) = s'$ by $app_{o_n}(\ldots app_{o_1}(s)\ldots)$, just like for non-succinct transition systems. For sets T of operators we define $[T]_s^{det} = \bigcup_{o\in T}[o]_s^{det}$. Define $app_T(s)$ as the state that is obtained from s by making the literals in $[T]_s^{det}$ true. For this to be defined must $[T]_s^{det}$ be consistent. We formally define deterministic succinct transition systems.

Definition 2.11 *A* deterministic succinct transition system *is a 4-tuple* $\Pi = \langle A, I, O, G\rangle$ *where*

1. A *is a finite set of state variables,*
2. I *is an initial state,*
3. O *is a finite set of operators over* A*, and*
4. G *is a formula over* A *describing the goal states.*

We can associate a deterministic transition system with every deterministic succinct transition system.

Definition 2.12 *Given a deterministic succinct transition system* $\Pi = \langle A, I, O, G\rangle$*, define the deterministic transition system* $F(\Pi) = \langle S, I, O', G'\rangle$ *where*

1. S *is the set of all Boolean valuations of* A*,*
2. $O' = \{R(o)|o \in O\}$*, and*
3. $G' = \{s \in S|s \models G\}$.

A subclass of operators considered in many early and recent works restrict to *STRIPS* operators. An operator $\langle c, e\rangle$ is a STRIPS operator if c is a conjunction of state variables and e is a conjunction of literals. STRIPS operators do not allow disjunctivity in formulae nor conditional effects. This class of operators is sufficient in the sense that any transition system can be expressed in terms of STRIPS operators only if the identities of operators are not important: when expressing a transition system in terms of STRIPS operators only some operators correspond to an exponential number of STRIPS operators.

Example 2.13 Let $A = \{a_1, \ldots, a_n\}$ be the set of state variables. Let $o = \langle \top, e\rangle$ where

$$e = (a_1 \rhd \neg a_1) \wedge (\neg a_1 \rhd a_1) \wedge \cdots \wedge (a_n \rhd \neg a_n) \wedge (\neg a_n \rhd a_n)\rangle.$$

This operator reverses the values of all state variables. As its set of active effects $[e]_s^{det}$ is different in every one of 2^n states, this operator corresponds to 2^n STRIPS operators.

$$\begin{array}{rcl}
o_0 & = & \langle \neg a_1 \wedge \neg a_2 \wedge \cdots \wedge \neg a_n, a_1 \wedge a_2 \wedge \cdots \wedge a_n\rangle \\
o_1 & = & \langle a_1 \wedge \neg a_2 \wedge \cdots \wedge \neg a_n, \neg a_1 \wedge a_2 \wedge \cdots \wedge a_n\rangle \\
o_2 & = & \langle \neg a_1 \wedge a_2 \wedge \cdots \wedge \neg a_n, a_1 \wedge \neg a_2 \wedge \cdots \wedge a_n\rangle \\
o_3 & = & \langle a_1 \wedge a_2 \wedge \cdots \wedge \neg a_n, \neg a_1 \wedge \neg a_2 \wedge \cdots \wedge a_n\rangle \\
\vdots & & \\
o_{2^n-1} & = & \langle a_1 \wedge a_2 \wedge \cdots \wedge a_n, \neg a_1 \wedge \neg a_2 \cdots \wedge \neg a_n\rangle
\end{array}$$

■

$$
\begin{aligned}
c \rhd (e_1 \wedge \cdots \wedge e_n) &\equiv (c \rhd e_1) \wedge \cdots \wedge (c \rhd e_n) && (2.1)\\
c_1 \rhd (c_2 \rhd e) &\equiv (c_1 \wedge c_2) \rhd e && (2.2)\\
(c_1 \rhd e) \wedge (c_2 \rhd e) &\equiv (c_1 \vee c_2) \rhd e && (2.3)\\
e \wedge (c \rhd e) &\equiv e && (2.4)\\
e &\equiv \top \rhd e && (2.5)\\
e_1 \wedge (e_2 \wedge e_3) &\equiv (e_1 \wedge e_2) \wedge e_3 && (2.6)\\
e_1 \wedge e_2 &\equiv e_2 \wedge e_1 && (2.7)\\
c \rhd \top &\equiv \top && (2.8)\\
e \wedge \top &\equiv e && (2.9)
\end{aligned}
$$

Table 2.1: Equivalences on effects

2.3.2 Extensions

The basic language for effects could be extended with further constructs. A natural construct is *sequential composition* of effects. If e and e' are effects, then also $e; e'$ is an effect that corresponds to first executing e and then e'. Definition 3.11 and Theorem 3.12 show how effects with sequential composition can be reduced to effects without sequential composition.

2.3.3 Normal form for deterministic operators

Deterministic operators can be transformed to a particularly simple form without nesting of conditionality $\rhd$ and with only atomic effects e as antecedents of conditionals $\phi \rhd e$. Normal forms are useful as they allow concentrating on a particularly simple form of effects.

Table 2.1 lists a number of equivalences on effects. Their proofs of correctness with Definition 2.10 are straightforward. An effect e is equivalent to $\top \wedge e$, and conjunctions of effects can be arbitrarily reordered without affecting the meaning of the operator. These trivial equivalences will later be used without explicitly mentioning them, for example in the definitions of the normal forms and when applying equivalences.

The normal form corresponds to moving all occurrences of $\rhd$ inside $\wedge$ so that the consequents of $\rhd$ are atomic effects.

Definition 2.14 *A deterministic effect e is in* normal form *if it is $\top$ or a conjunction of one or more effects $c \rhd a$ and $c \rhd \neg a$ with at most one occurrence of atomic effect a and $\neg a$ for any $a \in A$. An operator $\langle c, e\rangle$ is in normal form if e is in normal form.*

Theorem 2.15 *For every deterministic operator there is an equivalent one in normal form. There is one that has a size that is polynomial in the size of the operator.*

Proof: We can transform any deterministic operator into normal form by using the equivalences in Table 2.1. The proof is by structural induction on the effect e of the operator $\langle c, e\rangle$.

Induction hypothesis: the effect e can be transformed to normal form.

Base case 1, $e = \top$: This is already in normal form.

Base case 2, $e = a$ or $e = \neg a$: An equivalent effect in normal form is $\top \rhd e$ by Equivalence 2.5.

Inductive case 1, $e = e_1 \wedge e_2$: By the induction hypothesis e_1 and e_2 can be transformed into normal form, so assume that they already are. If one of e_1 and e_2 is $\top$, by Equivalence 2.9 we can eliminate it.

Assume e_1 contains $c_1 \rhd l$ for some literal l and e_2 contains $c_2 \rhd l$. We can reorder $e_1 \wedge e_2$ with Equivalences 2.6 and 2.7 so that one of the conjuncts is $(c_1 \rhd l) \wedge (c_2 \rhd l)$. Then by Equivalence 2.3 it can be replaced by $(c_1 \vee c_2) \rhd l$. Since this can be done repeatedly for every literal l, we can transform $e_1 \wedge e_2$ into normal form.

Inductive case 2, $e = z \rhd e_1$: By the induction hypothesis e_1 can be transformed to normal form, so assume that it already is.

If e_1 is $\top$, e can be replaced by $\top$ which is in normal form.

If $e_1 = z' \rhd e_2$ for some z' and e_2, then e can be replaced by the equivalent (by Equivalence 2.2) effect $(z \wedge z') \rhd e_2$ in normal form.

Otherwise, e_1 is a conjunction of effects $z \rhd l$. By Equivalence 2.1 we can move z inside the conjunction. Applications of Equivalences 2.2 transform the effect into normal form.

In this transformation the conditions c in $c \rhd e$ are copied into front of the atomic effects. Let m be the sum of the sizes of all the conditions c, and let n be the number of occurrences of atomic effects a and $\neg a$ in the effect. An upper bound on size of the new effect is $\mathcal{O}(nm)$ which is polynomial in the size of the original effect. □

2.3.4 Normal forms for nondeterministic operators

We can generalize the normal form defined in Section 2.3.3 to nondeterministic effects and operators. In the normal form nondeterministic choices and conjunctions are the outermost constructs, and consequents e of conditional effects $c \rhd e$ are atomic effects.

Definition 2.16 (Normal form for nondeterministic operators) *A deterministic effect is in normal form if it is $\top$ or a conjunction of one or more effects $c \rhd a$ and $c \rhd \neg a$ with at most one occurrence of a and $\neg a$ for any $a \in A$.*

A nondeterministic effect is in normal form if it is $e_1 | \cdots | e_n$ or $e_1 \wedge \cdots \wedge e_n$ for effects e_i that are in normal form.

A nondeterministic operator $\langle c, e \rangle$ is in normal form if e is in normal form.

For showing that every nondeterministic effect can be transformed into normal form we use further equivalences that are given in Table 2.2.

Theorem 2.17 *For every operator there is an equivalent one in normal form. There is one that has a size that is polynomial in the size of the former.*

Proof: Transformation to normal form is like in the proof of Theorem 2.15. Additional equivalences needed for nondeterministic choices are 2.10 and 2.11. □

Example 2.18 The effect

$$a \rhd (b | (c \wedge f)) \wedge ((d \wedge e) | (b \rhd e))$$

$$c \rhd (e_1|\cdots|e_n) \equiv (c \rhd e_1)|\cdots|(c \rhd e_n) \quad (2.10)$$
$$e \wedge (e_1|\cdots|e_n) \equiv (e \wedge e_1)|\cdots|(e \wedge e_n) \quad (2.11)$$
$$(e'_1|\cdots|e'_{n'})|e_2|\cdots|e_n \equiv e'_1|\cdots|e'_{n'}|e_2|\cdots|e_n \quad (2.12)$$
$$(e' \wedge (c \rhd e_1))|e_2|\cdots|e_n \equiv (c \rhd ((e' \wedge e_1)|e_2|\cdots|e_n)) \wedge (\neg c \rhd (e'|e_2|\cdots|e_n)) \quad (2.13)$$

Table 2.2: Equivalences on nondeterministic effects

in normal form is

$$((a \rhd b)|((a \rhd c) \wedge (a \rhd f))) \wedge (((\top \rhd d) \wedge (\top \rhd e))|(b \rhd e)).$$

■

For some applications a still simpler form of operators is useful. In the second normal form for nondeterministic operators nondeterminism may appear only at the outermost structure in the effect.

Definition 2.19 (Normal form II for nondeterministic operators) *A deterministic effect is in normal formal II if it is $\top$ or a conjunction of one or more effects $c \rhd a$ and $c \rhd \neg a$ with at most one occurrence of a and $\neg a$ for any $a \in A$.*

A nondeterministic effect is in normal form II if it is of form $e_1|\cdots|e_n$ where e_i are deterministic effects in normal form II.

A nondeterministic operator $\langle c, e\rangle$ is in normal form II if e is in normal form II.

Theorem 2.20 *For every operator there is an equivalent one in normal form II.*

Proof: By Theorem 2.17 there is an equivalent operator in normal form. The transformation further into normal form II requires equivalences 2.11 and 2.12. □

2.4 Computational complexity

In this section we discuss deterministic, nondeterministic and alternating Turing machines (DTMs, NDTMs and ATMs) and define several complexity classes in terms of them. For a detailed introduction to computational complexity see any of the standard textbooks [Balcázar *et al.*, 1988; 1990; Papadimitriou, 1994].

The definition of ATMs we use is like that of Balcázar et al. [1990] but without a separate input tape. Deterministic and nondeterministic Turing machines (DTMs, NDTMs) are a special case of alternating Turing machines.

Definition 2.21 *An* alternating Turing machine *is a tuple $\langle \Sigma, Q, \delta, q_0, g\rangle$ where*

- *Σ is a finite alphabet (the contents of tape cells),*
- *Q is a finite set of states (the internal states of the ATM),*

- *δ is a transition function* $\delta : Q \times (\Sigma \cup \{|, \square\}) \to 2^{(\Sigma \cup \{|\}) \times Q \times \{L,N,R\}}$,
- *q_0 is the initial state, and*
- $g : Q \to \{\forall, \exists, \textit{accept}, \textit{reject}\}$ *is a labeling of the states.*

The symbols | and $\square$, the end-of-tape symbol and the blank symbol, in the definition of δ respectively refer to the beginning of the tape and to the end of the tape. It is required that $s = |$ and $m = R$ for all $\langle s, q', m\rangle \in \delta(q, |)$ for any $q \in Q$, that is, at the left end of the tape the movement is always to the right and the end-of-tape symbol | may not be changed. For $s \in \Sigma$ we restrict s' in $\langle s', q', m\rangle \in \delta(q, s)$ to $s' \in \Sigma$, that is, | gets written onto the tape only in the special case when the R/W head is on the end-of-tape symbol. Note that the transition function is a total function, and the ATM computation terminated upon reaching an accepting or a rejecting state.

A configuration of an ATM is $\langle q, \sigma, \sigma'\rangle$ where q is the current state, σ is the tape contents left of the R/W head with the rightmost symbol under the R/W head, and σ' is the tape contents strictly right of the R/W head. This is a finite representation of the finite non-blank segment of the tape of the ATM. The configuration is universal ($\forall$) if $g(q) = \forall$, and existential ($\exists$) if $g(q) = \exists$.

The computation of an ATM starts from the initial configuration $\langle q_0, |a, \sigma\rangle$ where $a\sigma$ is the input string of the Turing machine. Below ϵ denotes the empty string.

Successor configurations are defined as follows.

1. A successor of $\langle q, \sigma a, \sigma'\rangle$ is $\langle q', \sigma, a'\sigma'\rangle$ if $\langle a', q', L\rangle \in \delta(q, a)$.
2. A successor of $\langle q, \sigma a, \sigma'\rangle$ is $\langle q', \sigma a', \sigma'\rangle$ if $\langle a', q', N\rangle \in \delta(q, a)$.
3. A successor of $\langle q, \sigma a, b\sigma'\rangle$ is $\langle q', \sigma a' b, \sigma'\rangle$ if $\langle a', q', R\rangle \in \delta(q, a)$.
4. A successor of $\langle q, \sigma a, \epsilon\rangle$ is $\langle q', \sigma a'\square, \epsilon\rangle$ if $\langle a', q', R\rangle \in \delta(q, a)$.

We write $\langle q, \sigma\rangle \vdash \langle q', \sigma'\rangle$ if the latter is a successor configuration of the former. A configuration $\langle q, \sigma, \sigma'\rangle$ of an ATM is *final* if $g(q) =$ accept or $g(q) =$ reject.

The acceptance of an input string by an ATM is defined recursively starting from final configurations. A final configuration is 0-accepting if $g(q) =$ accept. A non-final universal configuration is n-accepting for $n \geq 1$ if its every successor configuration is m-accepting for some $m < n$ and one of its successor configurations is $n-1$-accepting. A non-final existential configuration is n-accepting for $n \geq 1$ if at least one of its successor configurations is $n-1$-accepting and it has no m-accepting successor configurations for any $m < n-1$. Finally, an ATM accepts a given input string if its initial configuration is n-accepting for some $n \geq 0$. A configuration is *accepting* if it is n-accepting for some $n \geq 0$.

If an ATM accepts a given input string, then we can define *an accepting computation subtree* of the ATM and the input string as a set T of accepting configurations such that

1. the initial configuration is in T,
2. if $c \in T$ is a $\forall$-configuration then $c' \in T$ for all configurations c' such that $c \vdash c'$,
3. if $c \in T$ is an n-accepting $\exists$-configuration then $c' \in T$ for at least one c' such that $c \vdash c'$ and c' is m-accepting for some $m < n$.

A nondeterministic Turing machine is an ATM without universal states. A deterministic Turing machine is an ATM with $|\delta(q, s)| = 1$ for all $q \in Q$ and $s \in \Sigma$.

The complexity classes used in this work are the following. PSPACE is the class of decision problems solvable by deterministic Turing machines that use a number of tape cells bounded by a polynomial on the input length n. Formally,

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{DSPACE}(n^k).$$

Other complexity classes are similarly defined in terms of the time consumption on a deterministic Turing machine (DTIME($f(n)$), time consumption on a nondeterministic Turing machine (NTIME($f(n)$), or time or space consumption on alternating Turing machines (ATIME($f(n)$) or ASPACE($f(n)$)) [Balcázar *et al.*, 1988; 1990].

$$\begin{aligned}
\text{P} &= \textstyle\bigcup_{k \geq 0} \text{DTIME}(n^k) \\
\text{NP} &= \textstyle\bigcup_{k \geq 0} \text{NTIME}(n^k) \\
\text{EXP} &= \textstyle\bigcup_{k \geq 0} \text{DTIME}(2^{n^k}) \\
\text{NEXP} &= \textstyle\bigcup_{k \geq 0} \text{NTIME}(2^{n^k}) \\
\text{EXPSPACE} &= \textstyle\bigcup_{k \geq 0} \text{DSPACE}(2^{n^k}) \\
\text{2-EXP} &= \textstyle\bigcup_{k \geq 0} \text{DTIME}(2^{2^{n^k}}) \\
\text{2-NEXP} &= \textstyle\bigcup_{k \geq 0} \text{NTIME}(2^{2^{n^k}}) \\
\\
\text{APSPACE} &= \textstyle\bigcup_{k \geq 0} \text{ASPACE}(n^k) \\
\text{AEXPSPACE} &= \textstyle\bigcup_{k \geq 0} \text{ASPACE}(2^{n^k})
\end{aligned}$$

There are many useful connections between complexity classes defined in terms of deterministic and alternating Turing machines [Chandra *et al.*, 1981], for example

$$\begin{aligned}
\text{EXP} &= \text{APSPACE} \\
\text{2-EXP} &= \text{AEXPSPACE}.
\end{aligned}$$

Roughly, an exponential deterministic time bound corresponds to a polynomial alternating space bound.

We have defined all the complexity classes in terms of Turing machines. However, for all purposes of this work, we can equivalently use conventional programming languages (like C or Java) or simplified variants of them for describing computation. The main difference between conventional programming languages and Turing machines is that the former use random-access memory whereas memory access in Turing machines is local and only the current tape cell can be directly accessed. However, these two computational models can be simulated with each other with a polynomial overhead and are therefore for our purposes equivalent. The differences show up in complexity classes with very strict (subpolynomial) restrictions on time and space consumption.

Later in this work, the proofs of membership of a given computational problem in a certain complexity class are usually given in terms of a program in a simple programming language comparable to a small subset of C or Java, instead of giving a formal description of a Turing machine because the latter would usually be very complicated and difficult to understand.

A problem L is *C-hard* (where C is any of the complexity classes) if all problems in the class C are polynomial time *many-one reducible* to it; that is, for all problems $L' \in C$ there is a function

$f_{L'}$ that can be computed in polynomial time on the size of its input and $f_{L'}(x) \in L$ if and only if $x \in L'$ for all inputs x. We say that the function $f_{L'}$ is a translation from L' to L. A problem is *C-complete* if it belongs to the class C and is C-hard.

In complexity theory the most important distinction between computational problems is that between *tractable* and *intractable* problems. A problem is considered to be tractable, efficiently solvable, if it can be solved in polynomial time. Otherwise it is intractable. Most planning problems are highly intractable, but for many algorithmic approaches to planning it is important that certain basic steps in these algorithms can be guaranteed to be tractable.

In this work we analyze the complexity of many computational problems, showing them to be complete problems for some of the classes mentioned above. The proofs consist of two parts. We show that the problem belongs to the class. This is typically by giving an algorithm for the problem, possibly a nondeterministic one, and then showing that the algorithm obeys the resource bounds on time or memory consumption as required by the complexity class. Then we show the hardness of the problem for the class, that is, we can reduce any problem in the class to the problem in polynomial time. This can be either by simulating all Turing machines that represent computation in the class, or by reducing a complete problem in the class to the problem in question in polynomial time (a many-one reduction).

For almost all commonly used complexity classes there are more or less natural complete problems that often have a central role in proving the completeness of other problems for the class in question. Some complete problems for the complexity classes mentioned above are the following.[1]

class	complete problem
P	truth-value of formulae in the propositional logic in a given valuation
NP	satisfiability of formulae in the propositional logic (SAT)
PSPACE	truth-value of quantified Boolean formulae

Complete problems for classes like EXP and NEXP can be obtained from the P-complete and NP-problems by representing propositional formulae succinctly in terms of other propositional formulae [Papadimitriou and Yannakakis, 1986].

[1] For definition of P-hard problems we have to use more restricted many-one reductions that use only logarithmic space instead of polynomial time. Otherwise all non-trivial problems in P would be P-hard and P-complete.

Chapter 3

Deterministic planning

In this chapter we describe a number of algorithms for solving the historically most important and most basic type of planning problem. Two rather strong simplifying assumptions are made. First, all actions are deterministic which means that under every action every state has at most one successor state. Second, there is only one initial state.

Under these restrictions, whenever a goal state can be reached, it can be reached by a fixed sequence of actions. With more than one initial state it would be necessary to use different sequences of actions for different initial states, and with nondeterministic actions the sequence of actions to be taken is not simply a function of the initial state, and for producing appropriate sequences of actions a more general notion of plans with branches/conditionals becomes necessary. This is because after executing an action, even when the starting state is known, the state that is reached cannot be predicted, and the way plan execution continues depends on the newly reached state. In Chapters 4 and 5 we relax both of these restrictions, and consider planning with nondeterministic actions and more than one initial state.

The structure of this chapter is as follows. First we discuss the two ways of traversing the transition system without producing the graph explicitly. In forward traversal we repeatedly compute the successor states of our current state, starting from the initial state. In backward traversal we must use sets of states, represented as formulae, because there may be several goal states, and further, a given state may have several predecessor states under one action.

Then in Section 3.3 we discuss the use of heuristic search algorithms for performing the search in the transition graphs and the computation of distance heuristics to be used in estimating the value of the current states or sets of states.

A very different approach to planning is obtained by translating the planning problem into the classical propositional logic and then finding plans by algorithms that test the satisfiability of formulae in the propositional logic. This approach is called planning by satisfiability. In Section 3.7 we perform a detailed analysis of different notions of plans and in Section 3.8 we present efficient translations of these plan notions into the propositional logic. In Section 3.9 we consider the problem of plan search for planning as satisfiability and propose two new efficient algorithms for finding plans when no optimality guarantees are needed.

3.1 State-space search

The simplest possible planning algorithm generates all states (valuations of the state variables), constructs the transition graph, and then finds a path from the initial state I to a goal state $g \in G$ for example by a shortest-path algorithm. The plan is then simply the sequence of operators corresponding to the edges on the shortest path from the initial state to a goal state. However, this algorithm is not feasible when the number of state variables is higher than 20 or 30 because the number of valuations is very high: $2^{20} = 1048576 \sim 10^6$ for 20 Boolean state variables and $2^{30} = 1073741824 \sim 10^9$ for 30.

Instead, it will often be much more efficient to avoid generating most of the state space explicitly and to produce only the successor or predecessor states of the states currently under consideration. This form of plan search can be easiest viewed as the application of general-purpose search algorithms that can be employed in solving a wide range of search problems. The best known *heuristic search algorithms* are A$*$, IDA$*$ and their variants [Hart *et al.*, 1968; Pearl, 1984; Korf, 1985] which can be used in finding shortest plans or plans that are guaranteed to be close to the shortest ones.

There are two main possibilities to find a path from the initial state to a goal state: traverse the transition graph forwards starting from the initial state, or traverse it backwards starting from the goal states. The main difference between these possibilities is that there may be several goal states (and one state may have several predecessor states with respect to one operator) but only one initial state: in forward traversal we repeatedly compute the unique successor state of the current state, whereas with backward traversal we are forced to keep track of a possibly very high number of possible predecessor states of the goal states. Backward search is slightly more complicated to implement but it allows to simultaneously consider several paths leading to a goal state.

3.1.1 Progression and forward search

We have already defined *progression* for single states s as $app_o(s)$. The simplest algorithm for the deterministic planning problem does not require the explicit representation of the whole transition graph. The search starts in the initial state. New states are generated by progression. As soon as a state s such that $s \models G$ is found a plan is guaranteed to exist: it is the sequence of operators with which the state s is reached from the initial state.

A planner can use progression in connection with any of the standard search algorithms. Later in this chapter we will discuss how heuristic search algorithms together with heuristics yield an efficient planning method.

3.1.2 Regression and backward search

With backward search the starting point is a propositional formula G that describes the set of goal states. An operator is selected, the set of possible predecessor states is computed, and this set is again described by a propositional formula. A plan has been found when a formula that is true in the initial state is reached. The computation of a formula representing the predecessor states of the states represented by another formula is called *regression*. Regression is more powerful than progression because it allows handling potentially very big sets of states, but it is also more expensive.

Definition 3.1 *We define the condition $EPC_l(e)$ of literal l made true when an operator with the effect e is applied recursively as follows.*

$$\begin{aligned} EPC_l(\top) &= \bot \\ EPC_l(l) &= \top \\ EPC_l(l') &= \bot \text{ when } l \neq l' \text{ (for literals } l') \\ EPC_l(e_1 \wedge \cdots \wedge e_n) &= EPC_l(e_1) \vee \cdots \vee EPC_l(e_n) \\ EPC_l(c \rhd e) &= c \wedge EPC_l(e) \end{aligned}$$

The case $EPC_l(e_1 \wedge \cdots \wedge e_n) = EPC_l(e_1) \vee \cdots \vee EPC_l(e_n)$ is defined as a disjunction because it is sufficient that at least one of the effects makes l true.

Definition 3.2 *Let A be the set of state variables. We define the condition $EPC_l(o)$ of operator $o = \langle c, e \rangle$ being applicable so that literal l is made true as $c \wedge EPC_l(e) \wedge \bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$.*

For effects e the truth-value of the formula $EPC_l(e)$ indicates in which states l is a literal to which the effect e assigns the value true. The connection to the earlier definition of $[e]_s^{det}$ is stated in the following lemma.

Lemma 3.3 *Let A be the set of state variables, s a state on A, l a literal on A, and o and operator with effect e. Then*

1. *$l \in [e]_s^{det}$ if and only if $s \models EPC_l(e)$, and*
2. *$app_o(s)$ is defined and $l \in [e]_s^{det}$ if and only if $s \models EPC_l(o)$.*

Proof: We first prove (1) by induction on the structure of the effect e.

Base case 1, $e = \top$: By definition of $[\top]_s^{det}$ we have $l \notin [\top]_s^{det} = \emptyset$, and by definition of $EPC_l(\top)$ we have $s \not\models EPC_l(\top) = \bot$, so the equivalence holds.

Base case 2, $e = l$: $l \in [l]_s^{det} = \{l\}$ by definition, and $s \models EPC_l(l) = \top$ by definition.

Base case 3, $e = l'$ for some literal $l' \neq l$: $l \notin [l']_s^{det} = \{l'\}$ by definition, and $s \not\models EPC_l(l') = \bot$ by definition.

Inductive case 1, $e = e_1 \wedge \cdots \wedge e_n$:

$$\begin{aligned} l \in [e]_s^{det} \quad &\text{if and only if} \quad l \in [e']_s^{det} \text{ for some } e' \in \{e_1, \ldots, e_n\} \\ &\text{if and only if} \quad s \models EPC_l(e') \text{ for some } e' \in \{e_1, \ldots, e_n\} \\ &\text{if and only if} \quad s \models EPC_l(e_1) \vee \cdots \vee EPC_l(e_n) \\ &\text{if and only if} \quad s \models EPC_l(e_1 \wedge \cdots \wedge e_n). \end{aligned}$$

The second equivalence is by the induction hypothesis, the other equivalences are by the definitions of $EPC_l(e)$ and $[e]_s^{det}$ as well as elementary facts about propositional formulae.

Inductive case 2, $e = c \rhd e'$:

$$\begin{aligned} l \in [c \rhd e']_s^{det} \quad &\text{if and only if} \quad l \in [e']_s^{det} \text{ and } s \models c \\ &\text{if and only if} \quad s \models EPC_l(e') \text{ and } s \models c \\ &\text{if and only if} \quad s \models EPC_l(c \rhd e'). \end{aligned}$$

The second equivalence is by the induction hypothesis. This completes the proof of (1).

(2) follows from the fact that the conjuncts c and $\bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$ in $EPC_l(o)$ exactly state the applicability conditions of o. □

Note that any operator $\langle c, e\rangle$ can be expressed in normal form in terms of $\mathit{EPC}_a(e)$ as

$$\left\langle c, \bigwedge_{a\in A} (\mathit{EPC}_a(e) \rhd a) \wedge (\mathit{EPC}_{\neg a}(e) \rhd \neg a) \right\rangle.$$

The formula $\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$ expresses the condition for the truth $a \in A$ after the effect e is executed in terms of truth-values of state variables before: either a becomes true, or a is true before and does not become false.

Lemma 3.4 *Let $a \in A$ be a state variable, $o = \langle c, e\rangle \in O$ an operator, and s and $s' = app_o(s)$ states. Then $s \models \mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$ if and only if $s' \models a$.*

Proof: Assume that $s \models \mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$. We perform a case analysis and show that $s' \models a$ holds in both cases.

Case 1: Assume that $s \models \mathit{EPC}_a(e)$. By Lemma 3.3 $a \in [e]_s^{det}$, and hence $s' \models a$.

Case 2: Assume that $s \models a \wedge \neg \mathit{EPC}_{\neg a}(e)$. By Lemma 3.3 $\neg a \notin [e]_s^{det}$. Hence a is true in s'.

For the other half of the equivalence, assume that $s \not\models \mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$. Hence $s \models \neg \mathit{EPC}_a(e) \wedge (\neg a \vee \mathit{EPC}_{\neg a}(e))$.

Case 1: Assume that $s \models a$. Now $s \models \mathit{EPC}_{\neg a}(e)$ because $s \models \neg a \vee \mathit{EPC}_{\neg a}(e)$, and hence by Lemma 3.3 $\neg a \in [e]_s^{det}$ and hence $s' \not\models a$.

Case 2: Assume that $s \not\models a$. Since $s \models \neg \mathit{EPC}_a(e)$, by Lemma 3.3 $a \notin [e]_s^{det}$ and hence $s' \not\models a$.

Therefore $s' \not\models a$ in all cases. □

The formulae $\mathit{EPC}_l(e)$ can be used in defining regression.

Definition 3.5 (Regression) *Let ϕ be a propositional formula and $o = \langle c, e\rangle$ an operator. The* regression *of ϕ with respect to o is $\mathit{regr}_o(\phi) = \phi_r \wedge c \wedge \chi$ where $\chi = \bigwedge_{a\in A} \neg(\mathit{EPC}_a(e) \wedge \mathit{EPC}_{\neg a}(e))$ and ϕ_r is obtained from ϕ by replacing every $a \in A$ by $\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$. Define $\mathit{regr}_e(\phi) = \phi_r \wedge \chi$ and use the notation $\mathit{regr}_{o_1;\ldots;o_n}(\phi) = \mathit{regr}_{o_1}(\cdots \mathit{regr}_{o_n}(\phi)\cdots)$.*

The conjuncts of χ say that none of the state variables may simultaneously become true and false. The operator is not applicable in states in which χ is false.

Remark 3.6 *Regression can be equivalently defined in terms of the conditions the state variables stay or become false, that is, we could use the formula $\mathit{EPC}_{\neg a}(e) \vee (\neg a \wedge \neg \mathit{EPC}_a(e))$ which tells when a is false. The negation of this formula, which can be written as $(\mathit{EPC}_a(e) \wedge \neg \mathit{EPC}_{\neg a}(e)) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$, is not equivalent to $\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$. However, if $\mathit{EPC}_a(e)$ and $\mathit{EPC}_{\neg a}(e)$ are not simultaneously true, we do get equivalence, that is,*

$$\begin{aligned} \neg(\mathit{EPC}_a(e) \wedge \mathit{EPC}_{\neg a}(e)) \quad \models \quad & ((\mathit{EPC}_a(e) \wedge \neg \mathit{EPC}_{\neg a}(e)) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))) \\ & \leftrightarrow (\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))) \end{aligned}$$

because $\neg(\mathit{EPC}_a(e) \wedge \mathit{EPC}_{\neg a}(e)) \models (\mathit{EPC}_a(e) \wedge \neg \mathit{EPC}_{\neg a}(e)) \leftrightarrow \mathit{EPC}_a(e)$.

An upper bound on the size of the formula obtained by regression with operators $o_1, \ldots, o_n$ starting from ϕ is the product of the sizes of $\phi, o_1, \ldots, o_n$, which is exponential in n. However, the formulae can often be simplified because there are many occurrences of $\top$ and $\bot$, for example by using the equivalences $\top \wedge \phi \equiv \phi$, $\bot \wedge \phi \equiv \bot$, $\top \vee \phi \equiv \top$, $\bot \vee \phi \equiv \phi$, $\neg\bot \equiv \top$, and $\neg\top \equiv \bot$.

For unconditional operators $o_1, \ldots, o_n$ (with no occurrences of $\rhd$), an upper bound on the size of the formula (after eliminating $\top$ and $\bot$) is the sum of the sizes of $o_1, \ldots, o_n$ and ϕ.

The reason why regression is useful for planning is that it allows to compute the predecessor states by simple formula manipulation. The same does not seem to be possible for progression because there is no known simple definition of successor states of a *set* of states expressed in terms of a formula: simple syntactic progression is restricted to individual states only (see Section 5.2 for a general but expensive definition of progression for arbitrary formulae.)

The important property of regression is formalized in the following lemma.

Theorem 3.7 *Let ϕ be a formula over A, o an operator over A, and S the set of all states i.e. valuations of A. Then $\{s \in S | s \models \mathit{regr}_o(\phi)\} = \{s \in S | \mathit{app}_o(s) \models \phi\}$.*

Proof: We show that for any state s, $s \models \mathit{regr}_o(\phi)$ if and only if $\mathit{app}_o(s)$ is defined and $\mathit{app}_o(s) \models \phi$. By definition $\mathit{regr}_o(\phi) = \phi_r \wedge c \wedge \chi$ for $o = \langle c, e \rangle$ where ϕ_r is obtained from ϕ by replacing every state variable $a \in A$ by $\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$ and $\chi = \bigwedge_{a \in A} \neg(\mathit{EPC}_a(e) \wedge \mathit{EPC}_{\neg a}(e))$.

First we show that $s \models c \wedge \chi$ if and only if $\mathit{app}_o(s)$ is defined.

$s \models c \wedge \chi$ iff $s \models c$ and $\{a, \neg a\} \not\subseteq [e]_s^{det}$ for all $a \in A$ by Lemma 3.3
iff $\mathit{app}_o(s)$ is defined by Definition 2.10.

Then we show that $s \models \phi_r$ if and only if $\mathit{app}_o(s) \models \phi$. This is by structural induction over subformulae ϕ' of ϕ and formulae ϕ'_r obtained from ϕ' by replacing $a \in A$ by $\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$

Induction hypothesis: $s \models \phi'_r$ if and only if $\mathit{app}_o(s) \models \phi'$.

Base case 1, $\phi' = \top$: Now $\phi'_r = \top$ and both are true in the respective states.

Base case 2, $\phi' = \bot$: Now $\phi'_r = \bot$ and both are false in the respective states.

Base case 3, $\phi' = a$ for some $a \in A$: Now $\phi'_r = \mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$. By Lemma 3.4 $s \models \phi'_r$ if and only if $\mathit{app}_o(s) \models \phi'$.

Inductive case 1, $\phi' = \neg\theta$: By the induction hypothesis $s \models \theta_r$ iff $\mathit{app}_o(s) \models \theta$. Hence $s \models \phi'_r$ iff $\mathit{app}_o(s) \models \phi'$ by the truth-definition of $\neg$.

Inductive case 2, $\phi' = \theta \vee \theta'$: By the induction hypothesis $s \models \theta_r$ iff $\mathit{app}_o(s) \models \theta$, and $s \models \theta'_r$ iff $\mathit{app}_o(s) \models \theta'$. Hence $s \models \phi'_r$ iff $\mathit{app}_o(s) \models \phi'$ by the truth-definition of $\vee$.

Inductive case 3, $\phi' = \theta \wedge \theta'$: By the induction hypothesis $s \models \theta_r$ iff $\mathit{app}_o(s) \models \theta$, and $s \models \theta'_r$ iff $\mathit{app}_o(s) \models \theta'$. Hence $s \models \phi'_r$ iff $\mathit{app}_o(s) \models \phi'$ by the truth-definition of $\wedge$. □

Regression can be performed with any operator but not all applications of regression are useful. First, regressing for example the formula a with the effect $\neg a$ is not useful because the new unsatisfiable formula describes the empty set of states. Hence the sequence of operators of the previous regressions steps do not lead to a goal from any state. Second, regressing a with the operator $\langle b, c \rangle$ yields $\mathit{regr}_{\langle b,c \rangle}(a) = a \wedge b$. Finding a plan for reaching a state satisfying a is easier than finding a plan for reaching a state satisfying $a \wedge b$. Hence the regression step produced a subproblem that is more difficult than the original problem, and it would therefore be better not to take this regression step.

Lemma 3.8 *Let there be a plan $o_1, \ldots, o_n$ for $\langle A, I, O, G \rangle$. If $\mathit{regr}_{o_k;\ldots;o_n}(G) \models \mathit{regr}_{o_{k+1};\ldots;o_n}(G)$ for some $k \in \{1, \ldots, n-1\}$, then also $o_1, \ldots, o_{k-1}, o_{k+1}, \ldots, o_n$ is a plan for $\langle A, I, O, G \rangle$.*

Proof: By Theorem 3.7 $\mathit{app}_{o_{k+1};\ldots;o_n}(s) \models G$ for any s such that $s \models \mathit{regr}_{o_{k+1};\ldots;o_n}(G)$. Since $\mathit{app}_{o_1;\ldots;o_{k-1}}(I) \models \mathit{regr}_{o_k;\ldots;o_n}(G)$ and $\mathit{regr}_{o_k;\ldots;o_n}(G) \models \mathit{regr}_{o_{k+1};\ldots;o_n}(G)$ also $\mathit{app}_{o_1;\ldots;o_{k-1}}(I) \models$

$regr_{o_{k+1};\ldots;o_n}(G)$. Hence $app_{o_1;\ldots;o_{k-1};o_{k+1};\ldots;o_n}(I) \models G$ and $o_1;\ldots;o_{k-1};o_{k+1};\ldots;o_n$ is a plan for $\langle A, I, O, G\rangle$. □

Therefore any regression step that makes the set of states smaller in the set-inclusion sense is unnecessary. However, testing whether this is the case may be computationally expensive. Although the following two problems are closely related to SAT, it could be possible that the formulae obtained by reduction to SAT would fall in some polynomial-time subclass. We show that this is not the case.

Lemma 3.9 *The problem of testing whether $regr_o(\phi) \not\models \phi$ is NP-hard.*

Proof: We give a reduction from SAT to the problem. Let ϕ be any formula. Let a be a state variable not occurring in ϕ. Now $regr_{\langle\neg\phi\rightarrow a,a\rangle}(a) \not\models a$ if and only if $(\neg\phi \rightarrow a) \not\models a$, because $regr_{\langle\neg\phi\rightarrow a,a\rangle}(a) = \neg\phi \rightarrow a$. $(\neg\phi \rightarrow a) \not\models a$ is equivalent to $\not\models (\neg\phi \rightarrow a) \rightarrow a$ that is equivalent to the satisfiability of $\neg((\neg\phi \rightarrow a) \rightarrow a)$. Further, $\neg((\neg\phi \rightarrow a) \rightarrow a)$ is logically equivalent to $\neg(\neg(\phi \vee a) \vee a)$ and further to $\neg(\neg\phi \vee a)$ and $\phi \wedge \neg a$.

Satisfiability of $\phi \wedge \neg a$ is equivalent to the satisfiability of ϕ as a does not occur in ϕ: if ϕ is satisfiable, there is a valuation v such that $v \models \phi$, we can set a false in v to obtain v', and as a does not occur in ϕ, we still have $v' \models \phi$, and further $v' \models \phi \wedge \neg a$. Clearly, if ϕ is unsatisfiable also $\phi \wedge \neg a$ is.

Hence $regr_{\langle\neg\phi\rightarrow a,a\rangle}(a) \not\models a$ if and only if ϕ is satisfiable. □

Also the problem of testing whether a regression step leads to an empty set of states is difficult.

Lemma 3.10 *The problem of testing that $regr_o(\phi)$ is satisfiable is NP-hard.*

Proof: Proof is a reduction from SAT. Let ϕ be a formula. $regr_{\langle\phi,a\rangle}(a)$ is satisfiable if and only if ϕ is satisfiable because $regr_{\langle\phi,a\rangle}(a) \equiv \phi$.

The problem is NP-hard even if we restrict to operators that have a satisfiable precondition: ϕ is satisfiable if and only if $(\phi \vee \neg a) \wedge a$ is satisfiable if and only if $regr_{\langle\phi\vee\neg a,b\rangle}(a \wedge b)$ is satisfiable. Here a is a state variable that does not occur in ϕ. Clearly, $\phi \vee \neg a$ is true when a is false, and hence $\phi \vee \neg a$ is satisfiable. □

Of course, testing that $regr_o(\phi) \not\models \phi$ or that $regr_o(\phi)$ is satisfiable is not necessary for the correctness of backward search, but avoiding useless steps improves efficiency.

Early work on planning restricted to goals and operator preconditions that are conjunctions of state variables and to unconditional effects (STRIPS operators with only positive literals in preconditions.) In this special case both goals G and operator effects e can be viewed as sets of literals, and the definition of regression is particularly simple: regressing G with respect to $\langle c, e\rangle$ is $(G \backslash e) \cup c$. If there is $a \in A$ such that $a \in G$ and $\neg a \in e$, then the result of regression is $\bot$, that is, the empty set of states. We do not use this restricted type of regression in this work.

Some planners that use backward search and have operators with disjunctive preconditions and conditional effects eliminate all disjunctivity by branching. For example, the backward step from g with operator $\langle a \vee b, g\rangle$ yields $a \vee b$. This formula corresponds to two non-disjunctive goals, a and b. For each of these new goals a separate subtree is produced. Disjunctivity caused by conditional effects can similarly be handled by branching. However, this branching may lead to a very high branching factor and thus to poor performance.

In addition to being the basis of backward search, regression has many other applications in reasoning about actions. One of them is the composition of operators. The composition $o_1 \circ o_2$ of operators $o_1 = \langle c_1, e_1 \rangle$ and $o_2 = \langle c_2, e_2 \rangle$ is an operator that behaves like applying o_1 followed by o_2. For a to be true after o_2 we can regress a with respect to o_2, obtaining $EPC_a(e_2) \vee (a \wedge \neg EPC_{\neg a}(e_2))$. Condition for this formula to be true after o_1 is obtained by regressing with e_1, leading to

$$\begin{aligned}
&regr_{e_1}(EPC_a(e_2) \vee (a \wedge \neg EPC_{\neg a}(e_2))) \\
&= regr_{e_1}(EPC_a(e_2)) \vee (regr_{e_1}(a) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2))) \\
&= regr_{e_1}(EPC_a(e_2)) \vee ((EPC_a(e_1) \vee (a \wedge \neg EPC_{\neg a}(e_2))) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2))).
\end{aligned}$$

Since we want to define an effect $\phi \rhd a$ of $o_1 \circ o_2$ so that a becomes true whenever o_1 followed by o_2 would make it true, the formula ϕ does not have to represent the case in which a is true already before the application of $o_1 \circ o_2$. Hence we can simplify the above formula to

$$regr_{e_1}(EPC_a(e_2)) \vee (EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2))).$$

An analogous formula is needed for making $\neg a$ false. This leads to the following definition.

Definition 3.11 (Composition of operators) *Let $o_1 = \langle c_1, e_1 \rangle$ and $o_2 = \langle c_2, e_2 \rangle$ be two operators on A. Then their* composition $o_1 \circ o_2$ *is defined as*

$$\left\langle c, \bigwedge_{a \in A} \left(\begin{array}{l} ((regr_{e_1}(EPC_a(e_2)) \vee (EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2)))) \rhd a) \wedge \\ ((regr_{e_1}(EPC_{\neg a}(e_2)) \vee (EPC_{\neg a}(e_1) \wedge \neg regr_{e_1}(EPC_a(e_2)))) \rhd \neg a) \end{array} \right) \right\rangle$$

where $c = c_1 \wedge regr_{e_1}(c_2) \wedge \bigwedge_{a \in A} \neg (EPC_a(e_1) \wedge EPC_{\neg a}(e_1))$.

Note that in $o_1 \circ o_2$ first o_1 is applied and then o_2, so the ordering is opposite to the usual notation for the composition of functions.

Theorem 3.12 *Let o_1 and o_2 be operators and s a state. Then $app_{o_1 \circ o_2}(s)$ is defined if and only if $app_{o_1;o_2}(s)$ is defined, and $app_{o_1 \circ o_2}(s) = app_{o_1;o_2}(s)$.*

Proof: Let $o_1 = \langle c_1, e_1 \rangle$ and $o_2 = \langle c_2, e_2 \rangle$. Assume $app_{o_1 \circ o_2}(s)$ is defined. Hence $s \models c_1 \wedge regr_{e_1}(c_2) \wedge \bigwedge_{a \in A} \neg (EPC_a(e_1) \wedge EPC_{\neg a}(e_1))$, that is, the precondition of $o_1 \circ o_2$ is true, and $s \not\models (regr_{e_1}(EPC_a(e_2)) \vee (EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2)))) \wedge (((regr_{e_1}(EPC_{\neg a}(e_2)) \vee (EPC_{\neg a}(e_1) \wedge \neg regr_{e_1}(EPC_a(e_2))))))$ for all $a \in A$, that is, the effects do not contradict each other.

Now $app_{o_1}(s)$ in $app_{o_1;o_2}(s) = app_{o_2}(app_{o_1}(s))$ defined because $s \models c_1 \wedge \bigwedge_{a \in A} \neg (EPC_a(e_1) \wedge EPC_{\neg a}(e_1))$. Further $app_{o_1}(s) \models c_2$ by Theorem 3.7 because $s \models regr_{e_1}(c_2)$. From $s \not\models (regr_{e_1}(EPC_a(e_2)) \vee (EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2)))) \wedge (((regr_{e_1}(EPC_{\neg a}(e_2)) \vee (EPC_{\neg a}(e_1) \wedge \neg regr_{e_1}(EPC_a(e_2))))))$ for all $a \in A$ logically follows $s \not\models regr_{e_1}(EPC_a(e_2)) \wedge regr_{e_1}(EPC_{\neg a}(e_2))$ for all $a \in A$. Hence by Theorem 3.7 $app_{o_1}(s) \not\models EPC_a(e_2) \wedge EPC_{\neg a}(e_2)$ for all $a \in A$, and by Lemma 3.3 $app_{o_2}(app_{o_1}(s))$ is defined.

For the other direction, since $app_{o_1}(s)$ is defined, $s \models c_1 \wedge \bigwedge_{a \in A} \neg (EPC_a(e_1) \wedge EPC_{\neg a}(e_1))$. Since $app_{o_2}(app_{o_1}(s))$ is defined, $s \models regr_{e_1}(c_2)$ by Theorem 3.7.

It remains to show that the effects of $o_1 \circ o_2$ do not contradict. Since $app_{o_2}(app_{o_1}(s))$ is defined $app_{o_1}(s) \not\models EPC_a(e_2) \wedge EPC_{\neg a}(e_2)$ and $s \not\models EPC_a(e_1) \wedge EPC_{\neg a}(e_1)$ for all $a \in A$. Hence by Theorem 3.7 $s \not\models regr_{e_1}(EPC_a(e_2)) \wedge regr_{e_1}(EPC_{\neg a}(e_2))$ for all $a \in A$. Assume that for

some $a \in A$ $s \models \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)) \vee (\mathit{EPC}_a(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)))$, that is, $a \in [o_1 \circ o_2]^{det}_s$. If $s \models \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2))$ then $s \not\models \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)) \vee \neg \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2))$. Otherwise $s \models \mathit{EPC}_a(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2))$ and hence $s \not\models \mathit{EPC}_{\neg a}(e_1)$. Hence in both cases $s \not\models \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)) \vee (\mathit{EPC}_{\neg a}(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)))$, that is, $\neg a \notin [o_1 \circ o_2]^{det}_s$. Therefore $\mathit{app}_{o_1 \circ o_2}(s)$ is defined.

We show that for any $a \in A$, $\mathit{app}_{o_1 \circ o_2}(s) \models a$ if and only if $\mathit{app}_{o_1}(\mathit{app}_{o_2}(s)) \models a$. Assume $\mathit{app}_{o_1 \circ o_2}(s) \models a$. Hence one of two cases hold.

1. Assume $s \models \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)) \vee (\mathit{EPC}_a(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)))$.

   If $s \models \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2))$ then by Theorem 3.7 and Lemma 3.3 $a \in [e_1]^{det}_{\mathit{app}_{o_1}(s)}$. Hence $\mathit{app}_{o_1;o_2}(s) \models a$.

   Assume $s \models \mathit{EPC}_a(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2))$. Hence by Lemma 3.3 $a \in [e_1]^{det}_s$ and $\mathit{app}_{o_1}(s) \models a$, and $\mathit{app}_{o_1}(s) \not\models \mathit{EPC}_{\neg a}(e_2)$ and $\neg a \notin [e_2]^{det}_{\mathit{app}_{o_1}(s)}$. Hence $\mathit{app}_{o_1;o_2}(s) \models a$.

2. Assume $s \models a$ and $s \not\models \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)) \vee (\mathit{EPC}_{\neg a}(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)))$.

   Since $s \not\models \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2))$ by Theorem 3.7 $\mathit{app}_{o_1}(s) \not\models \mathit{EPC}_{\neg a}(e_2)$ and hence $\neg a \notin [e_2]^{det}_{\mathit{app}_{o_1}(s)}$.

   Since $s \not\models \mathit{EPC}_{\neg a}(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2))$ by Lemma 3.3 $\neg a \notin [e_1]^{det}_s$ or $\mathit{app}_{e_1}(s) \models \mathit{EPC}_a(e_2)$ and hence by Theorem 3.7 $a \in [e_2]^{det}_{\mathit{app}_{o_1}(s)}$.

Hence either o_1 does not make a false, or if it makes, makes o_2 it true again so that $\mathit{app}_{o_1;o_2}(s) \models a$ in all cases.

Assume $\mathit{app}_{o_1;o_2}(s) \models a$. Hence one of the following three cases must hold.

1. If $a \in [e_2]^{det}_{\mathit{app}_{o_1}(s)}$ then by Lemma 3.3 $\mathit{app}_{o_1}(s) \models \mathit{EPC}_a(e_2)$. By Theorem 3.7 $s \models \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2))$.

2. If $a \in [e_1]^{det}_s$ and $\neg a \notin [e_2]^{det}_{\mathit{app}_{o_1}(s)}$ then by Lemma 3.3 $\mathit{app}_{o_1}(s) \not\models \mathit{EPC}_{\neg a}(e_2)$. By Theorem 3.7 $s \models \mathit{EPC}_a(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2))$.

3. If $s \models a$ and $\neg a \notin [e_2]^{det}_{\mathit{app}_{o_1}(s)}$ and $\neg a \notin [e_1]^{det}_s$ then by Lemma 3.3 $\mathit{app}_{o_1}(s) \not\models \mathit{EPC}_{\neg a}(e_2)$. By Theorem 3.7 $s \not\models \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2))$.

   By Lemma 3.3 $s \not\models \mathit{EPC}_{\neg a}(e_1)$.

In the first two cases the antecedent of the first conditional in the definition of $o_1 \circ o_2$ is true, meaning that $\mathit{app}_{o_1 \circ o_2}(s) \models a$, and in the third case $s \models a$ and the antecedent of the second conditional effect is false, also meaning that $\mathit{app}_{o_1 \circ o_2}(s) \models a$. □

The above construction can be used to eliminate *sequential composition* from operator effects (Section 2.3.2).

3.2 Planning by heuristic search algorithms

Search for plans can be performed forwards or backwards respectively with progression or regression as described in Sections 3.1.1 and 3.1.2. There are several algorithms that can be used for

the purpose, including depth-first search, breadth-first search, and iterative deepening, but without informed selection of operators these algorithms perform poorly.

The use of additional information for guiding search is essential for achieving efficient planning with general-purpose search algorithms. Algorithms that use heuristic estimates on the values of the nodes in the search space for guiding the search have been applied to planning very successfully. Some of the more sophisticated search algorithms that can be used are A$*$ [Hart *et al.*, 1968], WA$*$ [Pearl, 1984], IDA$*$ [Korf, 1985], and simulated annealing [Kirkpatrick *et al.*, 1983].

The effectiveness of these algorithms is dependent on good heuristics for guiding the search. The most important heuristics are estimates of distances between states. The distance is the minimum number of operators needed for reaching a state from another state. In Section 3.4 we will present techniques for estimating the distances between states and sets of states. In this section we will discuss how heuristic search algorithms are applied in planning.

When search proceeds forwards by progression starting from the initial state, we estimate the distance between the current state and the set of goal states. When search proceeds backwards by regression starting from the goal states, we estimate the distance between the initial state and the current set of goal states as computed by regression.

All the systematic heuristic search algorithms can easily be implemented to keep track of the search history which for planning equals the sequence of operators in the incomplete plan under consideration. Therefore the algorithms are started from the initial state I (forward search) or from the goal formula G (backward search) and then proceed forwards with progression or backwards with regression. Whenever the search successfully finishes, the plan can be recovered from the data structures maintained by the algorithm.

Local search algorithms do not keep track of the search history, and we have to define the elements of the search space as prefixes or suffixes of plans. For forward search we use sequences of operators (prefixes of plans)

$$o_1; o_2; \ldots; o_n.$$

The search starts from the empty sequence. The neighbors of an incomplete plan are obtained by adding an operator to the end of the plan or by deleting some of the last operators.

Definition 3.13 (Neighbors for local search with progression) *Let $\langle A, I, O, G\rangle$ be a succinct transition system. For forward search, the neighbors of an incomplete plan $o_1; o_2; \ldots; o_n$ are the following.*

1. $o_1; o_2; \ldots; o_n; o$ *for any* $o \in O$ *such that* $app_{o_1;\ldots;o_n;o}(I)$ *is defined*
2. $o_1; o_2; \ldots; o_i$ *for any* $i < n$

When $app_{o_1;o_2;\ldots;o_n}(I) \models G$ then $o_1; \ldots; o_n$ is a plan.

Also for backward search the incomplete plans are sequence of operators (suffixes of plans)

$$o_n; \ldots; o_1.$$

The search starts from the empty sequence. The neighbors of an incomplete plan are obtained by adding an operator to the beginning of the plan or by deleting some of the first operators.

Definition 3.14 (Neighbors for local search with regression) *Let $\langle A, I, O, G\rangle$ be a succinct transition system. For backward search, the children of an incomplete plan $o_n; \ldots; o_1$ are the following.*

1. $o; o_n; \ldots; o_1$ *for any* $o \in O$ *such that* $regr_{o;o_n;\ldots;o_1}(G)$ *is defined*

2. $o_i; \ldots; o_1$ *for any* $i < n$

When $I \models regr_{o_n;\ldots;o_1}(G)$ then $o_n; \ldots; o_1$ is a plan.

Backward search and forward search are not the only possibilities to define planning as a search problem. In partial-order planning [McAllester and Rosenblitt, 1991] the search space consists of incomplete plans which are partially ordered multisets of operators. The neighbors of an incomplete plan are those obtained by adding an operator or an ordering constraint. Incomplete plans can also be formalized as fixed length sequences of operators in which zero or more of the operators are missing. This leads to the constraint-based approaches to planning, including the planning as satisfiability approach that is presented in Section 3.6.

3.3 Reachability

The notion of reachability is important in defining whether a planning problem is solvable and in deriving techniques that speed up search for plans.

3.3.1 Distances

First we define the distances between states in a transition system in which all operators are deterministic. Heuristics in Section 3.4 are approximations of distances.

Definition 3.15 *Let I be an initial state and O a set of operators. Define the* forward distance sets D_i^{fwd} *for I, O that consist of those states that are reachable from I by at most i operator applications as follows.*

$$\begin{array}{lcl} D_0^{fwd} & = & \{I\} \\ D_i^{fwd} & = & D_{i-1}^{fwd} \cup \{s | o \in O, s \in img_o(D_{i-1}^{fwd})\} \text{ for all } i \geq 1 \end{array}$$

Definition 3.16 *Let I be a state, O a set of operators, and $D_0^{fwd}, D_1^{fwd}, \ldots$ the forward distance sets for I, O. Then* the forward distance *of a state s from I is*

$$\delta_I^{fwd}(s) = \begin{cases} 0 \text{ if } s = I \\ i \text{ if } s \in D_i^{fwd} \backslash D_{i-1}^{fwd}. \end{cases}$$

If $s \notin D_i^{fwd}$ for all $i \geq 0$ then $\delta_I^{fwd}(s) = \infty$. States that have a finite forward distance are reachable *(from I with O).*

Distances can also be defined for formulae.

Definition 3.17 *Let ϕ be a formula. Then the* forward distance $\delta_I^{fwd}(\phi)$ *of ϕ is i if there is state s such that $s \models \phi$ and $\delta_I^{fwd}(s) = i$ and there is no state s' such that $s' \models \phi$ and $\delta_I^{fwd}(s) < i$. If $I \models \phi$ then $\delta_I^{fwd}(\phi) = 0$.*

A formula ϕ has a finite distance $< \infty$ if and only if $\langle A, I, O, \phi \rangle$ has a plan.

Reachability and distances are useful for implementing efficient planning systems. We mention two applications.

First, if we know that no state satisfying a formula ϕ is reachable from the initial states, then we know that no operator $\langle \phi, e \rangle$ can be a part of a plan, and we can ignore any such operator.

Second, distances help in finding a plan. Consider a deterministic planning problem with goal state G. We can now produce a shortest plan by finding an operator o so that $\delta_I^{fwd}(regr_o(G)) < \delta_I^{fwd}(G)$, using $regr_o(G)$ as the new goal state and repeating the process until the initial state I is reached.

Of course, since computing distances is in the worst case just as difficult as planning (PSPACE-complete) it is in general not useful to use subprocedures based on exact distances in a planning algorithm. Instead, different kinds of *approximations* of distances and reachability have to be used. The most important approximations allow the computation of useful reachability and distance information in polynomial time in the size of the succinct transition system. In Section 3.4 we will consider some of them.

3.3.2 Invariants

An *invariant* is a formula that is true in the initial state and in every state that is reached by applying an operator in a state in which it holds. Invariants are closely connected to reachability and distances: a formula ϕ is an invariant if and only if the distance of $\neg\phi$ from the initial state is ∞. Invariants can be used for example to speed up algorithms based on regression.

Definition 3.18 *Let I be a set of initial states and O a set of operators. An formula ϕ is* an invariant *of I, O if $s \models \phi$ for all states s that are reachable from I by a sequence of 0 or more operators in O.*

An invariant ϕ is *the strongest invariant* if $\phi \models \psi$ for any invariant ψ. The strongest invariant exactly characterizes the set of all states that are reachable from the initial state: for every state s, $s \models \phi$ if and only if s is reachable from the initial state. We say "the strongest invariant" even though there are actually several strongest invariants: if ϕ satisfies the properties of the strongest invariant, any other formula that is logically equivalent to ϕ, for example $\phi \vee \phi$, also does. Hence the uniqueness of the strongest invariant has to be understood up to logical equivalence.

Example 3.19 Consider a set of blocks that can be on the table or stacked on top of other blocks. Every block can be on at most one block and on every block there can be one block at most. The actions for moving the blocks can be described by the following schematic operators.

$$\langle \text{ontable}(x) \wedge \text{clear}(x) \wedge \text{clear}(y), \text{on}(x,y) \wedge \neg\text{clear}(y) \wedge \neg\text{ontable}(x) \rangle$$
$$\langle \text{clear}(x) \wedge \text{on}(x,y), \text{ontable}(x) \wedge \text{clear}(y) \wedge \neg\text{on}(x,y) \rangle$$
$$\langle \text{clear}(x) \wedge \text{on}(x,y) \wedge \text{clear}(z), \text{on}(x,z) \wedge \text{clear}(y) \wedge \neg\text{clear}(z) \wedge \neg\text{on}(x,y) \rangle$$

We consider the operators obtained by instantiating the schemata with the objects A, B and C. Let all the blocks be initially on the table. Hence the initial state satisfies the formula

$$\text{clear}(A) \wedge \text{clear}(B) \wedge \text{clear}(C) \wedge \text{ontable}(A) \wedge \text{ontable}(B) \wedge \text{ontable}(C) \wedge$$
$$\neg\text{on}(A,B) \wedge \neg\text{on}(A,C) \wedge \neg\text{on}(B,A) \wedge \neg\text{on}(B,C) \wedge \neg\text{on}(C,A) \wedge \neg\text{on}(C,B)$$

that determines the truth-values of all state variables uniquely. The strongest invariant of this

problem is the conjunction of the following formulae.

$\text{clear}(A) \leftrightarrow (\neg\text{on}(B,A) \wedge \neg\text{on}(C,A))$ $\quad$ $\text{clear}(B) \leftrightarrow (\neg\text{on}(A,B) \wedge \neg\text{on}(C,B))$
$\text{clear}(C) \leftrightarrow (\neg\text{on}(A,C) \wedge \neg\text{on}(B,C))$ $\quad$ $\text{ontable}(A) \leftrightarrow (\neg\text{on}(A,B) \wedge \neg\text{on}(A,C))$
$\text{ontable}(B) \leftrightarrow (\neg\text{on}(B,A) \wedge \neg\text{on}(B,C))$ $\quad$ $\text{ontable}(C) \leftrightarrow (\neg\text{on}(C,A) \wedge \neg\text{on}(C,B))$
$\neg\text{on}(A,B) \vee \neg\text{on}(A,C)$ $\quad$ $\neg\text{on}(B,A) \vee \neg\text{on}(B,C)$
$\neg\text{on}(C,A) \vee \neg\text{on}(C,B)$
$\neg\text{on}(B,A) \vee \neg\text{on}(C,A)$ $\quad$ $\neg\text{on}(A,B) \vee \neg\text{on}(C,B)$
$\neg\text{on}(A,C) \vee \neg\text{on}(B,C)$
$\neg(\text{on}(A,B) \wedge \text{on}(B,C) \wedge \text{on}(C,A))$ $\quad$ $\neg(\text{on}(A,C) \wedge \text{on}(C,B) \wedge \text{on}(B,A))$

We can schematically give the invariants for any set X of blocks as follows.

$\text{clear}(x) \leftrightarrow \forall y \in X \backslash \{x\}.\neg\text{on}(y,x)$
$\text{ontable}(x) \leftrightarrow \forall y \in X \backslash \{x\}.\neg\text{on}(x,y)$
$\neg\text{on}(x,y) \vee \neg\text{on}(x,z)$ when $y \neq z$
$\neg\text{on}(y,x) \vee \neg\text{on}(z,x)$ when $y \neq z$
$\neg(\text{on}(x_1,x_2) \wedge \text{on}(x_2,x_3) \wedge \cdots \wedge \text{on}(x_{n-1},x_n) \wedge \text{on}(x_n,x_1))$ for all $n \geq 1, \{x_1,\ldots,x_n\} \subseteq X$

The last formula says that the *on* relation is acyclic. ■

3.4 Approximations of distances

The approximations of distances are based on the following idea. Instead of considering the number of operators required to reach individual states, we approximately compute the number of operators to reach a state in which a certain state variable has a certain value. So instead of using distances of states, we use distances of literals. This is similar to the distance estimation of belief states in terms of distances of states in Section 5.3.2 in the sense that both use the distances of the components of an object to estimate the distance of the object.

The estimates are not accurate for two reasons. First, and more importantly, distance estimation is done one state variable at a time and dependencies between state variables are ignored. Second, to achieve polynomial-time computation, satisfiability tests for a formula and a set of literals to test the applicability of an operator and to compute the distance estimate of a formula, have to be performed by an inaccurate polynomial-time algorithm that approximates NP-hard satisfiability testing. As we are interested in computing distance estimates efficiently the inaccuracy is a necessary and acceptable compromise.

3.4.1 Admissible max heuristic

We give a recursive procedure that computes a lower bound on the number of operator applications that are needed for reaching from a state I a state in which state variables $a \in A$ have certain values. This is by computing a sequence of sets D_i^{max} of literals. The set D_i^{max} consists literals that are true in all states that have distance $\leq i$ from the state I.

Recall Definition 3.2 of $EPC_l(o)$ for literals l and operators $o = \langle c, e \rangle$:

$$EPC_l(o) = c \wedge EPC_l(e) \wedge \bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e)).$$

Definition 3.20 *Let $L = A \cup \{\neg a | a \in A\}$ be the set of literals on A and I a state. Define the sets D_i^{max} for $i \geq 0$ as follows.*

$$\begin{array}{lcl} D_0^{max} & = & \{l \in L | I \models l\} \\ D_i^{max} & = & D_{i-1}^{max} \backslash \{l \in L | o \in O, D_{i-1}^{max} \cup \{EPC_{\bar{l}}(o)\} \text{ is satisfiable}\}, \text{ for } i \geq 1 \end{array}$$

Since we consider only finite sets A of state variables and $|D_0^{max}| = |A|$ and $D_{i+1}^{max} \subseteq D_i^{max}$ for all $i \geq 0$, necessarily $D_i^{max} = D_j^{max}$ for some $i \leq |A|$ and all $j > i$.

The above computation starts from the set D_0^{max} of all literals that are true in the initial state I. This set of literals characterizes those states that have distance 0 from the initial state. The initial state is the only such state.

Then we repeatedly compute sets of literals characterizing sets of states that are reachable with 1, 2 and more operators. Each set D_i^{max} is computed from the preceding set D_{i-1}^{max} as follows. For each operator o it is tested whether it is applicable in one of the distance $i-1$ states and whether it could make a literal l false. This is by testing whether $EPC_{\bar{l}}(o)$ is true in one of the distance $i-1$ states. If this is the case, the literal l could be false, and it will not be included in D_i^{max}.

The sets of states in which the literals D_i^{max} are true are an upper bound (set-inclusion) on the set of states that have forward distance i.

Theorem 3.21 *Let $D_i^{fwd}, i \geq 0$ be the forward distance sets and D_i^{max} the max-distance sets for I and O. Then for all $i \geq 0$, $D_i^{fwd} \subseteq \{s \in S | s \models D_i^{max}\}$ where S is the set of all states.*

Proof: By induction on i.

Base case $i = 0$: D_0^{fwd} consists of the unique initial state I and D_0^{max} consists of exactly those literals that are true in I, identifying it uniquely. Hence $D_i^{fwd} = \{s \in S | s \models D_i^{max}\}$.

Inductive case $i \geq 1$: Let s be any state in D_i^{fwd}. We show that $s \models D_i^{max}$. Let l be any literal in D_i^{max}.

Assume $s \in D_{i-1}^{fwd}$. As $D_i^{max} \subseteq D_{i-1}^{max}$ also $l \in D_{i-1}^{max}$. By the induction hypothesis $s \models l$.

Otherwise $s \in D_i^{fwd} \backslash D_{i-1}^{fwd}$. Hence there is $o \in O$ and $s_0 \in D_{i-1}^{fwd}$ with $s = app_o(s_0)$. By $D_i^{max} \subseteq D_{i-1}^{max}$ and the induction hypothesis $s_0 \models l$. As $l \in D_i^{max}$, by definition of D_i^{max} the set $D_{i-1}^{max} \cup \{EPC_{\bar{l}}(o)\}$ is not satisfiable. By $s_0 \in D_{i-1}^{fwd}$ and the induction hypothesis $s_0 \models D_{i-1}^{max}$. Hence $s_0 \not\models EPC_{\bar{l}}(o)$. By Lemma 3.3 applying o in s_0 does not make l false. Hence $s \models l$. □

The sets D_i^{max} can be used for estimating the distances of formulae. The distance of a formula is the minimum of the distances of states that satisfy the formula.

Definition 3.22 *Let ϕ be a formula. Define*

$$\delta_I^{max}(\phi) = \begin{cases} 0 \text{ iff } D_0^{max} \cup \{\phi\} \text{ is satisfiable} \\ d \text{ iff } D_d^{max} \cup \{\phi\} \text{ is satisfiable and } D_{d-1}^{max} \cup \{\phi\} \text{ is not satisfiable, for } d \geq 1. \end{cases}$$

Lemma 3.23 *Let I be a state, O a set of operators, and $D_0^{max}, D_1^{max}, \ldots$ the sets given in Definition 3.20 for I and O. Then $app_{o_1;\ldots;o_n}(I) \models D_n^{max}$ for any operators $\{o_1, \ldots, o_n\} \subseteq O$.*

Proof: By induction on n.

Base case $n = 0$: The length of the operator sequence is zero, and hence $app_\epsilon(I) = I$. The set D_0^{max} consists exactly of those literals that are true in s, and hence $I \models D_0^{max}$.

Inductive case $n \geq 1$: By the induction hypothesis $app_{o_1;\ldots;o_{n-1}}(I) \models D_{n-1}^{max}$.

Let l be any literal in D_n^{max}. We show it is true in $app_{o_1;\ldots;o_n}(I)$. Since $l \in D_n^{max}$ and $D_n^{max} \subseteq D_{n-1}^{max}$, also $l \in D_{n-1}^{max}$, and hence by the induction hypothesis $app_{o_1;\ldots;o_{n-1}}(I) \models l$. Since $l \in D_n^{max}$ it must be that $D_{n-1}^{max} \cup \{EPC_{\bar{l}}(o_n)\}$ is not satisfiable (definition of D_n^{max}) and further that $app_{o_1;\ldots;o_{n-1}}(I) \not\models EPC_{\bar{l}}(o_n)$. Hence applying o_n in $app_{o_1;\ldots;o_{n-1}}(I)$ does not make l false, and consequently $app_{o_1;\ldots;o_n}(I) \models l$.

□

The next theorem shows that the distance estimates given for formulae yield a lower bound on the number of actions needed to reach a state satisfying the formula.

Theorem 3.24 *Let I be a state, O a set of operators, ϕ a formula, and $D_0^{max}, D_1^{max}, \ldots$ the sets given in Definition 3.20 for I and O. If $app_{o_1;\ldots;o_n}(I) \models \phi$, then $D_n^{max} \cup \{\phi\}$ is satisfiable.*

Proof: By Lemma 3.23 $app_{o_1;\ldots;o_n}(I) \models D_n^{max}$. By assumption $app_{o_1;\ldots;o_n}(I) \models \phi$. Hence $D_n^{max} \cup \{\phi\}$ is satisfiable. □

Corollary 3.25 *Let I be a state and ϕ a formula. Then for any sequence $o_1, \ldots, o_n$ of operators such that $app_{o_1;\ldots;o_n}(I) \models \phi$, $n \geq \delta_I^{max}(\phi)$.*

The estimate $\delta_s^{max}(\phi)$ never overestimates the distance from s to ϕ and it is therefore an admissible heuristic. It may severely underestimate the distance, as discussed in the end of this section.

Distance estimation in polynomial time

The algorithm for computing the sets D_i^{max} runs in polynomial time except that the satisfiability tests for $D \cup \{\phi\}$ are instances of the NP-complete SAT problem. For polynomial time computation we perform these tests by a polynomial-time approximation that has the property that if $D \cup \{\phi\}$ is satisfiable then $\text{asat}(D, \phi)$ returns true, but not necessarily vice versa. A counterpart of Theorem 3.21 can be established when the satisfiability tests $D \cup \{\phi\}$ are replaced by tests $\text{asat}(D, \phi)$.

The function $\text{asat}(D, \phi)$ tests whether there is a state in which ϕ and the literals D are true, or equivalently, whether $D \cup \{\phi\}$ is satisfiable. This algorithm does not accurately test satisfiability, and may claim that $D \cup \{\phi\}$ is satisfiable even when it is not. This, however, never leads to overestimating the distances, only underestimating. The algorithm runs in polynomial time and is defined as follows.

$$
\begin{aligned}
\text{asat}(D, \bot) &= \text{false} \\
\text{asat}(D, \top) &= \text{true} \\
\text{asat}(D, a) &= \text{true iff } \neg a \notin D \text{ (for state variables } a \in A) \\
\text{asat}(D, \neg a) &= \text{true iff } a \notin D \text{ (for state variables } a \in A) \\
\text{asat}(D, \neg\neg\phi) &= \text{asat}(D, \phi) \\
\text{asat}(D, \phi_1 \vee \phi_2) &= \text{asat}(D, \phi_1) \text{ or } \text{asat}(D, \phi_2) \\
\text{asat}(D, \phi_1 \wedge \phi_2) &= \text{asat}(D, \phi_1) \text{ and } \text{asat}(D, \phi_2) \\
\text{asat}(D, \neg(\phi_1 \vee \phi_2)) &= \text{asat}(D, \neg\phi_1) \text{ and } \text{asat}(D, \neg\phi_2) \\
\text{asat}(D, \neg(\phi_1 \wedge \phi_2)) &= \text{asat}(D, \neg\phi_1) \text{ or } \text{asat}(D, \neg\phi_2)
\end{aligned}
$$

In this and other recursive definitions about formulae the cases for $\neg(\phi_1 \wedge \phi_2)$ and $\neg(\phi_1 \vee \phi_2)$ are obtained respectively from the cases for $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$ by the De Morgan laws.

The reason why the satisfiability test is not accurate is that for formulae $\phi \wedge \psi$ (respectively $\neg(\phi \vee \psi)$) we make recursively two satisfiability tests that do not require that the subformulae ϕ and ψ (respectively $\neg\phi$ and $\neg\psi$) are *simultaneously* satisfiable.

We give a lemma that states the connection between asat(D, ϕ) and the satisfiability of $D \cup \{\phi\}$.

Lemma 3.26 *Let ϕ be a formula and D a consistent set of literals (it contains at most one of a and $\neg a$ for every $a \in A$.) If $D \cup \{\phi\}$ is satisfiable, then asat(D, ϕ) returns true.*

Proof: The proof is by induction on the structure of ϕ.

Base case 1, $\phi = \bot$: The set $D \cup \{\bot\}$ is not satisfiable, and hence the implication trivially holds.

Base case 2, $\phi = \top$: asat$(D, \top)$ always returns true, and hence the implication trivially holds.

Base case 3, $\phi = a$ for some $a \in A$: If $D \cup \{a\}$ is satisfiable, then $\neg a \notin D$, and hence asat(D, a) returns true.

Base case 4, $\phi = \neg a$ for some $a \in A$: If $D \cup \{\neg a\}$ is satisfiable, then $a \notin D$, and hence asat$(D, \neg a)$ returns true.

Inductive case 1, $\phi = \neg\neg\phi'$ for some ϕ': The formulae are logically equivalent, and by the induction hypothesis we directly establish the claim.

Inductive case 2, $\phi = \phi_1 \vee \phi_2$: If $D \cup \{\phi_1 \vee \phi_2\}$ is satisfiable, then either $D \cup \{\phi_1\}$ or $D \cup \{\phi_2\}$ is satisfiable and by the induction hypothesis at least one of asat(D, ϕ_1) and asat(D, ϕ_2) returns true. Hence asat$(D, \phi_1 \vee \phi_2)$ returns true.

Inductive case 3, $\phi = \phi_1 \wedge \phi_2$: If $D \cup \{\phi_1 \wedge \phi_2\}$ is satisfiable, then both $D \cup \{\phi_1\}$ and $D \cup \{\phi_2\}$ are satisfiable and by the induction hypothesis both asat(D, ϕ_1) and asat(D, ϕ_2) return true. Hence asat$(D, \phi_1 \wedge \phi_2)$ returns true.

Inductive cases 4 and 5, $\phi = \neg(\phi_1 \vee \phi_2)$ and $\phi = \neg(\phi_1 \wedge \phi_2)$: Like cases 2 and 3 by logical equivalence. □

The other direction of the implication does not hold because for example asat$(\emptyset, a \wedge \neg a)$ returns true even though the formula is not satisfiable. The procedure is a polynomial-time approximation of the logical consequence test from a set of literals: asat(D, ϕ) always returns true if $D \cup \{\phi\}$ is satisfiable, but it may return true also when the set is not satisfiable.

Informativeness of the max heuristic

The max heuristic often underestimates distances. Consider an initial state in which all n state variables are false and a goal state in which all state variables are true and a set of n operators each of which is always applicable and makes one of the state variables true. The max heuristic assigns the distance 1 to the goal state although the distance is n.

The problem is that assigning every state variable the desired value requires a different operator, and taking the maximum number of operators for each state variable ignores this fact. In this case the actual distance is obtained as the *sum* of the distances suggested by each of the n state variables. In other cases the max heuristic works well when the desired state variable values can be reached with the same operators.

Next we will consider heuristics that are not admissible like the max heuristic but in many cases provide a much better estimate of the distances.

3.4.2 Inadmissible additive heuristic

The max heuristic is very optimistic about the distances, and in many cases very seriously underestimates them. If two goal literals have to be made true, the maximum of the goal costs (distances) is assumed to be the combined cost. This however is only accurate when the easier goal is achieved for free while achieving the more difficult goal. Often the goals are independent and then a more accurate estimate would be the sum of the individual costs. This suggests another heuristic, first considered by Bonet and Geffner [2001] as a more practical variant of the max heuristic in the previous section. Our formalization differs from the one given by Bonet and Geffner.

Definition 3.27 *Let I be a state and $L = A \cup \{\neg a | a \in A\}$ the set of literals. Define the sets D_i^+ for $i \geq 0$ as follows.*

$$\begin{array}{rcl} D_0^+ & = & \{l \in L | I \models l\} \\ D_i^+ & = & D_{i-1}^+ \backslash \{l \in L | o \in O, \mathit{cost}(\mathit{EPC}_{\bar{l}}(o), i) < i\} \text{ for all } i \geq 1 \end{array}$$

We define $\mathit{cost}(\phi, i)$ by the following recursive definition.

$$\begin{array}{rcl} \mathit{cost}(\bot, i) & = & \infty \\ \mathit{cost}(\top, i) & = & 0 \\ \mathit{cost}(a, i) & = & 0 \text{ if } \neg a \notin D_0^+, \text{ for } a \in A \\ \mathit{cost}(\neg a, i) & = & 0 \text{ if } a \notin D_0^+, \text{ for } a \in A \\ \mathit{cost}(a, i) & = & j \text{ if } \neg a \in D_{j-1}^+ \backslash D_j^+ \text{ for some } j < i \\ \mathit{cost}(\neg a, i) & = & j \text{ if } a \in D_{j-1}^+ \backslash D_j^+ \text{ for some } j < i \\ \mathit{cost}(a, i) & = & \infty \text{ if } \neg a \in D_j^+ \text{ for all } j < i \\ \mathit{cost}(\neg a, i) & = & \infty \text{ if } a \in D_j^+ \text{ for all } j < i \\ \mathit{cost}(\phi_1 \vee \phi_2, i) & = & \min(\mathit{cost}(\phi_1, i), \mathit{cost}(\phi_2, i)) \\ \mathit{cost}(\phi_1 \wedge \phi_2, i) & = & \mathit{cost}(\phi_1, i) + \mathit{cost}(\phi_2, i) \\ \mathit{cost}(\neg\neg\phi, i) & = & \mathit{cost}(\phi, i) \\ \mathit{cost}(\neg(\phi_1 \wedge \phi_2), i) & = & \min(\mathit{cost}(\neg\phi_1, i), \mathit{cost}(\neg\phi_2, i)) \\ \mathit{cost}(\neg(\phi_1 \vee \phi_2), i) & = & \mathit{cost}(\neg\phi_1, i) + \mathit{cost}(\neg\phi_2, i) \end{array}$$

Note that a variant of the definition of the max heuristic could be obtained by replacing the sum $+$ in the definition of costs of conjunctions by $\max$. The definition of $\text{cost}(\phi, i)$ approximates satisfiability tests similarly to the definition of $\text{asat}(D, \phi)$ by ignoring the dependencies between propositions.

Similarly to max distances we can define distances of formulae.

Definition 3.28 *Let ϕ be a formula. Define*

$$\delta_I^+(\phi) = \mathit{cost}(\phi, n)$$

where n is the smallest i such that $D_i^+ = D_{i-1}^+$.

The following theorem shows that the distance estimates given by the sum heuristic for literals are at least as high as those given by the max heuristic.

Theorem 3.29 *Let $D_i^{max}, i \geq 0$ be the sets defined in terms of the approximate satisfiability tests $\mathit{asat}(D, \phi)$. Then $D_i^{max} \subseteq D_i^+$ for all $i \geq 0$.*

Proof: The proof is by induction on i.

Base case $i = 0$: By definition $D_0^+ = D_0^{max}$.

Inductive case $i \geq 1$: We have to show that $D_{i-1}^{max} \setminus \{l \in L | o \in O, \text{asat}(D_{i-1}^{max}, \mathit{EPC}_{\bar{l}}(o))\} \subseteq D_{i-1}^+ \setminus \{l \in L | o \in O, \text{cost}(\mathit{EPC}_{\bar{l}}(o), i) < i\}$. By the induction hypothesis $D_{i-1}^{max} \subseteq D_{i-1}^+$. It is sufficient to show that $\text{cost}(\mathit{EPC}_{\bar{l}}(o), i) < i$ implies $\text{asat}(D_{i-1}^{max}, \mathit{EPC}_{\bar{l}}(o))$.

We show this by induction on the structure of $\phi = \mathit{EPC}_{\bar{l}}(o)$.

Induction hypothesis: $\text{cost}(\phi, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \phi)$=true.

Base case 1, $\phi = \bot$: $\text{cost}(\bot, i) = \infty$ and $\text{asat}(D_i^{max}, \bot)$=false.

Base case 2, $\phi = \top$: $\text{cost}(\bot, i) = 0$ and $\text{asat}(D_i^{max}, \bot)$=true.

Base case 3, $\phi = a$: If $\text{cost}(a, i) < i$ then $\neg a \notin D_j^+$ for some $j < i$ or $\neg a \notin D_0^+$. Hence $\neg a \notin D_{i-1}^+$. By the outer induction hypothesis $\neg a \notin D_{i-1}^{max}$ and consequently $\neg a \notin D_i^{max}$. Hence $\text{asat}(D_i^{max}, \bot)$=true.

Base case 4, $\phi = \neg a$: Analogous to the case $\phi = a$.

Inductive case 5, $\phi = \phi_1 \vee \phi_2$: Assume $\text{cost}(\phi_1 \vee \phi_2, i) < i$. Since $\text{cost}(\phi_1 \vee \phi_2, i) = \min(\text{cost}(\phi_1, i), \text{cost}(\phi_2, i))$, either $\text{cost}(\phi_1, i) < i$ or $\text{cost}(\phi_2, i) < i$. By the induction hypothesis $\text{cost}(\phi_1, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \phi_1)$, and $\text{cost}(\phi_2, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \phi_2)$. Hence either $\text{asat}(D_{i-1}^{max}, \phi_1)$ or $\text{asat}(D_{i-1}^{max}, \phi_2)$. Therefore by definition $\text{asat}(D_{i-1}^{max}, \phi_1 \vee \phi_2)$.

Inductive case 6, $\phi = \phi_1 \wedge \phi_2$: Assume $\text{cost}(\phi_1 \wedge \phi_2, i) < i$. Since $i \geq 1$ and $\text{cost}(\phi_1 \vee \phi_2, i) = \text{cost}(\phi_1, i) + \text{cost}(\phi_2, i)$, both $\text{cost}(\phi_1, i) < i$ and $\text{cost}(\phi_2, i) < i$. By the induction hypothesis $\text{cost}(\phi_1, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \phi_1)$, and $\text{cost}(\phi_2, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \phi_2)$. Hence both $\text{asat}(D_{i-1}^{max}, \phi_1)$ an $\text{asat}(D_{i-1}^{max}, \phi_2)$. Therefore by definition $\text{asat}(D_{i-1}^{max}, \phi_1 \wedge \phi_2)$.

Inductive case 7, $\phi = \neg\neg\phi_1$: By the induction hypothesis $\text{cost}(\phi_1, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \phi_1)$. By definition $\text{cost}(\neg\neg\phi_1, i) = \text{cost}(\phi_1, i)$ and $\text{asat}(D, \neg\neg\phi) = \text{asat}(D, \phi)$. By the induction hypothesis $\text{cost}(\neg\neg\phi_1, i) < i$ implies $\text{asat}(D_{i-1}^{max}, \neg\neg\phi_1)$.

Inductive case 8, $\phi = \neg(\phi_1 \vee \phi_2)$: Analogous to the case $\phi = \phi_1 \wedge \phi_2$.

Inductive case 9, $\phi = \neg(\phi_1 \wedge \phi_2)$: Analogous to the case $\phi = \phi_1 \vee \phi_2$. □

That the sum heuristic gives higher estimates than the max heuristic could in many cases be viewed as an advantage because the estimates would be more accurate. However, in some cases this leads to overestimating the actual distance, and therefore the sum distances are not an admissible heuristic.

Example 3.30 Consider an initial state such that $I \models \neg a \wedge \neg b \wedge \neg c$ and the operator $\langle \top, a \wedge b \wedge c \rangle$. A state satisfying $a \wedge b \wedge c$ is reached by this operator in one step but $\delta_I^+(a \wedge b \wedge c) = 3$. ■

3.4.3 Relaxed plan heuristic

The max heuristic and the additive heuristic represent two extremes. The first assumes that sets of operators required for reaching the individual goal literals maximally overlap in the sense that the operators needed for the most difficult goal literal include the operators needed for all the remaining ones. The second assumes that the required operators are completely disjoint.

Usually, of course, the reality is somewhere in between and which notion is better depends on the properties of the operators. This suggests yet another heuristic: we attempt to find a set of operators that approximates, in a sense that will become clear later, the smallest set of operators that are needed to reach a state from another state. This idea has been considered by Hoffman and Nebel [2001]. If the approximation is exact, the cardinality of this set equals the actual dis-

tance between the states. The approximation may both overestimate and underestimate the actual distance, and hence it is does not yield an admissible heuristic.

The idea of the heuristic is the following. We first choose a set of goal literals the truth of which is sufficient for the truth of G. These literals must be reachable in the sense of the sets D_i^{max} which we defined earlier. Then we identify those goal literals that were the last to become reachable and a set of operators making them true. A new goal formula represents the conditions under which these operator can make the literals true, and a new set of goal literals is produced by a simplified form of regression from the new goal formula. The computation is repeated until we have a set of goal literals that are true in the initial state.

The function goals(D, ϕ) recursively finds a set M of literals such that $M \models \phi$ and each literal in M is consistent with D. Note that M itself is not necessarily consistent, for example for $D = \emptyset$ and $\phi = a \wedge \neg a$ we get $M = \{a, \neg a\}$. If a set M is found goals$(D, \phi) = \{M\}$ and otherwise goals$(D, \phi) = \emptyset$.

Definition 3.31 *Let D be a set of literals.*

$$
\begin{array}{rcl}
\mathit{goals}(D, \bot) & = & \emptyset \\
\mathit{goals}(D, \top) & = & \{\emptyset\} \\
\mathit{goals}(D, a) & = & \{\{a\}\} \textit{ if } \neg a \notin D \\
\mathit{goals}(D, a) & = & \emptyset \textit{ if } \neg a \in D \\
\mathit{goals}(D, \neg a) & = & \{\{\neg a\}\} \textit{ if } a \notin D \\
\mathit{goals}(D, \neg a) & = & \emptyset \textit{ if } a \in D \\
\mathit{goals}(D, \neg\neg\phi) & = & \mathit{goals}(D, \phi) \\
\mathit{goals}(D, \phi_1 \vee \phi_2) & = & \begin{cases} \mathit{goals}(D, \phi_1) & \textit{if } \mathit{goals}(D, \phi_1) \neq \emptyset \\ \mathit{goals}(D, \phi_2) & \textit{otherwise} \end{cases} \\
\mathit{goals}(D, \phi_1 \wedge \phi_2) & = & \begin{cases} \{L_1 \cup L_2\} & \textit{if } \mathit{goals}(D, \phi_1) = \{L_1\} \textit{ and } \mathit{goals}(D, \phi_2) = \{L_2\} \\ \emptyset & \textit{otherwise} \end{cases} \\
\mathit{goals}(D, \neg(\phi_1 \wedge \phi_2)) & = & \begin{cases} \mathit{goals}(D, \neg\phi_1) & \textit{if } \mathit{goals}(D, \neg\phi_1) \neq \emptyset \\ \mathit{goals}(D, \neg\phi_2) & \textit{otherwise} \end{cases} \\
\mathit{goals}(D, \neg(\phi_1 \vee \phi_2)) & = & \begin{cases} \{L_1 \cup L_2\} & \textit{if } \mathit{goals}(D, \neg\phi_1) = \{L_1\} \textit{ and } \mathit{goals}(D, \neg\phi_2) = \{L_2\} \\ \emptyset & \textit{otherwise} \end{cases}
\end{array}
$$

Above in the case for $\phi_1 \vee \phi_2$ if both ϕ_1 and ϕ_2 yield a set of goal literals the set for ϕ_1 is always chosen. A practically better implementation is to choose the smaller of the two sets.

Lemma 3.32 *Let D be a set of literals and ϕ a formula.*

1. *goals$(D, \phi) \neq \emptyset$ if and only if asat$(D, \phi) =$ true.*
2. *If goals$(D, \phi) = \{M\}$ then $\{\bar{l} | l \in M\} \cap D = \emptyset$ and asat$(D, \bigwedge_{l \in M} l) =$ true.*

Proof:

1. This is by an easy induction proof on the structure of ϕ based on the definitions of asat(D, ϕ) and goals(D, ϕ).
2. This is because $\bar{l} \notin D$ for all $l \in M$. This can be shown by a simple induction proof.

□

procedure relaxedplan(A,I,O,G);
$L := A \cup \{\neg a | a \in A\}$; *(* All literals *)*
compute sets D_i^{max} as in Definition 3.20;
if asat$(D_i^{max}, G) =$ false for all $i \geq 0$ **then return** ∞; *(* Goal not reachable *)*
$t := \delta_I^{max}(G)$;
$L_{t+1}^G := \emptyset$;
$N_{t+1} := \emptyset$;
$G_t := G$;
for $i := t$ **downto** 1 **do**
  begin
    $L_i^G := (L_{i+1}^G \backslash N_{i+1}) \cup \{l \in M | M \in \text{goals}(D_i^{max}, G_i)\}$; *(* The goal literals *)*
    $N_i := \{l \in L_i^G | \bar{l} \in D_{i-1}^{max}\}$; *(* Goal literals that become true between $i-1$ and i *)*
    $T_i :=$ a minimal subset of O so that $N_i \subseteq \{l \in L | o \in T_i, \text{asat}(D_{i-1}^{max}, \mathit{EPC}_l(o))\}$;
    $G_{i-1} := \bigwedge_{l \in N_i} \bigvee \{\mathit{EPC}_l(o) | o \in T_i\}$; *(* New goal formula *)*
  end
return $|T_1| + |T_2| + \cdots + |T_t|$;

Figure 3.1: Algorithm for finding a relaxed plan

Lemma 3.33 *Let D and $D' \subseteq D$ be sets of literals. If goals$(D, \phi) = \emptyset$ and goals$(D', \phi) = \{M\}$ for some M, then there is $l \in M$ such that $\bar{l} \in D \backslash D'$.*

Proof: Proof is by induction in the structure of formulae ϕ.

Induction hypothesis: If goals$(D, \phi) = \emptyset$ and goals$(D', \phi) = \{M\}$ for some M, then there is $l \in M$ such that $\bar{l} \in D \backslash D'$.

Base cases 1 & 2, $\phi = \top$ and 2 $\phi = \bot$: Trivial as the condition cannot hold.

Base case 3, $\phi = a$: If goals$(D, a) = \emptyset$ and goals$(D', a) = M = \{\{a\}\}$, then respectively $\neg a \in D$ and $\neg a \notin D'$. Hence there is $a \in M$ such that $\overline{a} \in D \backslash D'$.

Inductive case 1, $\phi = \neg\neg\phi'$: By the induction hypothesis as goals$(D, \neg\neg\phi') =$ goals(D, ϕ').

Inductive case 2, $\phi = \phi_1 \vee \phi_2$: Assume goals$(D, \phi_1 \vee \phi_2) = \emptyset$ and goals$(D', \phi_1 \vee \phi_2) = \{M\}$ for some M. Hence goals$(D, \phi_1) = \emptyset$ and goals$(D, \phi_2) = \emptyset$, and goals$(D', \phi_1) = \{M\}$ or goals$(D', \phi_2) = \{M\}$. Hence by the induction hypothesis with ϕ_1 or ϕ_2 there is $l \in M$ such that $\bar{l} \in D \backslash D'$.

Inductive case 3, $\phi = \phi_1 \wedge \phi_2$: Assume goals$(D, \phi_1 \wedge \phi_2) = \emptyset$ and goals$(D', \phi_1 \wedge \phi_2) = \{M\}$ for some M. Hence goals$(D, \phi_1) = \emptyset$ or goals$(D, \phi_2) = \emptyset$, and goals$(D', \phi_1) = \{L_1\}$ and goals$(D', \phi_2) = \{L_2\}$ for some L_1 and L_2 such that $M = L_1 \cup L_2$. Hence by the induction hypothesis with ϕ_1 or ϕ_2 there is either $l \in L_1$ or $l \in L_2$ such that $\bar{l} \in D \backslash D'$.

Inductive cases $\phi = \neg(\phi_1 \wedge \phi_2)$ and $\phi = \neg(\phi_1 \vee \phi_2)$ are analogous to cases 2 and 3. □

Definition 3.34 *Define $\delta_I^{rlx}(\phi) =$ relaxedplan(A, I, O, ϕ).*

Like the sum heuristic, the relaxed plan heuristic gives higher distance estimates than the max heuristic.

Theorem 3.35 *Let ϕ be a formula and $\delta_I^{max}(\phi)$ the max-distance defined in terms of asat(D, ϕ). Then $\delta_I^{rlx}(\phi) \geq \delta_I^{max}(\phi)$.*

Proof: We have to show that for any formula G the procedure call *relaxedplan*(A,I,O,G) returns a number $\geq \delta_I^{max}(G)$.

First, the procedure returns ∞ if and only if asat(D_i^{max}, G) = false for all $i \geq 0$. In this case by definition $\delta_I^{max}(G) = \infty$.

Otherwise $t = \delta_I^{max}(G)$. Now $t = 0$ if and only if asat(D_0^{max}, G) = true. In this case the procedure returns 0 without iterating the loop starting on line 9.

We show that if $t \geq 1$ then for every $i \in \{1, \ldots, t\}$ the set T_i is non-empty, entailing $|T_1| + \cdots + |T_t| \geq t = \delta_I^{max}(G)$. This is by an induction proof from t to 1.

We use the following auxiliary result. If asat(D_{i-1}^{max}, G_i) = false and asat(D_i^{max}, G_i) = true and $\bar{l} \notin D_i^{max}$ for all $l \in L_i^G$ then T_i is well-defined and $T_i \neq \emptyset$. The proof is as follows.

> By Lemma 3.32 goals$(D_{i-1}^{max}, G_i) = \emptyset$ and goals$(D_i^{max}, G_i) = \{M\}$ for some M. By Lemma 3.33 there is $l \in M$ such that $\bar{l} \in D_{i-1}^{max}$ and hence $N_i \neq \emptyset$. By definition $\bar{l} \in D_{i-1}^{max}$ for all $l \in N_i$. By $N_i \subseteq L_i^G$ and the assumption about L_i^G $\bar{l} \notin D_i^{max}$ for all $l \in N_i$. Hence $\bar{l} \in D_{i-1}^{max} \backslash D_i^{max}$ for all $l \in N_i$. Hence by definition of D_i^{max} for every $l \in N_i$ there is $o \in O$ such that asat$(D_{i-1}^{max}, \mathit{EPC}_l(o))$. Hence there is $T_i \subseteq O$ so that $N_i \subseteq \{l \in L | o \in T_i, \text{asat}(D_{i-1}^{max}, \mathit{EPC}_l(o))\}$ and the value of T_i is defined. As $N_i \neq \emptyset$ also $T_i \neq \emptyset$.

In the induction proof we establish the assumptions of the auxiliary result and then invoke the auxiliary result itself.

Induction hypothesis: For all $j \in \{i, \ldots, t\}$

1. $\bar{l} \notin D_j^{max}$ for all $l \in L_j^G$,
2. asat(D_j^{max}, G_j) = true and asat(D_{j-1}^{max}, G_j) = false, and
3. $T_j \neq \emptyset$.

Base case $i = t$:

1. $\bar{l} \notin D_t^{max}$ for all $l \in L_t^G$ by (2) of Lemma 3.32 because $L_t^G = \{l \in \text{goals}(D_t^{max}, G_t)\}$.
2. As $t = \delta_I^{max}(G_t)$ by definition asat(D_{t-1}^{max}, G_t) = false and asat(D_t^{max}, G_t) = true.
3. By the auxiliary result from the preceding case.

Inductive case $i < t$:

1. We have $\bar{l} \notin D_i^{max}$ for all $l \in L_i^G$ because $L_i^G = (L_{i+1}^G \backslash N_{i+1}) \cup \{l \in \text{goals}(D_i^{max}, G_i)\}$ and by the induction hypothesis $\bar{l} \notin D_{i+1}^{max}$ for all $l \in L_{i+1}^G$ and by (2) of Lemma 3.32 $\bar{l} \notin D_i^{max}$ for all $l \in M$ for $M \in \text{goals}(D_i^{max}, G_i)$.
2. By definition $G_i = \bigwedge_{l \in N_{i+1}} \bigvee \{\mathit{EPC}_l(o) | o \in T_{i+1}\}$. By definition of T_{i+1} for every $l \in N_{i+1}$ there is $o \in T_{i+1}$ such that asat$(D_i^{max}, \mathit{EPC}_l(o))$ = true. By definition of asat$(D_i^{max}, \phi_1 \vee \phi_2)$ and asat$(D_i^{max}, \phi_1 \wedge \phi_2)$ for ϕ_1 and ϕ_2 also asat(D_i^{max}, G_i) = true.

   Then we show that asat(D_{i-1}^{max}, G_i) = false. By definition of D_i^{max}, asat$(D_{i-1}^{max}, \mathit{EPC}_{\bar{l}}(o))$ = false for all $l \in D_i^{max}$ and $o \in O$. Hence asat$(D_{i-1}^{max}, \mathit{EPC}_l(o))$ = false for all $l \in N_{i+1}$ and $o \in O$ because $\bar{l} \in D_i^{max}$. Hence asat$(D_{i-1}^{max}, \mathit{EPC}_l(o))$ = false for all $l \in N_{i+1}$ and $o \in T_{i+1}$ because $T_{i+1} \subseteq O$. By definition $G_i = \bigwedge_{l \in N_{i+1}} \bigvee \{\mathit{EPC}_l(o) | o \in T_{i+1}\}$. Hence by definition of asat(D, ϕ) also asat(D_{i-1}^{max}, G_i) = false.

3. By the auxiliary result from the preceding case.

□

3.5 Algorithm for computing invariants

Planning with backward search and regression suffers from the following problem. Often only a fraction of all valuations of state variables represent states that are reachable from the initial state and represent possible world states. The goal formula and many of the formulae produced by regression often represent many unreachable states. If the formulae represent only unreachable states a planning algorithm may waste a lot of effort determining that a certain sequence of actions is not the suffix of any plan[1]. Also planning with propositional logic (Section 3.6) suffers from the same problem.

Planning can be made more efficient by restricting search to states that are reachable from the initial state. However, determining whether a given state is reachable from the initial state is PSPACE-complete. Consequently, exact information on the reachability of states could not be used for speeding up the basic forward and backward search algorithms: solving the subproblem would be just as complex as solving the problem itself.

In this section we will present a polynomial time algorithm for computing a class of invariants that approximately characterize the set of reachable states. These invariants help in improving the efficiency of planning algorithms based on backward search and on satisfiability testing in the propositional logic (Section 3.6).

Our algorithm computes invariants that are clauses with at most n literals, for some fixed n. For representing the strongest invariant arbitrarily high n may be needed. Although the runtime is polynomial for any fixed n, the runtimes grow quickly as n increases. However, for many applications short invariants of length $n = 2$ are sufficient, and longer invariants are less important.

The algorithm first computes the set of all 1-literal clauses that are true in the initial state. This set exactly characterizes the set of distance 0 states consisting of the initial state only. Then the algorithm considers the application of every operator. If an operator is applicable it may make some of the clauses false. These clauses are removed and replaced by weaker clauses which are also tested against every operator. When no further clauses are falsified, we have a set of clauses that are guaranteed to be true in all distance 1 states. This computation is repeated for distances 2, 3, and so on, until the clause set does not change. The resulting clauses are invariants because they are true after any number of operator applications.

The flavor of the algorithm is similar to the distance estimation in Section 3.4: starting from a description of what is possible in the initial state, inductively determine what is possible after i operator applications. In contrast to the distance estimation method in Section 3.4 the state sets are characterized by sets of clauses instead of sets of literals.

Let C_i be a set of clauses that characterizes those states that are reachable by i operator applications. Similarly to distance computation, we consider for each operator and for each clause in C_i whether applying the operator may make the clause false. If it can, the clause could be false after i operator applications and therefore will not be in the set C_{i+1}.

[1] A symmetric problem arises with forward search because with progression one may reach states from which goal states are unreachable.

```
1:  procedure preserved(φ,C,o);
2:  φ = l_1 ∨ ··· ∨ l_n for some l_1, ..., l_n and o = ⟨c, e⟩ for some c and e;
3:  for each l ∈ {l_1, ..., l_n} do
4:    if C ∪ {EPC_l̄(o)} is unsatisfiable then goto OK;        (* l cannot become false. *)
5:    for each l' ∈ {l_1, ..., l_n}\{l} do    (* Otherwise another literal in φ must be true. *)
6:      if C ∪ {EPC_l̄(o)} ⊨ EPC_l'(o) then goto OK;          (* l' becomes true. *)
7:      if C ∪ {EPC_l̄(o)} ⊨ l' ∧ ¬EPC_l̄'(o) then goto OK;    (* l' was and stays true. *)
8:    end do
9:    return false;                    (* Truth of the clause could not be guaranteed. *)
10:   OK:
11: end do
12: return true;
```

Figure 3.2: Algorithm that tests whether o may falsify $l_1 \vee \cdots \vee l_n$ in a state satisfying C

Figure 3.2 gives an algorithm that tests whether applying an operator $o \in O$ in some state s may make a formula $l_1 \vee \cdots \vee l_n$ false assuming that $s \models C \cup \{l_1 \vee \cdots \vee l_n\}$.

The algorithm performs a case analysis for every literal in the clause, testing in each case whether the clause remains true: if a literal becomes false, either another literal becomes true simultaneously or another literal was true before and does not become false.

Lemma 3.36 *Let C be a set of clauses, $\phi = l_1 \vee \cdots \vee l_n$ a clause, and o an operator. If preserved(ϕ,C,o) returns* true*, then $app_o(s) \models \phi$ for any state s such that $s \models C \cup \{\phi\}$ and o is applicable in s. (It may under these conditions also return* false*).*

Proof: Assume s is a state such that $s \models C \wedge \phi$, $app_o(s)$ is defined and $app_o(s) \not\models \phi$. We show that the procedure returns *false*.

Since $s \models \phi$ and $app_o(s) \not\models \phi$ at least one literal in ϕ is made false by o. Let $\{l_1^\perp, \ldots, l_m^\perp\} \subseteq \{l_1, \ldots, l_n\}$ be the set of all such literals. Hence $s \models l_1^\perp \wedge \cdots \wedge l_m^\perp$ and $\{\overline{l_1^\perp}, \ldots, \overline{l_m^\perp}\} \subseteq [e]_s^{det}$. The literals in $\{l_1, \ldots, l_n\}\backslash\{l_1^\perp, \ldots, l_m^\perp\}$ are false in s and o does not make them true.

Choose any $l \in \{l_1^\perp, \ldots, l_m^\perp\}$. We show that when the outermost *for each* loop starting on line 3 considers l the procedure will return *false*.

Since $\bar{l} \in [e]_s^{det}$ and o is applicable in s by Lemma 3.3 $s \models EPC_{\bar{l}}(o)$. Since by assumption $s \models C$, the condition of the *if* statement on line 4 is not satisfied and the execution proceeds by iteration of the inner *for each* loop.

Let l' be any of the literals in ϕ except l. Since $app_o(s) \not\models \phi$, $l' \notin [e]_s^{det}$. Hence by Lemma 3.3 $s \not\models EPC_{l'}(o)$, and as $s \models C \cup \{EPC_{\bar{l}}(o)\}$ the condition of the *if* statement on line 6 is not satisfied and the execution continues from line 7. Analyze two cases.

1. If $l' \in \{l_1^\perp, \ldots, l_m^\perp\}$ then by assumption $\overline{l'} \in [e]_s^{det}$ and by Lemma 3.3 $s \models EPC_{\overline{l'}}(o)$. Hence $C \cup \{EPC_{\bar{l}}(o)\} \not\models \neg EPC_{\overline{l'}}(o)$ and the condition of the *if* statement on line 7 is not satisfied.

2. If $l' \notin \{l_1^\perp, \ldots, l_m^\perp\}$ then $s \not\models l'$. Hence $C \cup \{EPC_{\bar{l}}(o)\} \not\models l'$ and the condition of the *if* statement on line 7 is not satisfied.

Hence on none of the iterations of the inner *for each* loop is a *goto OK* executed, and as the loop exits, the procedure returns *false*. □

```
1:  procedure invariants(A, I, O, n);
2:  C := {a ∈ A | I ⊨ a} ∪ {¬a | a ∈ A, I ⊭ a};        (* Clauses true in the initial state *)
3:  repeat
4:    C' := C;
5:    for each o ∈ O and l_1 ∨ ··· ∨ l_m ∈ C such that not preserved(l_1 ∨ ··· ∨ l_m,C',o) do
6:      C := C\{l_1 ∨ ··· ∨ l_m};
7:      if m < n then                                  (* Clause length within pre-defined limit. *)
8:        begin                                        (* Add weaker clauses. *)
9:          C := C ∪ {l_1 ∨ ··· ∨ l_m ∨ a | a ∈ A, {a, ¬a} ∩ {l_1, ..., l_m} = ∅};
10:         C := C ∪ {l_1 ∨ ··· ∨ l_m ∨ ¬a | a ∈ A, {a, ¬a} ∩ {l_1, ..., l_m} = ∅};
11:       end
12:   end do
13: until C = C';
14: return C;
```

Figure 3.3: Algorithm for computing a set of invariant clauses

Figure 3.3 gives the algorithm for computing invariants consisting of at most n literals. The loop on line 5 is repeated until there are no $o \in O$ and clauses ϕ in C such that preserved(ϕ,C',o) returns false. This exit condition for the loop is critical for the correctness proof.

Theorem 3.37 *Let A be a set of state variables, I a state, O a set of operators, and $n \geq 1$ an integer. Then the procedure call invariants(A, I, O, n) returns a set C of clauses with at most n literals so that for any sequence $o_1; \ldots; o_m$ of operators from O $app_{o_1;\ldots;o_m}(I) \models C$.*

Proof: Let C_0 be the value first assigned to the variable C in the procedure *invariants*, and $C_1, C_2, \ldots$ be the values of the variable in the end of each iteration of the outermost *repeat* loop.

Induction hypothesis: for every $\{o_1, \ldots, o_i\} \subseteq O$ and $\phi \in C_i$, $app_{o_1;\ldots;o_i}(I) \models \phi$.

Base case $i = 0$: $app_\epsilon(I)$ for the empty sequence is by definition I itself, and by construction C_0 consists of only formulae that are true in the initial state.

Inductive case $i \geq 1$: Take any $\{o_1, \ldots, o_i\} \subseteq O$ and $\phi \in C_i$. First notice that preserved(ϕ,C_i,o) returns *true* because otherwise ϕ could not be in C_i. Analyze two cases.

1. If $\phi \in C_{i-1}$, then by the induction hypothesis $app_{o_1;\ldots;o_{i-1}}(I) \models \phi$. Since $\phi \in C_i$ preserved(ϕ,C_{i-1},o) returns *true*. Hence by Lemma 3.36 $app_{o_1;\ldots;o_i}(I) \models \phi$.

2. If $\phi \notin C_{i-1}$, it must be because preserved(ϕ',C_{i-1},o') returns *false* for some $o' \in O$ and $\phi' \in C_{i-1}$ such that ϕ is obtained from ϕ' by conjoining some literals to it. Hence $\phi' \models \phi$.

   Since $\phi' \in C_{i-1}$ by the induction hypothesis $app_{o_1;\ldots;o_{i-1}}(I) \models \phi'$. Since $\phi' \models \phi$ also $app_{o_1;\ldots;o_{i-1}}(I) \models \phi$. Since the function call preserved(ϕ,C_i,o) returns *true* by Lemma 3.36 $app_{o_1;\ldots;o_i}(I) \models \phi$.

This finishes the induction proof. The iteration of the procedure stops when $C_i = C_{i-1}$, meaning that the claim of the theorem holds for arbitrarily long sequences $o_1; \ldots; o_m$ of operators. □

The algorithm does not find the strongest invariant for two reasons. First, only clauses until some fixed length are considered. Expressing the strongest invariant may require clauses that are

longer. Second, the test performed by *preserved* tries to prove for one of the literals in the clause that it is true after an operator application. Consider the clause $a \vee b \vee c$ and the operator $\langle b \vee c, \neg a \rangle$. We cannot show for any literal that it is true after applying the operator but we know that either b or c is true. The test performed by *preserved* could be strengthened to handle cases like these, for example by using the techniques discussed in Section 5.2, but this would make the computation more expensive and eventually lead to intractability.

To make the algorithm run in polynomial time the satisfiability and logical consequence tests should be performed by algorithms that approximate these tests in polynomial time. The procedure asat(D, ϕ) is not suitable because it assumes that D is a set of literals, whereas for *preserved* the set C usually contain clauses with 2 or more literals. There are generalizations of the ideas behind asat(D, ϕ) to this more general case but we do not discuss the topic further.

3.5.1 Applications of invariants in planning by regression and satisfiability

Invariants can be used to speed up backward search with regression. Consider the blocks world with the goal *AonB*$\wedge$*BonC*. Regression with the operator that moves B onto C from the table yields *AonB* $\wedge$ *Bclear* $\wedge$ *Cclear* $\wedge$ *BonT*. This formula does not correspond to an intended blocks world state because *AonB* is incompatible with *Bclear*, and indeed, $\neg$*AonB* $\vee$ $\neg$*Bclear* is an invariant for the blocks world. Any regression step that leads to a formula that is incompatible with the invariants can be ignored because that formula does not represent any state that is reachable from the initial state, and hence no plan extending the current incomplete plan can reach the goals.

Another application of invariants and the intermediate sets C_i produced by our invariant algorithm is improving the heuristics in Section 3.4. Using D_i^{max} for testing whether an operator precondition, for example $a \wedge b$, has distance i from the initial state, the distances of a and b are used separately. But even when it is possible to reach both a and b with i operator applications, it might still not be possible to reach them both simultaneously with i operator applications. For example, for $i = 1$ and an initial state in which both a and b are false, there might be no single operator that makes them both true, but two operators, each of which makes only one of them true. If $\neg a \vee \neg b \in C_i$, we know that after i operator applications one of a or b must still be false, and then we know that the operator in question is not applicable at time point i. Therefore the invariants and the sets C_i produced during the invariant computation can improve distance estimates.

3.6 Planning as satisfiability in the propositional logic

A very powerful approach to deterministic planning was introduced in 1992 by Kautz and Selman [1992; 1996]. In this approach the problem of reachability of a goal state from a given initial state is translated into propositional formulae $\phi_0, \phi_1, \phi_2, \ldots$ so that every valuation that satisfies formula ϕ_i corresponds to a plan of length i. Planning proceeds by first testing the satisfiability of ϕ_0. If ϕ_0 is unsatisfiable, continue with ϕ_1, ϕ_2, and so on, until a satisfiable formula ϕ_n is found. From a valuation that satisfies ϕ_n a plan of length n can be constructed.

3.6.1 Actions as propositional formulae

First we need a representation of actions in the propositional logic. We can view arbitrary propositional formulae as actions, or we can translate operators into formulae in the propositional logic. We discuss both of these possibilities.

Given a set of state variables $A = \{a_1, \ldots, a_n\}$, one could describe an action directly as a propositional formula ϕ over propositional variables $A \cup A'$ where $A' = \{a'_1, \ldots, a'_n\}$. Here the variables A represent the values of state variables in the state s in which an action is taken, and variables A' the values of state variables in a successor state s'.

A pair of valuations s and s' can be understood as a valuation of $A \cup A'$ (the state s assigns a value to variables A and s' to variables A'), and a transition from s to s' is possible if and only if $s, s' \models \phi$.

Example 3.38 The action that reverses the values of state variables a_1 and a_2 is described by $\phi = (a_1 \leftrightarrow \neg a'_1) \wedge (a_2 \leftrightarrow \neg a'_2)$. The following 4×4 incidence matrix represents this action.

	$a'_1a'_2$	$a'_1a'_2$	$a'_1a'_2$	$a'_1a'_2$
a_1a_2	00	01	10	11
00	0	0	0	1
01	0	0	1	0
10	0	1	0	0
11	1	0	0	0

The matrix can be equivalently represented as the following truth-table.

a_1	a_2	a'_1	a'_2	ϕ
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

■

Example 3.39 Let the set of state variables be $A = \{a_1, a_2, a_3\}$. The formula $(a_1 \leftrightarrow a'_2) \wedge (a_2 \leftrightarrow a'_3) \wedge (a_3 \leftrightarrow a'_1)$ represents the action that rotates the values of the state variables a_1, a_2 and a_3 one position right. The formula can be represented as the following adjacency matrix. The rows

correspond to valuations of A and the columns to valuations of $A' = \{a'_1, a'_2, a'_3\}$.

	000	001	010	011	100	101	110	111
000	1	0	0	0	0	0	0	0
001	0	0	0	0	1	0	0	0
010	0	1	0	0	0	0	0	0
011	0	0	0	0	0	1	0	0
100	0	0	1	0	0	0	0	0
101	0	0	0	0	0	0	1	0
110	0	0	0	1	0	0	0	0
111	0	0	0	0	0	0	0	1

A more conventional way of depicting the valuations of this formula would be as a truth-table with one row for every valuation of $A \cup A'$, a total of 64 rows. ■

The action in Example 3.39 is deterministic. Not all actions represented by propositional formulae are deterministic. A sufficient (but not necessary) condition for determinism is that the formula is of the form $(\phi_1 \leftrightarrow a'_1) \wedge \cdots \wedge (\phi_n \leftrightarrow a'_n) \wedge \psi$ where $A = \{a_1, \ldots, a_n\}$ is the set of all state variables, ϕ_i are formulae over A (without occurrences of $A' = \{a'_1, \ldots, a'_n\}$). There are no restrictions on ψ. Formulae of this form uniquely determine the value of every state variable in the successor state in terms of the values in the predecessor state. Therefore they represent deterministic actions.

3.6.2 Translation of operators into propositional logic

We first give the simplest possible translation of deterministic planning into the propositional logic. In this translation every operator is separately translated into a formula, and the choice between the operators is represented as disjunction.

Definition 3.40 *The formula $\tau_A(o)$ which represents the operator $o = \langle c, e\rangle$ is defined by*

$$\begin{array}{lcl} \tau_A(e) & = & \bigwedge_{a\in A}((\mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))) \leftrightarrow a') \wedge \bigwedge_{a\in A} \neg(\mathit{EPC}_a(e) \wedge \mathit{EPC}_{\neg a}(e)) \\ \tau_A(o) & = & c \wedge \tau_A(e). \end{array}$$

The formula $\tau_A(e)$ expresses the value of a in the successor state in terms of the values of the state variables in the predecessor state and requires that executing e may not make any state variable simultaneously true and false. This is like in the definition of regression in Section 3.1.2. The formula $\tau_A(o)$ additionally requires that the operator's precondition is true.

Example 3.41 Consider operator $\langle a \vee b, (b \rhd a) \wedge (c \rhd \neg a) \wedge (a \rhd b)\rangle$. The corresponding propositional formula is

$$\begin{array}{rl} (a \vee b) & \wedge((b \vee (a \wedge \neg c)) \leftrightarrow a') \\ & \wedge((a \vee (b \wedge \neg\bot)) \leftrightarrow b') \\ & \wedge((\bot \vee (c \wedge \neg\bot)) \leftrightarrow c') \\ & \wedge\neg(b \wedge c) \wedge \neg(a \wedge \bot) \wedge \neg(\bot \wedge \bot) \\ \equiv (a \vee b) & \wedge((b \vee (a \wedge \neg c)) \leftrightarrow a') \\ & \wedge((a \vee b) \leftrightarrow b') \\ & \wedge(c \leftrightarrow c') \\ & \wedge\neg(b \wedge c). \end{array}$$

■

Lemma 3.42 *Let s and s' be states and o an operator. Let $v : A \cup A' \rightarrow \{0,1\}$ be a valuation such that*

1. *for all $a \in A$, $v(a) = s(a)$, and*
2. *for all $a \in A$, $v(a') = s'(a)$.*

Then $v \models \tau_A(o)$ if and only if $s' = app_o(s)$.

Proof: Assume $v \models \tau_A(o)$. Hence $s \models c$ and $s \models \bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$, and therefore $app_o(s)$ is defined. Consider any state variable $a \in A$. By Lemma 3.4 and the assumption $v \models (EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))) \leftrightarrow a'$, the value of every state variable in s' matches the definition of $app_o(s)$. Hence $s' = app_o(s)$.

Assume $s' = app_o(s)$. Since s' is defined, $v \models \tau_A(o)$ and $v \models \bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$. By Lemma 3.4 $v \models EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$ if and only if $s' \models a$. □

Definition 3.43 *Define $\mathcal{R}_1(A, A') = \tau_A(o_1) \vee \cdots \vee \tau_A(o_n)$.*

The valuations that satisfy this formula do not uniquely determine which operator was applied because for a given state more than one operator may produce the same successor state. However, in such cases it does not matter which operator is applied, and when constructing a plan from the valuation any of the operators may be chosen arbitrarily.

It has been noticed that extending $\mathcal{R}_1(A, A')$ by 2-literal invariants (see Section 3.5) reduces runtimes of algorithms that test satisfiability. Note that invariants do not affect the set of models of a formula representing planning: any satisfying valuation of the original formula also satisfies the invariants because the values of variables describing the values of state variables at any time point corresponds to a state that is reachable from the initial state, and hence this valuation also satisfies any invariant.

3.6.3 Finding plans by satisfiability algorithms

We show how plans can be found by first translating succinct transition systems $\langle A, I, O, G\rangle$ into propositional formulae, and then finding satisfying valuations by a satisfiability algorithm.

In Section 3.6.1 we showed how operators can be described by propositional formulae over sets A and A' of propositional variables, the set A describing the values of the state variables in the state in which the operator is applied, and the set A' describing the values of the state variables in the successor state of that state.

For a fixed plan length n, we use sets $A^0, \ldots, A^n$ of variables to represent the values of state variables at different time points, with variables A^i representing the values at time i. In other words, a valuation of these propositional variables represents a sequence $s_0, \ldots, s_n$ of states. If $a \in A$ is a state variable, then we use the propositional variable a^i for representing the value of a at time point i.

Then we construct a formula so that the state s_0 is determined by I, the state s_n is determined by G, and the changes of state variables between any two consecutive states corresponds to the application of an operator.

Definition 3.44 *Let $\langle A, I, O, G\rangle$ be a deterministic transition system. Define $\iota^0 = \bigwedge\{a^0 | a \in A, I(a) = 1\} \cup \{\neg a^0 | a \in A, I(a) = 0\}$ for the initial state and G^n as the formula G with every variable $a \in A$ replaced by a^n. Define*

$$\Phi_n^{seq} = \iota^0 \wedge \mathcal{R}_1(A^0, A^1) \wedge \mathcal{R}_1(A^1, A^2) \wedge \cdots \wedge \mathcal{R}_1(A^{n-1}, A^n) \wedge G^n$$

where $A^i = \{a^i | a \in A\}$ for all $i \in \{0, \ldots, n\}$.

A plan can be found by using the formulae Φ_i^{seq} as follows. We start with plan length $i = 0$, test the satisfiability of Φ_i^{seq}, and depending on the result, either construct a plan (if Φ_i^{seq} is satisfiable), or increase i by one and repeat the previous steps, until a plan is found.

If there are no plans, it has to be somehow decided when to stop increasing i. An upper bound on plan length is $2^{|A|} - 1$ where A is the set of state variables but this upper bound does not provide a practical termination condition for this procedure. Some work on more practical termination conditions are cited in Section 3.10.

The construction of a plan from a valuation v that satisfies Φ_i^{seq} is straightforward. The plan has exactly i operators, and this plan is known to be the shortest one because the formula Φ_{i-1}^{seq} had already been determined to be unsatisfiable. First construct the execution $s_0, \ldots, s_i$ of the plan from v as follows. For all $j \in \{0, \ldots, i\}$ and $a \in A$, $s_j(a) = v(a_j)$. The plan has the form $o_1, \ldots, o_i$. Operator o_j for $j \in \{1, \ldots, i\}$ is identified by testing for all $o \in O$ whether $app_o(s_{j-1}) = s_j$. There may be several operators satisfying this condition, and any of them can be chosen.

Example 3.45 Let $A = \{a, b\}$. Let the state I satisfy $I \models a \wedge b$. Let $G = (a \wedge \neg b) \vee (\neg a \wedge b)$ and $o_1 = \langle \top, (a \rhd \neg a) \wedge (\neg a \rhd a)\rangle$ and $o_2 = \langle \top, (b \rhd \neg b) \wedge (\neg b \rhd b)\rangle$. The following formula is satisfiable if and only if $\langle A, I, \{o_1, o_2\}, G\rangle$ has a plan of length 3.

$$\begin{array}{l}
(a^0 \wedge b^0) \\
\wedge(((a^0 \leftrightarrow a^1) \wedge (b^0 \leftrightarrow \neg b^1)) \vee ((a^0 \leftrightarrow \neg a^1) \wedge (b^0 \leftrightarrow b^1))) \\
\wedge(((a^1 \leftrightarrow a^2) \wedge (b^1 \leftrightarrow \neg b^2)) \vee ((a^1 \leftrightarrow \neg a^2) \wedge (b^1 \leftrightarrow b^2))) \\
\wedge(((a^2 \leftrightarrow a^3) \wedge (b^2 \leftrightarrow \neg b^3)) \vee ((a^2 \leftrightarrow \neg a^3) \wedge (b^2 \leftrightarrow b^3))) \\
\wedge((a^3 \wedge \neg b^3) \vee (\neg a^3 \wedge b^3))
\end{array}$$

One of the valuations that satisfy the formula is the following.

	time i			
	0	1	2	3
a^i	1	0	0	0
b^i	1	1	0	1

This valuation corresponds to the plan that applies operator o_1 at time point 0, o_2 at time point 1, and o_2 at time point 2. There are also other satisfying valuations. The shortest plans have length 1 and respectively consist of the operators o_1 and o_2. ■

Example 3.46 Consider the following problem. There are two operators, one for rotating the values of bits abc one step right, and the other for inverting the values of all the bits. Consider reaching from the initial state 100 the goal state 001 with two actions. This is represented as the

following formula.

$$
\begin{array}{l}
(a^0 \wedge \neg b^0 \wedge \neg c^0) \\
\wedge(((a^0 \leftrightarrow b^1) \wedge (b^0 \leftrightarrow c^1) \wedge (c^0 \leftrightarrow a^1)) \vee ((\neg a^0 \leftrightarrow a_1) \wedge (\neg b^0 \leftrightarrow b^1) \wedge (\neg c^0 \leftrightarrow c^1))) \\
\wedge(((a^1 \leftrightarrow b^2) \wedge (b^1 \leftrightarrow c^2) \wedge (c^1 \leftrightarrow a^2)) \vee ((\neg a^1 \leftrightarrow a^2) \wedge (\neg b^1 \leftrightarrow b^2) \wedge (\neg c^1 \leftrightarrow c^2))) \\
\wedge(\neg a^2 \wedge \neg b^2 \wedge c^2)
\end{array}
$$

Since the literals describing the initial and the goal state must be true, we can replace occurrences of these state variables in the subformulae for operators by $\top$ and $\bot$.

$$
\begin{array}{l}
(a^0 \wedge \neg b^0 \wedge \neg c^0) \\
\wedge(((\top \leftrightarrow b^1) \wedge (\bot \leftrightarrow c^1) \wedge (\bot \leftrightarrow a^1)) \vee ((\neg\top \leftrightarrow a_1) \wedge (\neg\bot \leftrightarrow b^1) \wedge (\neg\bot \leftrightarrow c^1))) \\
\wedge(((a^1 \leftrightarrow \bot) \wedge (b^1 \leftrightarrow \top) \wedge (c^1 \leftrightarrow \bot)) \vee ((\neg a^1 \leftrightarrow \bot) \wedge (\neg b^1 \leftrightarrow \bot) \wedge (\neg c^1 \leftrightarrow \top))) \\
\wedge(\neg a^2 \wedge \neg b^2 \wedge c^2)
\end{array}
$$

After simplifying we have the following.

$$
\begin{array}{l}
(a^0 \wedge \neg b^0 \wedge \neg c^0) \\
\wedge((b^1 \wedge \neg c^1 \wedge \neg a^1) \vee (\neg a_1 \wedge b^1 \wedge c^1) \\
\wedge((\neg a^1 \wedge b^1 \wedge \neg c^1) \vee (a^1 \wedge b^1 \wedge \neg c^1)) \\
\wedge(\neg a^2 \wedge \neg b^2 \wedge c^2)
\end{array}
$$

The only way of satisfying this formula is to make the first disjuncts of both disjunctions true, that is, b^1 must be true and a^1 and c^1 must be false. The resulting valuation corresponds to taking the rotation action twice.

Consider the same problem but now with the goal state 101.

$$
\begin{array}{l}
(a^0 \wedge \neg b^0 \wedge \neg c^0) \\
\wedge(((a^0 \leftrightarrow b^1) \wedge (b^0 \leftrightarrow c^1) \wedge (c^0 \leftrightarrow a^1)) \vee ((\neg a^0 \leftrightarrow a_1) \wedge (\neg b^0 \leftrightarrow b^1) \wedge (\neg c^0 \leftrightarrow c^1))) \\
\wedge(((a^1 \leftrightarrow b^2) \wedge (b^1 \leftrightarrow c^2) \wedge (c^1 \leftrightarrow a^2)) \vee ((\neg a^1 \leftrightarrow a^2) \wedge (\neg b^1 \leftrightarrow b^2) \wedge (\neg c^1 \leftrightarrow c^2))) \\
\wedge(a^2 \wedge \neg b^2 \wedge c^2)
\end{array}
$$

We simplify again and get the following formula.

$$
\begin{array}{l}
(a^0 \wedge \neg b^0 \wedge \neg c^0) \\
\wedge((b^1 \wedge \neg c^1 \wedge \neg a^1) \vee (\neg a_1 \wedge b^1 \wedge c^1)) \\
\wedge((\neg a^1 \wedge b^1 \wedge c^1) \vee (\neg a^1 \wedge b^1 \wedge \neg c^1)) \\
\wedge(a^2 \wedge \neg b^2 \wedge c^2)
\end{array}
$$

Now there are two possible plans, to rotate first and then invert the values, or first invert and then rotate. These respectively correspond to making the first disjunct of the first disjunction and the second disjunct of the second disjunction true, or the second and the first disjunct. ■

3.7 Definitions of parallel plans

In this section we consider a more general notion of plans in which several operators can be applied simultaneously. This kind of plans are formalized as sequences of sets of operators. In such a plan the operators are partially ordered because there is no ordering on the operators taking place at the same time point. This notion of plans is useful for two reasons.

First, consider a number of operators that affect and depend on disjoint state variables so that they can be applied in any order. If there are n such operators, there are $n!$ plans that are equivalent in the sense that each leads to the same state. When a satisfiability algorithm shows that there is no plan of length n consisting of these operators, it has to show that none of the $n!$ plans reaches the goals. This may be combinatorially very difficult if n is high.

Second, when several operators can be applied simultaneously, it is not necessary to represent all intermediate states of the corresponding sequential plans: partially-ordered plans require less time points than the corresponding sequential plans. This reduces the number of propositional variables that are needed for representing the planning problem, which may make testing the satisfiability of these formulae much more efficient.

For presenting the results in the rest of this chapter we use a simpler definition of operators that makes it easier to distinguish between STRIPS operators and general conditional operators. An operator is defined as a triple $\langle p, e, c\rangle$ where p is a propositional formula, e is a set of literals (the unconditional effects) and c is a set of pairs $d \rhd g$ (the conditional effects) where d is a propositional formula and g is a set of literals. These operators in our standard definition are $\langle p, \bigwedge(e \cup \{d \rhd \bigwedge g | d \rhd g \in c\})\rangle$. For operators $\langle p, e, c\rangle$ we define $\mathit{EPC}_l(e, c) = \mathit{EPC}_l(\bigwedge(e \cup \{d \rhd \bigwedge g | d \rhd g \in c\}))$.

3.7.1 $\forall$-Step semantics

We formally present a semantics that generalizes the semantics used in most works on parallel plans, for example by Kautz and Selman [1996].

Earlier definitions of parallel plans have been based on the notion of *interference*. The parallel application of a set of operators is possible if the operators do not interfere. Lack of interference guarantees that the operators can be executed sequentially in any total order and that the terminal state is independent of the ordering. As shown in Theorem 3.49, non-interference and executability in any order coincide for STRIPS operators. Our definition of operators extends the definition of STRIPS operators considerably, and instead of non-interference in Definition 3.47 we adopt the more abstract and intuitive order-independence as the basic principle in the $\forall$-step semantics.

For the efficiency of plan search and plan validation it is important that the test whether a plan is executable and achieves the goals is tractable. For this reason we investigate the tractability of our general definition of $\forall$-step semantics and then identify restricted tractable classes of $\forall$-step plans. This investigation goes beyond earlier works like by Blum and Furst [1997] and Kautz and Selman [1996; 1999] which restrict to STRIPS operators.

Definition 3.47 ($\forall$-Step plans) *For a set of operators O and an initial state I,* a $\forall$-step plan for O and I *is a sequence $T = \langle S_0, \ldots, S_{l-1}\rangle$ of sets of operators for some $l \geq 0$ such that there is a sequence of states $s_0, \ldots, s_l$ (the execution of T) such that*

1. *$s_0 = I$, and*
2. *for all $i \in \{0, \ldots, l-1\}$ and every total ordering $o_1, \ldots, o_n$ of S_i, $\mathit{app}_{o_1;\ldots;o_n}(s_i)$ is defined and equals s_{i+1}.*

We show that this abstract definition yields the standard definition of parallel plans for STRIPS operators which requires that no operator falsifies the precondition of any other operator that is applied simultaneously.

Lemma 3.48 *Let $T = \langle S_0, \ldots, S_{l-1} \rangle$ be a $\forall$-step plan with execution $s_0, \ldots, s_l$. Then the following hold.*

1. *There is no $i \in \{0, \ldots, l-1\}$ and $\{\langle p, e, c\rangle, \langle p', e', c'\rangle\} \subseteq S_i$ and $a \in A$ such that $a \in e$ and $\neg a \in e'$.*
2. *$app_o(s_i)$ is defined for every $o \in S_i$.*

Proof: For (1) we derive a contradiction by assuming the opposite. Take an ordering of the operators such that $\langle p, e, c\rangle$ and $\langle p', e', c'\rangle$ are the last operators in this order. Hence $s_{i+1} \models \neg a$. But the ordering in which the two operators are the other way round leads to a state s'_{i+1} such that $s'_{i+1} \models a$. This contradicts the assumption that T is a $\forall$-step plan. Hence (1) holds.

Consider any operator $o \in S_i$ and any ordering in which o is the first operator. For the operators to be executable in this order, o has to be applicable in s_i. Therefore (2). □

For operators without conditional effects (including STRIPS operators) the above lemma means that for every set S_i of parallel operators $app_{S_i}(s_i)$ is defined. With conditional effects sequential execution in any order is sometimes possible even when simultaneous execution is not: consider for example $\{\langle \top, \emptyset, \{(\neg a \wedge \neg b) \rhd \{a, \neg b\}, b \rhd \{a\}\}\rangle, \langle \top, \emptyset, \{(\neg a \wedge \neg b) \rhd \{\neg a, b\}, a \rhd \{b\}\}\rangle\}$ executed in a state satisfying $\neg a \wedge \neg b$.

Theorem 3.49 *Let O be a set of STRIPS operators, I a state, and $T = \langle S_0, \ldots, S_{l-1} \rangle \in (2^O)^l$. Then T is a $\forall$-step plan for O and I if and only if there is a sequence of states $s_0, \ldots, s_l$ such that*

1. *$s_0 = I$,*
2. *$s_{i+1} = app_{S_i}(s_i)$ for all $i \in \{0, \ldots, l-1\}$, and*
3. *for no $i \in \{0, \ldots, l-1\}$ and two operators $\{\langle p, e, \emptyset\rangle, \langle p', e', \emptyset\rangle\} \subseteq S_i$ there is $m \in e$ such that $\overline{m}$ is one of the conjuncts of p'.*

Proof: We first prove the *only if* part. Since T is a $\forall$-step plan, it has an execution $s_0, \ldots, s_l$ as in Definition 3.47. We show that the three conditions on the right side of the equivalence are satisfied by this sequence of states.

By the definition of $\forall$-step plans, the first state of the execution is the initial state I. Hence we get (1).

By (1) of Lemma 3.48 for all $i \in \{0, \ldots, l-1\}$ the sets $E_i = [S_i]^{det}_{s_i} = \bigcup\{e | \langle p, e, \emptyset\rangle \in S_i\}$ are consistent. By (2) of the same lemma the preconditions of all operators in S_i are true in s_i. Hence the state $app_{S_i}(s_i)$ is defined. The changes made by any total ordering of S_i equal E_i because the effects of no operator in S_i override any effect of another operator in S_i. Therefore $s_{i+1} = app_{S_i}(s_i)$. This establishes (2).

For the sake of argument assume that there is literal m and $i \in \{0, \ldots, l-1\}$ so that $m \in e$ for some $\langle p, e, \emptyset\rangle \in S_i$ and $\overline{m}$ is a conjunct of the precondition p' of some other $\langle p', e', \emptyset\rangle \in S_i$. Then in every total ordering of the operators in which $\langle p, e, \emptyset\rangle$ immediately precedes $\langle p', e', \emptyset\rangle$ the latter would not be applicable. This, however, contradicts the definition of $\forall$-step plans. Therefore (3).

Then we prove the *if* part. Assume there is a sequence $s_0, \ldots, s_l$ satisfying (1), (2) and (3). We show that T and $s_0, \ldots, s_l$ satisfy Definition 3.47 of $\forall$-step plans.

That $s_0 = I$ is directly by our assumption (1).

We show that $app_{o_1;\ldots;o_n}(s_i) = app_{S_i}(s_i)$ for all $i \in \{0,\ldots,l-1\}$ and all total orderings $o_1,\ldots,o_n$ of S_i. Since $app_{S_i}(s_i)$ is defined, the precondition of every $o \in S_i$ is true in s_i and $E_i = \bigcup\{e|\langle p,e,\emptyset\rangle \in S_i\}$ is consistent. Take any total ordering $o_1,\ldots,o_n$ of S_i. By (3) no operator in S_i can disable another operator in S_i. Hence $app_{o_1;\ldots;o_n}(s_i)$ is defined. Since E_i is consistent, effects of no operator can be overridden by another operator in S_i. Hence $app_{S_i}(s_i) = s_{i+1} = app_{o_1;\ldots;o_n}(s_i)$. Since this holds for any total ordering of S_i, the definition of $\forall$-step plans is fulfilled. □

Testing whether a sequence of sets of STRIPS operators is a $\forall$-step plan can be done in polynomial time. A simple quadratic algorithm tests the operators pairwise for occurrences of a literal and its complement in the effects of the two operators and in the effect of one and in the precondition of the other. Computing the successor states is similarly polynomial time computation.

In the general case, however, the definition of $\forall$-step plans is computationally rather complex. The next theorem gives the justification for restricting to a narrow class of $\forall$-step plans in the following. The proof of the theorem shows that co-NP-hardness holds even when operators have no conditional effects. Hence the high complexity emerges merely from disjunctivity in operator preconditions.

Theorem 3.50 *Testing whether a sequence of sets of operators is a $\forall$-step plan is co-NP-hard.*

Proof: The proof is by reduction from TAUT. Let ϕ be any propositional formula. Let $A = \{a_1,\ldots,a_n\}$ be the set of propositional variables occurring in ϕ. The set of state variables is A. Let $o_z = \langle\phi,\emptyset,\emptyset\rangle$. Let $S = \{\langle\top,\{a_1\},\emptyset\rangle,\ldots,\langle\top,\{a_n\},\emptyset\rangle,o_z\}$. Let s and s' be states such that $s \not\models a$ and $s' \models a$ for all $a \in A$. We show that ϕ is a tautology if and only if $T = \langle S\rangle$ is a $\forall$-step plan for S and s.

Assume ϕ is a tautology. Now for any total ordering $o_0,\ldots,o_n$ of S the state $app_{o_0;\ldots,o_n}(s)$ is defined and equals s' because all preconditions are true in all states, and the set of effects of all operators is A (the set is consistent and making the effects true in s yields s'.) Hence T is a $\forall$-step plan.

Assume T is a $\forall$-step plan. Let v be any valuation. We show that $v \models \phi$. Let $S_v = \{\langle\top,\{a\},\emptyset\rangle|a \in A, v \models a\}$. The operators S can be ordered to $o_0,\ldots,o_n$ so that the operators $S_v = \{o_0,\ldots,o_k\}$ precede o_z and $S\backslash(S_v \cup \{o_z\})$ follow o_z. Since T is a $\forall$-step plan, $app_{o_0;\ldots;o_n}(s)$ is defined. Since also $app_{o_0;\ldots;o_k;o_z}(s)$ is defined, the precondition ϕ of o_z is true in $v = app_{o_0;\ldots;o_k}(s)$. Hence $v \models \phi$. Since this holds for any valuation v, ϕ is a tautology. □

Membership in co-NP is easy to show. There is a nondeterministic polynomial-time algorithm that can determine that a sequence of sets of operators is not a $\forall$-step plan. It first guesses an index i and a total ordering for the first $i-1$ steps and two total orderings for step i and then computes the two states that are reached by applying the operators in the first $i-1$ steps followed by one total ordering of step i. If the states differ or if not all operators are applicable, then the definition of $\forall$-step plans is not fulfilled.

To obtain a tractable notion of $\forall$-step plans for all operators we can generalize the notion of interference used for STRIPS operators to arbitrary operators. Lack of interference is a sufficient but not necessary condition for a set of operators to be executable in every order with the same results. First we define positive and negative occurrences of state variables $a \in A$ in a formula inductively as follows.

Definition 3.51 (Positive and negative occurrences) *We say that a state variable* a occurs positively in ϕ *if* $\mathit{positive}(a, \phi)$ *is true. Similarly,* a occurs negatively in ϕ *if* $\mathit{negative}(a, \phi)$ *is true.*

$$\begin{aligned}
\mathit{positive}(a, a) &= \textit{true, for all } a \in A \\
\mathit{positive}(a, b) &= \textit{false, for all } \{a, b\} \subseteq A \textit{ such that } a \neq b \\
\mathit{positive}(a, \phi \wedge \phi') &= \mathit{positive}(a, \phi) \textit{ or } \mathit{positive}(a, \phi') \\
\mathit{positive}(a, \phi \vee \phi') &= \mathit{positive}(a, \phi) \textit{ or } \mathit{positive}(a, \phi') \\
\mathit{positive}(a, \neg\phi) &= \mathit{negative}(a, \phi) \\
\\
\mathit{negative}(a, b) &= \textit{false, for all } \{a, b\} \subseteq A \\
\mathit{negative}(a, \phi \wedge \phi') &= \mathit{negative}(a, \phi) \textit{ or } \mathit{negative}(a, \phi') \\
\mathit{negative}(a, \phi \vee \phi') &= \mathit{negative}(a, \phi) \textit{ or } \mathit{negative}(a, \phi') \\
\mathit{negative}(a, \neg\phi) &= \mathit{positive}(a, \phi)
\end{aligned}$$

A state variable a occurs in ϕ *if it occurs positively or occurs negatively in* ϕ*.*

Below we also consider positive and negative occurrences of state variables in effects. A state variable a *occurs positively as an effect* in operator $\langle p, e, c\rangle$ if $a \in e$ or if there is $f \rhd d \in c$ so that $a \in d$. A state variable a *occurs negatively as an effect* in operator $\langle p, e, c\rangle$ if $\neg a \in e$ or there is $f \rhd d \in c$ such that $\neg a \in d$.

Definition 3.52 (Interference) *Let* A *be a set of state variables. Operators* $o = \langle p, e, c\rangle$ *and* $o' = \langle p', e', c'\rangle$ *over* A interfere *if there is* $a \in A$ *that*

1. *occurs positively as an effect in* o *and occurs in* f *for some* $f \rhd d \in c'$ *or occurs negatively in* p'*,*
2. *occurs positively as an effect in* o' *and occurs in* f *for some* $f \rhd d \in c$ *or occurs negatively in* p*,*
3. *occurs negatively as an effect in* o *and occurs in* f *for some* $f \rhd d \in c'$ *or occurs positively in* p'*, or*
4. *occurs negatively as an effect in* o' *and occurs in* f *for some* $f \rhd d \in c$ *or occurs positively in* p*.*

Proposition 3.53 *Testing whether two operators interfere can be done in polynomial time in the size of the operators.*

There are simple examples of valid $\forall$-step plans in which operators interfere according to the above definition. Hence the restriction to steps without interfering operators rules out many plans covered by the general definition (Definition 3.47.)

Example 3.54 Consider a set A of state variables and any set S of operators of the form

$$\langle \top, \emptyset, \{a \rhd \{\neg a\} | a \in A'\} \cup \{\neg a \rhd \{a\} | a \in A'\}\rangle$$

where A' is any subset of A (dependent on the operator.) Hence each operator reverses the values of a certain set of state variables. Executing the operators in any order results in the same state in every case. Hence $\langle S\rangle$ is a $\forall$-step plan according to Definition 3.47 but any two operators affecting the same state variable interfere. ■

Before formally connecting the notion of interference to plans that satisfy the $\forall$-step semantics we define a more relaxed notion of interference that is dependent on the state. In Section 3.8 we primarily use the state-independent notion of interference.

Definition 3.55 (Interference in a state) *Let A be a set of state variables. Operators $o = \langle p, e, c\rangle$ and $o' = \langle p', e', c'\rangle$ over A* interfere in a state s *if there is $a \in A$ so that*

1. *$a \in [o]_s^{det}$ and a occurs in d for some $d \rhd f \in c'$ or occurs negatively in p',*
2. *$a \in [o']_s^{det}$ and a occurs in d for some $d \rhd f \in c$ or occurs negatively in p,*
3. *$\neg a \in [o]_s^{det}$ and a occurs in d for some $d \rhd f \in c'$ or occurs positively in p', or*
4. *$\neg a \in [o']_s^{det}$ and a occurs in d for some $d \rhd f \in c$ or occurs positively in p.*

Lemma 3.56 *Let s be a state and o and o' two operators. If o and o' interfere in s, then o and o' interfere.*

Proof: Definition of interference has the form that o and o' interfere if there is an effect (conditional or unconditional) that fulfills some property. Interference in s is the same, except that a restriction to the subclass of effects active in s is made.

As an example we consider one case. Other cases are analogous. So assume o and o' interfere in s because (case (1)) there is $a \in A$ such that $a \in [o]_s^{det}$ and a occurs negatively in the precondition of o'. Now case (1) of the definition of interference is fulfilled because there is $a \in A$ that occurs negatively in the precondition of o'. □

Lemma 3.57 *Let s be a state and S a set of operators so that $app_S(s)$ is defined and no two operators interfere in s. Then $app_S(s) = app_{o_1;\ldots;o_n}(s)$ for any total ordering $o_1, \ldots, o_n$ of S.*

Proof: Let $o_1, \ldots, o_n$ be any total ordering of S. We prove by induction on the length of a prefix of $o_1, \ldots, o_n$ the following statement for all $i \in \{0, \ldots, n-1\}$ by induction on i: $s \models a$ if and only if $app_{o_1;\ldots;o_i}(s) \models a$ for all state variables a occurring in an antecedent of a conditional effect or a precondition of operators $o_{i+1}, \ldots, o_n$.

Base case $i = 0$: Trivial.

Inductive case $i \geq 1$: By the induction hypothesis the antecedents of conditional effects of o_i have the same value in s and in $app_{o_1;\ldots;o_{i-1}}(s)$, from which follows $[o_i]_s^{det} = [o_i]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$. Since o_i does not interfere in s with operators $o_{i+1}, \ldots, o_n$, no state variable occurring in $[o_i]_s^{det}$ occurs in an antecedent of a conditional effect or in the precondition of $o_{i+1}, \ldots, o_n$. Hence these state variables do not change. Since $[o_i]_s^{det} = [o_i]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$, this also holds when o_i is applied in $app_{o_1;\ldots;o_{i-1}}(s)$. This completes the induction proof.

Since $app_S(s)$ is defined, the precondition of every $o \in S$ is true in s and $[o]_s^{det}$ is consistent. Based on the fact we have established above, the precondition of every $o \in S$ is true also in $app_{o_1;\ldots;o_k}(s)$ and $[o]^{det}_{app_{o_1;\ldots;o_k}(s)}$ is consistent for any $\{o_1, \ldots, o_k\} \subseteq S\backslash\{o\}$. Hence any total ordering of the operators is executable. Based on the fact we have established above, $[o]_s^{det} = [o]^{det}_{app_{o_1;\ldots;o_k}(s)}$ for every $\{o_1, \ldots, o_k\} \subseteq S\backslash\{o\}$. Hence every operator causes the same changes no matter what the total ordering is. Since $app_S(s)$ is defined, no operator in S undoes the effects

of another operator. Hence the same state $s' = app_S(s)$ is reached in every case. □

Theorem 3.58 *Let I be a state, O a set of operators, and $T = \langle S_0, \ldots, S_{l-1} \rangle \in (2^O)^l$ such that there is a sequence $s_0, s_1, \ldots, s_l$ of states with $s_0 = I$ and $s_{i+1} = app_{S_i}(s_i)$ for all $i \in \{0, \ldots, l-1\}$. If for no $i \in \{0, \ldots, l-1\}$ and $\{o, o'\} \subseteq S_i$ such that $o \neq o'$ the operators o and o' interfere in s_i, then T is a $\forall$-step plan for O and I.*

Proof: Directly by Lemma 3.57. □

Theorem 3.59 *Let I be a state, O a set of operators, and $T = \langle S_0, \ldots, S_{l-1} \rangle \in (2^O)^l$ such that there is a sequence $s_0, s_1, \ldots, s_l$ of states with $s_0 = I$ and $s_{i+1} = app_{S_i}(s_i)$ for all $i \in \{0, \ldots, l-1\}$. If for no $i \in \{0, \ldots, l-1\}$ and $\{o, o'\} \subseteq S_i$ such that $o \neq o'$ the operators o and o' interfere, then T is a $\forall$-step plan for O and I.*

Proof: By Lemma 3.56 and Theorem 3.58. □

The state-dependent definition of interference in some cases allows more parallelism than the state-independent definition.

Example 3.60 Consider $S = \{\langle \top, \emptyset, \{a \rhd \{\neg b\}\}\rangle, \langle \top, \emptyset, \{b \rhd \{\neg a\}\}\rangle\}$. The operators interfere according to Definition 3.52. However, the operators do not interfere in states s such that $s \models \neg a \wedge \neg b$ because no effect is active. ■

A still more relaxed notion of interference that allows changing shared state variables as long as the preconditions do not become false nor the values of antecedents of conditional effects change leads to high complexity because states other than the current one have to be considered. Even if none of the operators change the values of antecedents of conditional effects or preconditions in the current state, they may do this in states reachable by applying another operator. For example, the operator $\langle a \vee b, \{c\}, \emptyset\rangle$ is not disabled by $\langle \top, \{\neg a\}, \emptyset\rangle$ nor $\langle \top, \{\neg b\}, \emptyset\rangle$ alone, but in states reached by one of these operators the other operator disables it.

The source of the high complexity of the general definition is that on different execution orders, all of which must result in the same state, a different sequence of intermediate states is visited, and it seems unavoidable to make these intermediate states explicit when reasoning about the executions.

3.7.2 Process semantics

The idea of the process semantics is that we only consider those $\forall$-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 $\forall$-step plan that leads to the same state.

As an example consider a set S of operators that are all initially applicable and no two operators interfere or have contradicting effects. 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 $\forall$-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. Hence under process semantics there is only one plan instead of $2^{|S|}$.

The idea of the process semantics was previously investigated in connection with Petri nets [Best and Devillers, 1987]. It can be seen as a way of canonizing $\forall$-step executions into a normal form in which each operator of the $\forall$-step plan occurs as early as possible. This canonical normal form is similar to the Foata normal form in the theory of Mazurkiewicz traces [Diekert and Métivier, 1997; Heljanko, 2001]. Bounded model checking with 1-safe Petri nets roughly corresponds to planning with STRIPS operators.

Definition 3.61 (Process plans) *For a set of operators O and an initial state I* a process plan for O and I *is a $\forall$-step plan $\langle S_0, \ldots, S_{l-1}\rangle$ for O and I with the execution $s_0, \ldots, s_l$ such that there is no $i \in \{1, \ldots, l-1\}$ and $o \in S_i$ so that $\langle S_0, \ldots, S_{i-1} \cup \{o\}, S_i \backslash \{o\}, \ldots, S_{l-1}\rangle$ is a $\forall$-step plan for O and I with the execution $s'_0, \ldots, s'_l$ such that $s_j = s'_j$ for all $j \in \{0, \ldots, i-1, i+1, \ldots, l\}$.*

Note that it is possible that $o \in S_{i-1}$, and when transforming a $\forall$-step plan to a corresponding process plan, the number of operators in the plan may decrease. It is possible to define an alternative process semantics so that moving an operator earlier is possible only if the total number of operators is preserved.

The important property of process semantics is that even though the additional condition reduces the number of valid plans, whenever there is a plan with t time steps under $\forall$-step semantics, there is also a plan with at most t time steps under process semantics that leads to the same final state. From any $\forall$-step plan a plan satisfying the process condition is obtained by repeatedly moving operators violating the condition one time point earlier.

Theorem 3.62 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system and $\langle S_0, \ldots, S_{l-1}\rangle$ a $\forall$-step plan for π. Then there is a process plan $\langle S'_0, \ldots, S'_{l-1}\rangle$ for π.*

Proof: Define a mapping ρ from plans to plans: plan $\rho(T)$ is obtained from T by moving one operator earlier according to Definition 3.61 if possible, and otherwise $\rho(T) = T$. Define the function $f(\langle S_0, \ldots, S_{l-1}\rangle) = \sum_{i=0}^{l-1}(i \cdot |S_i|)$. Note that $f(\rho(T)) < f(T)$ if $\rho(T) \neq T$. Since f can take only positive values, only finitely many moves are possible. When $f(\rho(T)) = f(T)$, T is a process plan. Hence a process plan is obtained after finitely many moves. □

Theorem 3.63 *Testing whether a sequence of sets of operators is a process plan is polynomial-time reducible to testing whether a sequence of sets of operators is a $\forall$-step plan.*

Proof: The definition of process plans gives a procedure for doing the test. Consider $\langle S_0, \ldots, S_{l-1}\rangle$. For every operator in $S_1 \cup \cdots \cup S_{l-1}$ we have to test the process condition. There are $|S_1| + \cdots + |S_{l-1}|$ such tests. □

We will later concentrate on $\forall$-step plans in which no two simultaneous operators interfere, and hence it is convenient to define a narrower class of process plans that is compatible with the narrower class of $\forall$-step plans.

Definition 3.64 (i-Process plans) *For a set of operators O and an initial state I* a process plan for O and I *is a $\forall$-step plan $\langle S_0, \ldots, S_{l-1}\rangle$ for O and I with the execution $s_0, \ldots, s_l$ such that there is*

no $i \in \{1, \ldots, l-1\}$ and $o \in S_i$ so that $\langle S_0, \ldots, S_{i-1} \cup \{o\}, S_i \backslash \{o\}, \ldots, S_{l-1} \rangle$ is a $\forall$-step plan for O and I with the execution $s'_0, \ldots, s'_l$ such that $s_j = s'_j$ for all $j \in \{0, \ldots, i-1, i+1, \ldots, l\}$ and additionally, for no $i \in \{0, \ldots, l-1\}$ and $\{o, o'\} \in S_i$ such that $o \neq o'$ the operators o and o' interfere.

3.7.3 $\exists$-Step semantics

We present a general formalization of a notion of parallel plans that was first considered by Dimopoulos et al. [1997].

Definition 3.65 ($\exists$-Step plans) *For a set O of operators and an initial state I,* a $\exists$-step plan *is $T = \langle S_0, \ldots, S_{l-1} \rangle \in (2^O)^l$ together with a sequence of states $s_0, \ldots, s_l$ (the execution of T) for some $l \geq 0$ such that*

1. *$s_0 = I$, and*
2. *for every $i \in \{0, \ldots, l-1\}$ there is a total ordering $o_1 < \ldots < o_n$ of S_i such that $s_{i+1} = app_{o_1;\ldots;o_n}(s_i)$.*

The difference to $\forall$-step semantics is that instead of requiring that each step S_i can be ordered to any total order, it is sufficient that there is one order that maps state s_i to s_{i+1}. Unlike in $\forall$-step semantics, the successor s_{i+1} of s_i is not uniquely determined solely by S_i, as the successor depends on the implicit ordering of S_i. Hence the definition has to make the execution $s_0, \ldots, s_l$ explicit. There are also other important technical differences between $\exists$-step and $\forall$-step semantics, most notably the fact that the properties given in Lemma 3.48 for $\forall$-step semantics do not hold for $\exists$-step semantics.

The more relaxed definition of $\exists$-step plans sometimes allows much more parallelism than the definition of $\forall$-step plans.

Example 3.66 Consider a row of n Russian dolls, each slightly bigger than the preceding one. We can nest all the dolls by putting the first inside the second, then the second inside the third, and so on, until every doll except the biggest one is inside another doll.

For four dolls this can be formalized as follows.

$$\begin{aligned} o_1 &= \langle \text{out1} \wedge \text{out2} \wedge \text{empty2}, \{\text{1in2}, \neg\text{out1}, \neg\text{empty2}\}, \emptyset \rangle \\ o_2 &= \langle \text{out2} \wedge \text{out3} \wedge \text{empty3}, \{\text{2in3}, \neg\text{out2}, \neg\text{empty3}\}, \emptyset \rangle \\ o_3 &= \langle \text{out3} \wedge \text{out4} \wedge \text{empty4}, \{\text{3in4}, \neg\text{out3}, \neg\text{empty4}\}, \emptyset \rangle \end{aligned}$$

The shortest $\forall$-step plan that nests the dolls is $\langle \{o_1\}, \{o_2\}, \{o_3\} \rangle$. The $\exists$-step plan $\langle \{o_1, o_2, o_3\} \rangle$ nests the dolls in one step. ■

Theorem 3.67 *(i) Each $\forall$-step plan is a $\exists$-step plan, and (ii) for every $\exists$-step plan T there is a $\forall$-step plan whose execution leads to the same final state as that of T.*

Proof: (i) Consider a $\forall$-step plan $T = \langle S_0, \ldots, S_{l-1} \rangle$. Any total ordering of $S_i, i \in \{0, \ldots, l-1\}$ takes state s_i to the same s_{i+1}. Hence, T is a $\exists$-step plan. (ii) For a $\exists$-step plan $T = \langle S_0, \ldots, S_{l-1} \rangle$, a $\forall$-step plan whose execution leads to the same final state as that of T is $\{o_1^0\}, \ldots, \{o_{n_0}^0\}, \ldots, \{o_1^{l-1}\}$, where for every $i \in \{0, \ldots, l-1\}$, the sequence $\{o_1^i\}, \ldots, \{o_{n_i}^i\}$ is a total ordering of S_i given

by Condition 2 of Definition 3.65. □

Next we identify restricted intractable and tractable classes of ∃-step plans.

Theorem 3.68 *Let O be a set of operators and I a state. Testing whether $T = \langle S_0, \ldots, S_{l-1}\rangle \in (2^O)^l$ is a ∃-step plan for O and I with some execution $s_0, \ldots, s_l$ is NP-hard, even when the set of atomic effects of operators in S_i for every $i \in \{0, \ldots, l-1\}$ is consistent.*

Proof: By reduction from SAT. Let ϕ be any propositional formula. Let A be the set of propositional variables occurring in ϕ. Let s and s' be states such that $s \not\models a$ for all $a \in A$ and $s' \models a$ for all $a \in A$. We claim that ϕ is satisfiable if and only if $\langle S\rangle$ with $S = \{\langle \top, \{a\}, \emptyset\rangle | a \in A\} \cup \{\langle \phi, \emptyset, \emptyset\rangle\}$ is a ∃-step plan with execution s, s'.

So assume ϕ is satisfiable and $v : A \to \{0, 1\}$ is a valuation satisfying ϕ. Then for any total order on S such that exactly the operators $S_v = \{\langle \top, \{a\}, \emptyset\rangle | a \in A, v(a) = 1\}$ precede $o_\phi = \langle \phi, \emptyset, \emptyset\rangle$ satisfies the definition of ∃-step plans because executing S_v produces the state/valuation v that satisfies the precondition of o_ϕ.

Assume $\langle S\rangle$ is a ∃-step plan. Hence there is a total ordering $o_1, \ldots, o_n$ of S such that $app_{o_1;\ldots;o_n}(s)$ is defined. Hence $app_{o_1;\ldots;o_j}(s) \models \phi$ where $o_1, \ldots, o_j$ are the operators preceding o_ϕ. Therefore ϕ is satisfiable. □

The preceding theorem (Theorem 3.68) and the following (Theorem 3.69) can be strengthened so that all operators in S_i are applicable in s_i. This shows that our later restriction to sets S_i so that $app_{S_i}(s_i)$ is defined does not directly reduce complexity.

From the above proof we see that NP-hardness holds even when there are no conditional effects and the effects of the operators are not in conflict with each other. However, the proof assumes disjunctivity in preconditions because ϕ may be any formula. The question arises if the problem is easier for STRIPS operators.

Theorem 3.69 *Let O be a set of STRIPS operators and I a state. Testing whether $T = \langle S_0, \ldots, S_{l-1}\rangle \in (2^O)^l$ is a ∃-step plan for O and I with some execution $s_0, \ldots, s_l$ is NP-hard.*

Proof: We reduce the NP-complete problem SAT to testing whether a sequence of sets of operators is a ∃-step plan. Let C be a set of clauses, $n = |C|$ and P the set of propositional variables occurring in C. Assign an index $i \in \{1, \ldots, n\}$ to each clause. The state variables are $A = \{c_1, \ldots, c_n\} \cup \{U_a | a \in P\}$. Define

$$
\begin{array}{rcl}
o_a^+ & = & \langle U_a, \{\neg U_a, c_{i_1^{a+}}, \ldots, c_{i_{m_{a+}}^{a+}}\}, \emptyset\rangle \text{ for all } a \in P, \\
 & & \text{where } i_1^{a+}, \ldots, i_{m_{a+}}^{a+} \text{ are the indices of clauses in which } a \text{ occurs positively} \\
o_a^- & = & \langle U_a, \{\neg U_a, c_{i_1^{a-}}, \ldots, c_{i_{m_{a-}}^{a-}}\}, \emptyset\rangle \text{ for all } a \in P, \\
 & & \text{where } i_1^{a-}, \ldots, i_{m_{a-}}^{a-} \text{ are the indices of clauses in which } a \text{ occurs negatively} \\
o_m & = & \langle c_1 \wedge \cdots \wedge c_n, \{U_a | a \in P\}, \emptyset\rangle, \text{ and} \\
S & = & \{o_a^+ | a \in A\} \cup \{o_a^- | a \in P\} \cup \{o_m\}.
\end{array}
$$

Let s and s' be states such that $s \models \neg c_1 \wedge \cdots \wedge \neg c_n \wedge \bigwedge_{a \in P} U_a$ and $s' \models c_1 \wedge \cdots \wedge c_n \wedge \bigwedge_{a \in P} \neg U_a$. We show that $\langle S\rangle$ is a ∃-step plan with execution s, s' if and only if C is satisfiable. Assume that $v : P \to \{0, 1\}$ is a valuation that satisfies C. Take any total ordering $<$ of S such that for all

```
1:  procedure linearize(s,S)
2:  while S ≠ ∅ do
3:    if there is o = ⟨p, e, ∅⟩ ∈ S
4:      such that s ⊨ p and e ∩ {l̄|l ∈ p'} = ∅ for all ⟨p', e', ∅⟩ ∈ S\{o}
5:    then S := S\{o}
6:    else return false;
7:    s := app_o(s);
8:  end while
9:  return true;
```

Figure 3.4: Algorithm for testing whether a set of non-conflicting STRIPS operators can be linearized

$a \in P$, $o_a^+ < o_m$ iff $v(a) = 1$ and $o_a^- < o_m$ iff $v(a) = 0$. Applying the operators preceding o_m makes the state variables $c_1, \ldots, c_n$ true (because v is a valuation that satisfies C) and the state variables $U_a, a \in P$ false. Now o_m is applicable and its application makes all $U_a, a \in P$ true again. Then the remaining operators are applicable, making every $U_a, a \in P$ false. Hence that total ordering satisfies the definition of $\exists$-step plans for $\langle S \rangle$ with execution s, s'.

For the other direction, assume that $\langle S \rangle$ is a $\exists$-step plan with execution s, s' which means that the operators can be applied in some order $<$ to obtain s' from s. Since for every $a \in P$ the operators o_a^+ and o_a^- have U_a as the precondition and both make U_a false and only o_m can make U_a true, it must be that $o_a^+ < o_m < o_a^-$ or $o_a^- < o_m < o_a^+$. Define $v : P \to \{0, 1\}$ by $v(a) = 1$ iff $o_a^+ < o_m$. For o_m to be applicable $c_1 \wedge \cdots \wedge c_n$ must be true. Hence the operators applied before o_m correspond to a valuation v that satisfies every clause in C. Therefore $v \models C$. □

Restrictions of the previous two theorems separately do not yield tractability, but together they do.

Theorem 3.70 *Let O be a set of STRIPS operators and I a state. Testing whether $T = \langle S_0, \ldots, S_{l-1} \rangle \in (2^O)^l$ with no S_i containing operators with mutually conflicting effects, is a $\exists$-step plan for O and I with some execution $s_0, \ldots, s_l$ is polynomial time.*

Proof: Since no two simultaneous operators have effects that conflict each other the execution of the plan – if one exists – is unambiguously determined by the sets of effects of operators of $S_0, \ldots, S_{l-1}$: $s_0 = I$ and $s_{i+1} = app_{\{\langle \top, e, \emptyset \rangle | \langle p, e, \emptyset \rangle \}}(s_i)$ for all $i \in \{0, \ldots, l-1\}$. The question that we must answer in polynomial time is whether the operators at each time point can be ordered so that the precondition is satisfied when an operator is applied.

The test is performed by the procedure calls linearize(s_i, S_i) for all $i \in \{0, \ldots, l-1\}$. This procedure is given in Figure 3.4. It runs in polynomial time in the size of S because the number of iterations of the *while* loop is bounded by the cardinality of S and all the computation in one iteration is polynomial time in the size of S. We show that the procedure returns *true* if and only if a linearization of S exists.

Assume linearize(s,S) returns *true*. Hence there is a sequence of states $s'_0, \ldots, s'_{|S|}$ and a sequence $o'_0, \ldots, o'_{|S|-1}$ of operators such that $s'_0 = s$ and $s'_{i+1} = app_{o'_i}(s'_i)$ for every $i \in \{0, \ldots, |S| - 1\}$. Hence $app_{o'_0;\ldots;o'_{|S|-1}}(s) = app_S(s)$ which satisfies the conditions a set S has to satisfy in the definition of $\exists$-step plans.

Assume linearize(s,S) returns *false*. We show that no linearization exists. Since *false* is returned, for every $\langle p, e, \emptyset\rangle \in S' \subseteq S$ either $s' \not\models p$ (where S' and s' are the last values the variables S and s have obtained) or e falsifies the precondition of at least one of the operators in $S'\backslash\{\langle p, e, \emptyset\rangle\}$. Let $o_1, \ldots, o_n$ be any total ordering of S. We show that $app_{o_1;\ldots;o_n}(s)$ is not defined, and hence the total ordering does not satisfy Definition 3.65.

Take the operator $o_i = \langle p_i, e_i, \emptyset\rangle \in S'$ that comes earliest in the ordering $o_1, \ldots, o_n$.

If $s'_i = app_{o_1;\ldots;o_{i-1}}(s)$ is not defined (because the precondition of one of the operators is false when the operator is applied), then also $app_{o_1;\ldots;o_n}(s)$ is not defined. So assume $s'_i = app_{o_1;\ldots;o_{i-1}}(s)$ is defined.

Since linearize(s,S) returns *false*, either $s' \not\models p_i$ or o_i falsifies the precondition of at least one of $S'\backslash\{o_i\}$.

In the first case, as none of the operators in $S\backslash S'$ falsifies any literal in the precondition of any operator in S', it must be that $s \not\models p_i$. Since $s' \not\models p_i$, there is at least one conjunct (a literal) of p_i that is not made true by any operator in $S\backslash S'$. Since $\{o_1, \ldots, o_{i-1}\} \subseteq S\backslash S'$, this literal is also not true in s'_i and hence $s'_i \not\models p_i$.

In the second case, as o_i is the first operator of S_i in the ordering, one of the literals in the precondition of at least one operator in $S'\backslash\{o_i\}$ becomes false when o_i is applied. Since the operators in S are pairwise non-conflicting, there is no operator that could make this literal and precondition true again (here we use the assumption that S consists of STRIPS operators.) Hence $app_{o_1;\ldots;o_n}(s)$ is not defined, and the definition of $\exists$-step plans is not satisfied. □

To obtain a tractable notion of $\exists$-step plans for operators in general we introduce, similarly to $\forall$-step semantics, a syntactic notion characterizing dependencies between operators that leads to a simple graph-theoretic test for plans.

Our quest for tractable notions of $\exists$-step plans is motivated by the need to effectively encode the planning problem in the propositional logic (Section 3.8.) Even though Theorem 3.70 allows $\exists$-step plans in which the preconditions of some of the operators in S_i are false in s_i, we will not consider encodings of this generality. The reason for this is that there seem to be no such simple encodings of the semantics in the propositional logic that would not involve making the implicit intermediate states explicit. Making the intermediate states explicit would directly contradict the motivation of studying parallel encodings in the first place.

Definition 3.71 (Affect) *Let A be a set of state variables and $o = \langle p, e, c\rangle$ and $o' = \langle p', e', c'\rangle$ operators over A. Then o* affects *o' if there is $a \in A$ such that*

1. *$a \in (e \cup \bigcup\{d | f \rhd d \in c\})$ and a occurs in f for some $f \rhd d \in c'$ or occurs negatively in p', or*

2. *$\neg a \in e$ or $\neg a \in d$ for some $f \rhd d \in c$ and a occurs in f for some $f \rhd d \in c'$ or occurs positively in p'.*

This is like Definition 3.52 but considers only one direction of interference: if o and o' interfere, then either o affects o' or o' affects o.

Lemma 3.72 *Let $o_1 < \cdots < o_n$ be an ordering of a set S of operators so that if $o < o'$ then o does not affect o'. Let s be a state so that $s \models p$ and $[o]^{det}_s$ is consistent for every $\langle p, e, c\rangle \in S$. Then the following hold.*

1. $app_{o_1;\ldots;o_i}(s) \models p_j$ *for every* $i \in \{1,\ldots,n-1\}$ *and* $j \in \{i+1,\ldots,n\}$ *where* p_j *is the precondition of* o_j.

2. $[o_j]^{det}_s = [o_j]^{det}_{app_{o_1;\ldots;o_i}(s)}$ *for every* $i \in \{1,\ldots,n-1\}$ *and* $j \in \{i+1,\ldots,n\}$.

3. *For every* $i \in \{1,\ldots,n\}$, *if* $app_{\{o_1,\ldots,o_i\}}(s)$ *is defined, then* $app_{o_1;\ldots;o_i}(s) = app_{\{o_1,\ldots,o_i\}}(s)$.

Proof: By induction on i.

Base case $i = 0$: Trivial.

Inductive case $i \geq 1$: First we note that $app_{o_1;\ldots;o_i}(s)$ is defined because by the induction hypothesis for case (1) the precondition of o_i is true in $app_{o_1;\ldots;o_{i-1}}(s)$, and by the assumptions and the induction hypothesis for case (2) $[o_i]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$ is consistent.

Now consider any $j \in \{i+1,\ldots,n\}$.

Case (1): By the induction hypothesis $app_{o_1;\ldots;o_{i-1}}(s) \models p_j$. Since o_i does not affect o_j, o_i does not falsify p_j. Hence $app_{o_1;\ldots;o_i}(s) \models p_j$.

Case (2): By the induction hypothesis $[o_j]^{det}_s = [o_j]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$. Since o_i does not affect o_j, o_i does not change the value of any state variable occurring in the antecedent of a conditional effect of o_j. Hence $[o_j]^{det}_s = [o_j]^{det}_{app_{o_1;\ldots;o_i}(s)}$.

Case (3): By the induction hypothesis, if $app_{\{o_1,\ldots,o_{i-1}\}}(s)$ is defined, then $app_{o_1;\ldots;o_{i-1}}(s) = app_{\{o_1,\ldots,o_{i-1}\}}(s)$. So assume also $app_{\{o_1,\ldots,o_i\}}(s)$ is defined, that is, $[o_i]^{det}_s$ does not contradict $[\{o_1,\ldots,o_{i-1}\}]^{det}_s$. By (2) $[o_i]^{det}_s = [o_i]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$. Since the effects of o_i do not override the effects of any operator earlier in the sequence, we get $app_{o_1;\ldots;o_i}(s) = app_{\{o_1,\ldots,o_i\}}(s)$. □

Theorem 3.73 *Let* O *be a set of operators,* I *a state,* $T = \langle S_0,\ldots,S_{l-1}\rangle \in (2^O)^l$, *and* $s_0,\ldots,s_l$ *a sequence of states. If*

1. $s_0 = I$,

2. *for every* $i \in \{0,\ldots,l-1\}$ *there is a total ordering* $<$ *of* S_i *such that if* $o < o'$ *then* o *does not affect* o', *and*

3. $s_{i+1} = app_{S_i}(s_i)$ *for every* $i \in \{0,\ldots,l-1\}$,

then T *is a* $\exists$*-step plan for* O *and* I.

Proof: Since by assumption $app_{S_i}(s_i)$ is defined, the preconditions of all operators in S_i are true in s_i and $[o]^{det}_{s_i}$ is consistent for every $o \in S_i$. Hence the assumptions of Lemma 3.72 are satisfied and by (3) $app_{o_1;\ldots;o_n}(s_i) = app_{S_i}(s_i)$ for some total ordering $o_1,\ldots,o_n$ of S_i. □

For STRIPS operators the subclass of $\exists$-step plans definable by using the notion of *affects* in Theorem 3.73 is not very restrictive. In comparison to arbitrary $\exists$-step plans, the only restrictions are that sets S of simultaneous operators have no contradicting effects and all operators are applicable in the current state s, or in other words, that $app_S(s)$ is defined. This is stated in the following theorem.

Theorem 3.74 *Let* $\pi = \langle A, I, O, G\rangle$ *be a transition system so that every operator in* O *is a STRIPS operator and let* $T = \langle S_0,\ldots,S_{l-1}\rangle$ *be a* $\exists$*-step plan for* π *with execution* $s_0,\ldots,s_l$ *so*

that $s_0 = I$ *and* $s_{i+1} = app_{S_i}(s_i)$ *for every* $i \in \{0, \ldots, l-1\}$. *Then for every* $i \in \{0, \ldots, l-1\}$ *there is a total ordering* $<$ *of* S_i *such that if* $o < o'$ *then* o *does not affect* o'.

Proof: For STRIPS operators an operator o *affects* o' if and only if o has an effect m and $\overline{m}$ is one of the conjuncts in the precondition of o'. The result follows from the proof of Theorem 3.70. The procedure *linearize* repeatedly selects an operator that does not affect any of the remaining operators. □

Even though the class of $\exists$-step plans based on *affects* is narrower than the class sanctioned by Definition 3.65, much more parallelism is still possible in comparison to the class of $\forall$-step plans satisfying the non-interference condition. For instance, nesting of Russian dolls in Example 3.66 belongs to this class.

Similarly to the notion of interference in a state (Definition 3.55), we could define a state-specific notion of *affects*. This would lead to a slightly more relaxed but still efficient test of whether $\exists$-step semantics is fulfilled.

It is possible to combine $\exists$-step semantics with processes, but we leave this to future work.

3.8 Planning as satisfiability with parallel plans

Planning as satisfiability was introduced by Kautz and Selman [1992]. In addition to being a powerful approach to planning, it is also the basis of *bounded model checking* [Biere *et al.*, 1999][2].

In this section we present encodings of the different semantics of parallel plans in the propositional logic. A basic assumption in all these encodings is that for sets S of simultaneous operators applied in state s the state $app_S(s)$ is defined, that is, all the preconditions are true in s and the set of active effects of the operators is consistent. Given this assumption, the encodings of all the semantics share a common part which is described next.

3.8.1 The base encoding

Planning can be performed by propositional satisfiability testing as follows. Produce formulae $\phi_0, \phi_1, \phi_2, \ldots$ such that ϕ_l is satisfiable iff there is a plan of length l. The formulae are tested for satisfiability in the order of increasing plan length, and from the satisfying assignment that is found a plan is constructed. The encodings of the different semantics for parallel plans differ only in the formulae that restrict the simultaneous application of operators. Next we describe the part of the encodings shared by all the semantics.

For the transition system $\pi = \langle A, I, O, G \rangle$ let the (Boolean) state variables be $A = \{a^1, \ldots, a^n\}$ and the operators $O = \{o^1, \ldots, o^m\}$. For every state variable $a \in A$ we have the propositional variables a_t which express the value of a at different time points $t \in \{0, \ldots, l\}$. Similarly, for every operator $o \in O$ we have o_t for expressing whether o is applied at $t \in \{0, \ldots, l-1\}$. For formulae ϕ about the values of the state variables we denote the formula with all state variables subscripted with the index to a time point t by ϕ_t.

Given a transition system $\pi = \langle A, I, O, G \rangle$, a formula $\Phi_{\pi,l}$ is generated to answer the following question. Is there an execution of a sequence of l sets of operators from O that reaches a state

[2]Bounded model checking was developed at CMU after Alessandro Cimatti gave a seminar talk on the techniques used in the 1998 AIPS planning competition in which the BLACKBOX planner by Kautz and Selman participated [Cimatti, 2003].

satisfying G from the initial state I? The formula $\Phi_{\pi,l}$ is conjunction of I_0 (formula describing the initial state with propositional variables subscripted by time point 0), G_l, and the formulae described below, instantiated with all $t \in \{0,\ldots,l-1\}$.

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

$$o_t \rightarrow p_t \tag{3.1}$$

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

$$o_t \rightarrow e_{t+1} \tag{3.2}$$

Here we view sets e of literals as conjunctions of literals. For every $f \rhd d \in c$ the effects d will be true if f is true at the preceding time point.

$$(o_t \wedge f_t) \rightarrow d_{t+1} \tag{3.3}$$

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 from true to false,

$$(a_t \wedge \neg a_{t+1}) \rightarrow ((o_t^1 \wedge (\mathit{EPC}_{\neg a}(e^1, c^1))_t) \vee \cdots \vee (o_t^m \wedge (\mathit{EPC}_{\neg a}(e^m, c^m))_t)) \tag{3.4}$$

where e^i, c^i are the unconditional and conditional effects of operator i. The other expresses change of a from false to true,

$$(\neg a_t \wedge a_{t+1}) \rightarrow ((o_t^1 \wedge (\mathit{EPC}_{a}(e^1, c^1))_t) \vee \cdots \vee (o_t^m \wedge (\mathit{EPC}_{a}(e^m, c^m))_t)). \tag{3.5}$$

The formulae $\Phi_{\pi,l}$, just like the definition of $\mathit{app}_S(s)$, allow sets of operators in parallel that do not correspond to any sequential plan. For example, the operators $\langle a, \{\neg b\}, \emptyset\rangle$ and $\langle b, \{\neg a\}, \emptyset\rangle$ may be executed simultaneously resulting in a state satisfying $\neg a \wedge \neg b$, even though this state is not reachable by the two operators sequentially. Plans following the three semantics of parallel plans can always be executed sequentially. Further formulae that are discussed in the next sections are needed for capturing the three semantics.

Theorem 3.75 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system. Then there is $T = \langle S_0, \ldots, S_{l-1}\rangle \in (2^O)^l$ so that $s_0, \ldots, s_l$ are states so that $I = s_0$, $s_l \models G$, and $s_{i+1} = \mathit{app}_{S_i}(s_i)$ for all $i \in \{0,\ldots,l-1\}$ if and only if there is a valuation satisfying the formula $\Phi_{\pi,l}$.*

Proof: For the proof from left to right, we construct a valuation v as follows. For all $i \in \{0,\ldots,l\}$ and all state variables $a \in A$ define $v(a_i) = s_i(a)$. For all $i \in \{0,\ldots,l-1\}$ and all operators $o \in O$ define $v(o_i) = 1$ iff $o \in S_i$.

We show that $v \models \Phi_{\pi,l}$. From this it directly follows that $v \models I_0 \wedge G_l$. It remains to show satisfaction of instances of the schemata (3.1), (3.2), (3.3), (3.4) and (3.5).

1. Consider any $i \in \{0,\ldots,l-1\}$ and $o = \langle p, e, c\rangle \in O$. If $o \notin S_i$, then $v \not\models o_i$ and immediately $v \models o_i \rightarrow p_i$ (Formula 3.1). So assume $o \in S_i$. By assumption s_i is a state such that $\mathit{app}_{S_i}(s_i)$ is defined. Hence the precondition of o is true in s_i. Hence $v \models o_i \rightarrow p_i$ (Formula 3.1).

2. Consider any $i \in \{0,\ldots,l-1\}$ and $o = \langle p,e,c\rangle \in O$. If $o \notin S_i$, then $v \not\models o_i$ and immediately $v \models o_i \rightarrow e_{i+1}$ (Formula 3.2). So assume $o \in S_i$. As $o \in S_i$, the unconditional effects e of o are true in $s_{i+1} = app_{S_i}(s_i)$. Hence $v \models o_i \rightarrow e_{i+1}$ (Formula 3.2).

3. Consider any $i \in \{0,\ldots,l-1\}$ and $o = \langle p,e,c\rangle \in O$ and $f \rhd d \in c$. If $o \notin S_i$, then $v \not\models o_i$ and immediately $v \models (o_i \wedge f_i) \rightarrow e_{i+1}$ (Formula 3.2). So assume $o \in S_i$. Now $v \models (o_i \wedge f_i) \rightarrow d_{i+1}$ (Formula 3.3) because if $s_i \models f$ then the literals d are active effects and are true in s_{i+1} and consequently $v \models d_{i+1}$.

4. Consider any $i \in \{0,\ldots,l-1\}$ and $a \in A$. According to the definition of $s_{i+1} = app_{S_i}(s_i)$, a can be true in s_i and false in s_{i+1} only if $\neg a \in [o]^{det}_{s_i}$ for some $o \in S_i$. By Lemma 3.3 $\neg a \in [o]^{det}_{s_i}$ if and only if $s_i \models EPC_{\neg a}(e,c)$, where $o = \langle p,e,c\rangle$. So if the antecedent of $(a_i \wedge \neg a_{i+1}) \rightarrow ((o_i^1 \wedge (EPC_{\neg a}(e^1,c^1))_i) \vee \cdots \vee (o_i^m \wedge (EPC_{\neg a}(e^m,c^m))_i))$ is true, then one of the disjuncts of the consequent is true, where $O = \{o^1,\ldots,o^m\}$ and e^i, c^i are the effects of o^i. This yields the truth of instances of Formula 3.4.

   Proof for Formula 3.5 is analogous.

For the proof from right to left, assume v is a valuation satisfying the formula $\Phi_{\pi,l}$. We construct a plan $\langle S_0,\ldots,S_{l-1}\rangle$ and a corresponding execution $s_0,\ldots,s_l$.

Define for all $i \in \{0,\ldots,l\}$ the state s_i as the valuation of A such that $s_i(a) = v(a_i)$ for every $a \in A$. Define $S_j = \{o \in O | v(o_j) = 1\}$ for all $j \in \{0,\ldots,l-1\}$.

Obviously $I = s_0$ and $s_l \models G$. We show that $s_{i+1} = app_{S_i}(s_i)$ for all $i \in \{0,\ldots,l-1\}$.

The precondition p of every operator $o \in S_i$ is true in s_i because $v \models o_i$ and $v \models o_i \rightarrow p_i \in \Phi_{\pi,l}$ (Formula 3.1).

$s_{i+1} \models [o]^{det}_{s_i}$ for every $o \in S_i$ because $v \models o_i$ and $v \models o_i \rightarrow e_{i+1} \in \Phi_{\pi,l}$ for the unconditional effects e of o (Formula 3.2) and $v \models (o_i \wedge f_i) \rightarrow d_{i+1}$ for conditional effects $f \rhd d$ of o. This also means that $[S_i]^{det}_{s_i}$ is consistent and $app_{S_i}(s_i)$ is defined.

For state variables a not occurring in $[S_i]^{det}_{s_i}$ we have to show that $s_i(a) = s_{i+1}(a)$. Since a does not occur in $[S_i]^{det}_{s_i}$, for every $o \in \{o^1,\ldots,o^m\} = O$ either $o \notin S_i$ or both $a \notin [o]^{det}_{s_i}$ and $\neg a \notin [o]^{det}_{s_i}$. Hence either $v \not\models o_i$ or (by Lemma 3.3) $v \models \neg(EPC_a(o))_i \wedge \neg(EPC_{\neg a}(e,c))_i$where $o = \langle p,e,c\rangle$. This together with the assumptions that $v \models (a_i \wedge \neg a_{i+1}) \rightarrow ((o_i^1 \wedge (EPC_{\neg a}(e^1,c^1))_i) \vee \cdots \vee (o_i^m \wedge (EPC_{\neg a}(e^m,c^m))_i))$ (Formula 3.4) and $v \models (\neg a_i \wedge a_{i+1}) \rightarrow ((o_i^1 \wedge (EPC_a(e^1,c^1))_i) \vee \cdots \vee (o_i^m \wedge (EPC_a(e^m,c^m))_i))$ (Formula 3.5) implies $v \models (a_i \rightarrow a_{i+1}) \wedge (\neg a_i \rightarrow \neg a_{i+1})$. Therefore every $a \in A$ not occurring in $[S_i]^{det}_{s_i}$ remains unchanged. Hence $s_{i+1} = app_{S_i}(s_i)$. □

Proposition 3.76 *The size of the formula $\Phi_{\pi,l}$ is linear in l and the size of π.*

Theorem 3.75 says that a sequence of operators fulfilling certain conditions exists if and only if a given formula is satisfiable. The theorems connecting certain formulae to certain notions of plans (Theorems 3.77, 3.80, 3.85, 3.86, 3.87) provide an implication only in one direction: whenever the formula for a given value of parameter l is satisfiable, a plan of l time points exists. The other direction is missing because the formulae in general only approximate the respective semantics and there is no guarantee that the formula for a given l is satisfiable when a plan with l time points exists. However, the formula with some higher value of l is satisfiable. This follows from the fact that whenever a $\forall$-step or $\exists$-step plan $\langle S_0,\ldots,S_{l-1}\rangle$ with $n = |S_0| + \cdots + |S_{l-1}|$ occurrences

of operators exists, there is a plan consisting of n singleton sets, and the corresponding formulae $\Phi_{\pi,n} \wedge \Phi^x_{O,n}$ are satisfiable. The formulae $\Phi^x_{O,n}$ encode the parallel semantics x for formulae O.

An exact match between the $\forall$-step semantics and its encodings holds for transition systems with STRIPS operators only (Theorem 3.78.)

The implications of the approximative nature of the $\forall$-step semantics encodings for process semantics are more serious. For STRIPS operators the encodings for process semantics are exact: the formula for n time points is satisfiable if and only if a process plan of length n exists. However, in the general case the inexactness of the underlying $\forall$-step encoding leads to a mismatch between process semantics and the formulae. The problem is that the movement of an operator to an earlier time point may be prevented by the too strict $\forall$-step semantics encoding even when it is allowed by Definition 3.47. Hence the process semantics has to be understood in relation to particular classes of $\forall$-step plans: an operator has to be moved earlier only if there is a corresponding $\forall$-step plan *belonging to the subclass in question*, for example, the subclass of $\forall$-step plans in which no two parallel operators interfere. This is the reason why we introduced the notion of i-process plans in Definition 3.64.

In planning as satisfiability it is often useful to use constraints that do not affect the set of satisfying valuations but help pruning the set of incomplete solutions encountered during satisfiability testing and therefore 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 [Blum and Furst, 1997; Rintanen, 1998]. Theorem 3.75 does not hold if invariants are included because invariants contain information about the set of states that are not reachable by any sequential plan. For example, the formula $a \vee b$ is an invariant that would rule out states satisfying $\neg a \wedge \neg b$ that are reachable from any state satisfying $a \wedge b$ by simultaneous application of $\langle a, \{\neg b\}, \emptyset\rangle$ and $\langle b, \{\neg a\}, \emptyset\rangle$ but not sequentially reachable by these operators. However, the additional constraints in the following sections which restrict the parallel application of operators guarantee that only sequentially reachable states are considered. Therefore in the presence of the additional constraints for the different semantics invariants do not affect the set of satisfying valuations.

3.8.2 $\forall$-Step semantics

We have showed in Section 3.7.1 that the classes of $\forall$-step plans definable in terms of the notions of interference and interference in a state are tractable, in contrast to the general definition that is co-NP-hard.

In this section we present two encodings of the subclass of plans following $\forall$-step semantics in which no two parallel operators interfere. The first encoding is similar to the one used by Kautz and Selman in the BLACKBOX planner [Kautz and Selman, 1999] and has a size that is quadratic in the number of the operators. The size of the second encoding is linear in the size of the operators. Encodings for the more relaxed notion of interference in a state can be given, including an encoding with a linear size, but we do not discuss them in detail in this work.

A quadratic encoding

The simplest encoding of the interference condition in Definition 3.52 is by formulae

$$\neg o_t \vee \neg o'_t \tag{3.6}$$

for every pair of interfering operators o and o'. Note that according to our definition, operators that could never be applied simultaneously (because of conflicting preconditions or effects) may interfere. The formulae (3.6) for these kinds of pairs of operators are of course superfluous. Define $\Phi_{O,l}^{\forall step,1}$ as the conjunction of the formulae (3.6) for all time points $t \in \{0,\ldots,l-1\}$ and for all pairs of interfering operators $\{o,o'\} \subseteq O$ that could be applied simultaneously. There are $\mathcal{O}(ln^2)$ such formulae for n operators.

Theorem 3.77 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system. There is a $\forall$-step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,1}$ is satisfiable.*

Proof: Directly by Theorems 3.59 and 3.75. □

A similar quadratic-size encoding can also be given for state-dependent interference. The state-dependence is easy to encode by a formula that has a size proportional to the two operators: the simultaneous execution is allowed if none of the operators has an active effect that changes a state variable in the precondition or antecedent of a conditional effect of the other. Note that for STRIPS operators the state-dependent and state-independent notions of interference coincide, and even further, the above encoding of the $\forall$-step semantics is perfectly accurate.

Theorem 3.78 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system where O is a set of STRIPS operators. There is a $\forall$-step plan of length l for π if and only if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,1}$ is satisfiable.*

Proof: The *if* direction is by Theorem 3.77. It remains to show the *only if* direction. So assume there is a $\forall$-step plan $T = \langle S_0,\ldots,S_{l-1}\rangle$. By Theorem 3.75 there is a valuation v such that $v \models \Phi_{\pi,l}$. We show that also $v \models \Phi_{O,l}^{\forall step,1}$, that is, any conjunct $\neg o_i \vee \neg o_i'$ of $\Phi_{O,l}^{\forall step,1}$ for $i \in \{0,\ldots,l-1\}$ and $\{o,o'\} \subseteq O$ is satisfied by v.

Since $\neg o_i \vee \neg o_i'$ is in $\Phi_{O,l}^{\forall step,1}$, o and o' interfere. By Definition 3.52 this means for operators without conditional effects that there is a literal m such that m is an effect of o and $\overline{m}$ is a conjunct of the precondition of o', or the other way round. Hence by Theorem 3.49 $\{o,o'\} \not\subseteq S_i$. By the construction of v in the proof of Theorem 3.75 $v \models \neg o_i \vee \neg o_i'$. Hence every conjunct of $\Phi_{O,l}^{\forall step,1}$ is satisfied by v. □

A linear encoding

As the size of $\Phi_{\pi,l}$ is linear in l and the size of π, the quadratic encoding of the interference constraints may dominate the size of $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,1}$. We give a linear-size encoding for the interference constraints.

The idea of the encoding is to order all operators that may make a state variable $p \in A$ false (respectively true) or that have a positive (respectively negative) occurrence of p in the precondition or any occurrence in an antecedent of a conditional effect arbitrarily as $o^1,\ldots,o^n$. Whenever an operator o that falsifies p is applied, a sequence of implications prevents the application of every operator o' preceding or following o whenever o' has positive occurrences of p in the precondition or any occurrences in the antecedents of conditional effects. One chain of implications, through a set of auxiliary propositional variables, goes to the right in the ordering and another chain to the left.

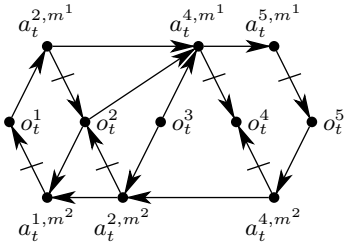


Figure 3.5: A linear-size encoding of interference constraints

We define a formula for every literal $m \in A \cup \{\neg p | p \in A\}$ for preventing the simultaneous application of operators that falsify m and operators that require m to remain true. Let $o^1, \ldots, o^n$ be any fixed ordering of the operators. Let E_m be the set of operators that may falsify m, and let R_m be the set of operators that may require m to remain true.

The formula is the conjunction of $\mathit{chain}(o^1, \ldots, o^n; E_m; R_m; m^1)$ and $\mathit{chain}(o^n, \ldots, o^1; E_m; R_m; m^2)$ for all literals m where

$$\begin{array}{rcl} \mathit{chain}(o^1, \ldots, o^n; E; R; m) & = & \bigwedge\{o_t^i \rightarrow a_t^{j,m} | i < j, o^i \in E, o^j \in R, \{o^{i+1}, \ldots, o^{j-1}\} \cap R = \emptyset\} \\ & & \cup\{a_t^{i,m} \rightarrow a_t^{j,m} | i < j, \{o^i, o^j\} \subseteq R, \{o^{i+1}, \ldots, o^{j-1}\} \cap R = \emptyset\} \\ & & \cup\{a_t^{i,m} \rightarrow \neg o_t^i | o^i \in R\}. \end{array}$$

The parameter m is needed to make the names of the auxiliary variables unique. The m^1 and m^2 are two names distinguishing the auxiliary variables for the two sets of formulae for literal m.

Example 3.79 Consider the following operators.

$$\begin{array}{rcl} o^1 & = & \langle x, \{\neg x, y\}, \emptyset\rangle \\ o^2 & = & \langle x, \{\neg x, z\}, \emptyset\rangle \\ o^3 & = & \langle z, \{\neg x\}, \emptyset\rangle \\ o^4 & = & \langle x, \{z\}, \emptyset\rangle \\ o^5 & = & \langle x, \{\neg x\}, \emptyset\rangle \end{array}$$

The formulae that encode the constraints on the simultaneous application for these operators and the state variable x are depicted in Figure 3.5. ■

The number of 2-literal clauses in $\mathit{chain}(o^1, \ldots, o^n; E_m; R_m; m^i)$ is at most three times the number of operators in which m occurs and hence the number of 2-literal clauses in

$$\mathit{chain}(o^1, \ldots, o^n; E_m; R_m; m^1) \wedge \mathit{chain}(o^n, \ldots, o^1; E_m; R_m; m^2)$$

is at most six times the number of operators. Since we have these formulae for every literal m, the number of 2-literal clauses is linearly bounded by the size of the set of operators. Let $\Phi_{O,l}^{\forall \mathit{step},2}$ be the conjunction of the above formulae for all literals m and time points $t \in \{0, \ldots, l-1\}$.

Theorem 3.80 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system. $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall \mathit{step},1}$ is satisfiable if and only if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall \mathit{step},2}$ is satisfiable. Hence there is a $\forall$-step plan for π of length l if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall \mathit{step},2}$ is satisfiable.*

Proof: Let v be a valuation such that $v \models \Phi_{O,l}^{\forall step,1}$. We construct a valuation v' that satisfies $\Phi_{O,l}^{\forall step,2}$. For all variables occurring in $\Phi_{O,l}^{\forall step,1}$ we have $v'(x) = v(x)$. Additionally, v' assigns values to the auxiliary variables a_t^{i,m^1} and a_t^{i,m^2} occurring only in $\Phi_{O,l}^{\forall step,2}$.

Let $v'(a_t^{j,m^1}) = 1$ iff there is $o^i \in E_m$ such that $i < j$ and $v(o_t^i) = 1$. Let $v'(a_t^{j,m^2}) = 1$ iff there is $o^i \in E_m$ such that $i > j$ and $v(o_t^i) = 1$.

We consider only the components of the first conjunct of $\mathit{chain}(o^1, \ldots, o^n; E_m; R_m; m^1) \wedge \mathit{chain}(o^n, \ldots, o^1; E_m; R_m; m^2)$. The second conjunct is analogous.

Consider $o_t^i \rightarrow a_t^{j,m^1}$ such that $i < j, o^i \in E_m, o^j \in R_m, \{o^{i+1}, \ldots, o^{j-1}\} \cap R_m = \emptyset$. If $v'(o_t^i) = 1$, then by the definition of v' also $v'(a_t^{j,m^1}) = 1$ because $i < j$ and $v'(o_t^i) = 1$.

Consider $a_t^{i,m^1} \rightarrow a_t^{j,m^1}$ such that $i < j, \{o^i, o^j\} \subseteq R_m, \{o^{i+1}, \ldots, o^{j-1}\} \cap R_m = \emptyset$. If $v(a_t^{i,m^1}) = 1$, then there is $o^{i'} \in E_m$ such that $i' < i$ and $v'(o_t^{i'}) = 1$. Therefore by the definition of v' we have $v'(a_t^{j,m^1}) = 1$.

Consider $a_t^{i,m^1} \rightarrow \neg o_t^i$ such that $o^i \in R_m$. If $v(a_t^{i,m^1}) = 1$, then there is $o^{i'} \in E_m$ such that $i' < i$ and $v'(o_t^{i'}) = 1$. Since $v' \models \neg o_t^{i'} \vee \neg o_t^i$, it must be that $v' \models \neg o_t^i$.

Hence all conjuncts of $\mathit{chain}(o^1, \ldots, o^n; E_m; R_m; m^1)$ are true in v'.

For the other direction, let v be a valuation such that $v \models \Phi_{O,l}^{\forall step,2}$. We show that $v \models \Phi_{O,l}^{\forall step,1}$. Take any conjunct $\neg o_t \vee \neg o'_t$ of $\Phi_{O,l}^{\forall step,1}$. If $v \not\models o_t$, then the truth immediately follows. Assume $v \models o_t$. Since $o = \langle p, e, c\rangle$ and $o' = \langle p', e', c'\rangle$ interfere, there is a state variable $a \in A$ that occurs as a negative effect of o and either in d for some $f \rhd d \in c'$ or positively in p' (or, the roles of o and o' are the other way around, or the polarity of the occurrences of a is complementary: the proofs of these cases are analogous.) Now $o \in E_a$ and $o' \in R_a$. We assume that the index o is lower than that of o'. The case with a higher index is analogous: instead of $\mathit{chain}(o^1, \ldots, o^n; E_a; R_a; a^1)$ we consider $\mathit{chain}(o^n, \ldots, o^1; E_a; R_a; a^2)$.

We show that because $v \models \mathit{chain}(o^1, \ldots, o^n; E_a; R_a; a^1)_t$, also $v \models \neg o'_t$.

The formula $\mathit{chain}(o^1, \ldots, o^n; E_a; R_a; a^1)_t$ contains a sequence of implications $o'_t \rightarrow a_t^{j_1,a} \rightarrow a_t^{j_2,a} \rightarrow \cdots \rightarrow a_t^{j_k,a} \rightarrow \neg o_t^{j_k}$ where $o^{j_k} = o'$. Since these implications are true in v, $v \not\models o'_t$. Therefore $v \models \neg o_t \vee \neg o'_t$. Since this holds for all conjuncts of $\Phi_{O,l}^{\forall step,1}$, we have $v \models \Phi_{O,l}^{\forall step,1}$. Since $v \models \Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,1}$ by Theorem 3.77 there is a $\forall$-step plan of length l for π. □

The number of auxiliary propositional variables is linearly proportional to the number of operators and state variables. Hence this linear-size encoding of the interference constraints may lead to formulae with a much higher number of propositional variables than with the quadratic size encoding of the constraints. The higher number of propositional variables may negatively affect the runtimes of satisfiability algorithms.

A compromise between the size of the constraints and the number of propositional variables is possible. There is an encoding of the constraints with only a logarithmic number of new propositional variables and with only $\mathcal{O}(n \log n)$ clauses which improves the quadratic encoding with respect to the number of clauses and the linear encoding with respect to the number of propositional variables. We describe the idea of the encoding without formalizing it and proving it correct.

The idea of the encoding is similar to that of $\mathit{chain}(o^1, \ldots, o^n; E_m; R_m;)$ in that an arbitrary ordering is imposed on the operators and the application of an operator prevents the application of operators later in the ordering. For each literal m we encode a binary number between 0 and

$|R_m| - 1$ in a logarithmic number of state variables. Then there is a formula for each operator o in E_m stating that the binary number for m has a value that is at least as high as the index of the first operator in R_m that follows o. For each operator o' in R_m there is similarly a formula that says that o' is not applied if the value of the binary number is lower than the index of o'. Hence no operator in R_m following an applied operator in E_m is applied.

The linear-size encoding and the above $n \log n$-size encoding can both be made state-dependent by observing the application of o with respect to the constraints related to literal m only if m is an active effect of o.

3.8.3 Process semantics

The encoding of process semantics extends the encoding of $\forall$-step semantics. We take all formulae for the latter (for example $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,2}$) and have further formulae specific to process semantics.

The encoding of the underlying $\forall$-step semantics encoding and the additional constraints for process semantics are tightly coupled: when the constraints force the movement of an operator to the preceding time point, the $\forall$-step semantics constraints for the preceding time points must be compatible with the move. In this section we discuss the encoding of the process constraints for the subclass of $\forall$-step plans based on interference (Definition 3.52 and Section 3.8.2.) Constraints compatible with broader classes of $\forall$-step plans (for example based on Definition 3.55) are more complicated.

The formulae for process semantics prevent the application of an operator o at time $t + 1$ if moving o to time t also resulted in a valid $\forall$-step plan according to Definition 3.47 and the state at time $t + 2$ stayed the same.

An operator o may be applied at time $t + 1$ only if at least one of the following conditions hold.

- The precondition of o became true at $t + 1$ (and is false at t.)
- The operator o interferes with an operator at time point t (Definition 3.52.)

  This includes the following pairwise tests.

  - Could one operator falsify the precondition of the other?
  - Could one operator change the set of active effects of the other. In other words, could it change the value of the antecedent of a conditional effect of the other?

  Note that if none of the operators at t interfere with the operator at $t + 1$ then the operator would have the same effects at t as it has at $t + 1$.
- The active effects of o are in conflict with the active effects of an operator at t.

We give a linear-size encoding of these conditions. Let the set of state variables be $A = \{a^1, \ldots, a^n\}$. We introduce the following auxiliary propositional variables.

- The variables $a_t^{i,1}$ denote that an operator at time $t+1$ makes (may make) a^i *true*, and hence a justification for not moving the operator earlier is that
  - there is an operator at t with a *negative* occurrence of a^i in its precondition, or
  - there is an operator at t with an occurrence of a^i in the lhs of a conditional effect,

- The variables $a_t^{i,\neg 1}$ denote that an operator at time $t+1$ makes (may make) a^i *false*, and hence a justification for not moving that operator earlier is that
  - there is an operator at t with a *positive* occurrence of a^i in its precondition, or
  - there is an operator at t with an occurrence of a^i in the lhs of a conditional effect.
- The variables $a_t^{i,2}$ denote that an operator at time $t+1$ has an occurrence of a^i in the antecedent of a conditional effect, and hence a justification for not moving that operator earlier is that there is an operator at t that changes the value of a^i.
- The variables $a_t^{i,3}$ denote that an operator at time $t+1$ has a *positive* occurrence of a^i in the precondition, and hence a justification for not moving that operator earlier is that there is an operator at t that makes (may make) a^i *false*.
- The variables $a_t^{i,\neg 3}$ denote that an operator at time $t+1$ has a *negative* occurrence of a^i in the precondition, and hence a justification for not moving that operator earlier is that there is an operator at t that makes (may make) a^i *true*.
- The variables $a_t^{i,4}$ denote that an operator at time $t+1$ (actually) makes a^i *true*, and hence a justification for not moving that operator earlier is that there is an operator at t that (actually) makes a^i *false*.
- The variables $a_t^{i,\neg 4}$ denote that an operator at time $t+1$ (actually) makes a^i *false*, and hence a justification for not moving that operator earlier is that there is an operator at t that (actually) makes a^i *true*.

Note that the definition of interference in Definition 3.52 is about occurrences of a state variable in the effects of one operator and in the precondition or in the antecedents of conditional effects of another operator. This is the reason why in the above description we have stated that an operator *may make* a state variable true or false. Below we make this more explicit.

We need the following formulae for each state variable a^i and all $t \in \{0, \ldots, l-1\}$.

$$a_{t+1}^{i,1} \rightarrow (o_t^1 \vee \cdots \vee o_t^n) \tag{3.7}$$

where $o^1, \ldots, o^n$ are all the operators o that have an occurrence of a^i in the lhs of a conditional effect, or a *negative* occurrence of a^i in the precondition.

$$a_{t+1}^{i,\neg 1} \rightarrow (o_t^1 \vee \cdots \vee o_t^n) \tag{3.8}$$

where $o^1, \ldots, o^n$ are all the operators o that have a *positive* occurrence of a^i in the precondition, or an occurrence of a^i in the lhs of a conditional effect.

$$a_{t+1}^{i,2} \rightarrow (o_t^1 \vee \cdots \vee o_t^n) \tag{3.9}$$

where $o^1, \ldots, o^n$ are all the operators in which a^i occurs in an effect.

$$a_{t+1}^{i,3} \rightarrow (o_t^1 \vee \cdots \vee o_t^n) \tag{3.10}$$

where $o^1, \ldots, o^n$ are all the operators o that have the effect $\neg a^i$ (possibly conditional).

$$a_{t+1}^{i,\neg 3} \rightarrow (o_t^1 \vee \cdots \vee o_t^n) \tag{3.11}$$

where $o^1, \ldots, o^n$ are all the operators o that have the effect a^i (possibly conditional).

Additionally, for each operator $o \in O$ we need a formula that lists all the possible justifications for not moving the operator one step earlier. These formulae are

$$o_t \rightarrow (\neg p_{t-1} \vee \phi) \tag{3.12}$$

where p is the precondition of o and ϕ is the disjunction of the propositional variables

- $a_t^{i,1}$ such that a^i is an effect (possibly conditional) of o,
- $a_t^{i,\neg 1}$ such that $\neg a^i$ is an effect (possibly conditional) of o,
- $a_t^{i,2}$ such that a^i occurs in the antecedent of a conditional effect of o,
- $a_t^{i,3}$ such that a^i occurs positively in the precondition of o, and
- $a_t^{i,\neg 3}$ such that a^i occurs negatively in the precondition of o.

For the variables $a_t^{i,4}$ and $a_t^{i,\neg 4}$ we replace each positive occurrence of a_t^i in the consequent of the implication of Formula 3.3 by $(a_t^i \wedge a_t^{i,4} \wedge a_{t-1}^{i,\neg 4})$ and each occurrence of $\neg a_t^i$ by $(\neg a_t^i \wedge a_t^{i,\neg 4} \wedge a_{t-1}^{i,\neg 4})$ for all $t \in \{1, \ldots, l-1\}$. This is to indicate that a^i or $\neg a^i$ is an active effect of the operator at time t.

The variables $a_t^{i,2}$, $a_t^{i,3}$ and $a_t^{i,\neg 3}$ and the associated formulae are not needed if all operators are STRIPS operators. For STRIPS operators the use of variables $a_t^{i,4}$ and $a_t^{i,\neg 4}$ could be replaced by the use $a_t^{i,1}$ and $a_t^{i,\neg 1}$.

Let the formula $\Phi_{O,l}^{process}$ be a conjunction of all the above formulae. The size of $\Phi_{O,l}^{process}$ is linear in the size of the set O of operators because there are at most $2l$ variable occurrences for every state variable occurrence in every operator.

Theorem 3.81 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system. There is i-process plan T of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,2} \wedge \Phi_{O,l}^{process}$ is satisfiable.*

Proof: Assume v is a valuation such that $v \models \Phi_{\pi,l} \wedge \Phi_{O,l}^{\forall step,2} \wedge \Phi_{O,l}^{process}$. Define for all $i \in \{0, \ldots, l\}$ the state s_i as the valuation of A such that $s_i(a) = v(a_i)$ for every $a \in A$. Define $S_j = \{o \in O | v(o_j) = 1\}$ for all $j \in \{0, \ldots, l-1\}$. By Theorem 3.80 $T = \langle S_0, \ldots, S_{l-1}\rangle$ is a $\forall$-step plan.

Assume that T is not an i-process plan because for some $i \in \{1, \ldots, l-i\}$ and $o^x \in S_i$, $T' = \langle S_0, \ldots, S_{i-1} \cup \{o^x\}, S_i \backslash \{o^x\}, \ldots, S_{l-1}\rangle$ is a $\forall$-step plan in which no two simultaneous operators interfere. We show that this leads to a contradiction with the assumption that $v \models \Phi_{O,l}^{process}$.

Consider $o_i^x \rightarrow (\neg p_{i-1}^x \vee j^1 \vee \cdots \vee j^n)$. Assume that v satisfies this formula. Since $v \models o_i^x$ (as $o^x \in S_i$), at least one of the disjuncts in the right side is true in v. It cannot be that $v \models \neg p_{i-1}^x$ where p^x is the precondition of o^x because otherwise o^x would not be applicable at time $i-1$ in T'.

So some other disjunct of $j^1 \vee \cdots \vee j^n$ must be satisfied by v. This leads to a long and tedious case analysis. We only give as an example the proof for the disjunct $a_i^{q,1}$ for a state variable a^q that is a positive effect of o^x. If $v \models a_i^{q,1}$, then because $v \models a_i^{q,1} \rightarrow (o_{i-1}^1 \vee \cdots \vee o_{i-1}^n)$ where $o^1, \ldots, o^n$ are all the operators that have an occurrence of a^q in the lhs of a conditional effect or a negative occurrence in the precondition. Hence there is an operator $o^y \in S_{i-1}$ that has an occurrence of

a^q in the lhs of a conditional effect or a negative occurrence in the precondition. Hence o^x and o^y interfere, and both are at $i-1$ in T', which contradicts our assumptions.

Therefore it must be the case that T is an i-process plan. □

3.8.4 ∃-Step semantics

We give three encodings of the constraints that guarantee that the plans follow the ∃-step semantics. The first two (Sections 3.8.4 and 3.8.4) exactly encode the acyclicity test, allowing maximum parallelism with respect to a given disabling graph (as defined in Section 3.8.4). However, the first of these encodings has a cubic size and the second involves guessing a topological ordering for the set of operators, and therefore these encodings would not appear to be practical. The third encoding (Section 3.8.4) is based on assigning a fixed ordering on the operators and allowing the simultaneous application of a subset of the operators only if none of the operators affects the operators later in the ordering. The size of this encoding is linear in the size of the set of operators, but it allows less parallelism than the first two encodings. However, in our experiments this encoding has turned out to be very efficient.

To improve the efficiency of the encodings we consider a method for utilizing the structural properties of planning problems in the form of *disabling graphs* in Section 3.8.4. The idea is to identify operators for which the existence of a linearization required by the ∃-step semantics can be guaranteed, no matter in which state the set of operators is simultaneously applied. The set of operators is partitioned to subsets of operators potentially involved in a cycle that cannot be linearized. Constraints guaranteeing the linearization property need to be given only for such subsets. The decomposition method in some cases splits the set of all operators to singleton subsets. If all sets are singleton, the linearization property is guaranteed for any subset of operators applied simultaneously, and there is no need to introduce further constraints on operator application. The technique improves all the three encodings of the ∃-step semantics on many types of structured problems.

Disabling graphs

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 and on all total orderings of the operators at least one operator affects an operator later in the ordering. Now any set-inclusion minimal circularly disabled set is a subset of a strong component (or strongly connected component, abbreviated as SCC) of the disabling graph.

Definition 3.82 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system. A graph $\langle O, E\rangle$ is a* disabling graph *for π when $E \subseteq O \times O$ is the set of directed edges so that $\langle o, o'\rangle \in E$ if*

1. *there is a state s such that s is reachable from I by operators in O and $app_{\{o,o'\}}(s)$ is defined, and*
2. *o affects o'.*

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. Also the complete graph $\langle O, O \times O\rangle$ is a disabling graph. For every set of operators there is a unique minimal disabling graph, but

computing minimal disabling graphs is NP-hard because of the consistency tests and PSPACE-hard because of the reachability tests of s in Condition 1. Computing non-minimal disabling graphs is easier because the consistency and reachability tests may be approximated.

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.[3]

Lemma 3.83 *Let O be a set of operators and $G = \langle O, E\rangle$ a disabling graph for O. Let $C_1, \ldots, C_m$ be the strong components of G. Let s be a state. Let O' be a set of operators so that $app_{O'}(s)$ is defined. If for every $i \in \{1, \ldots, m\}$ the subgraph $\langle C_i \cap O', E \cap ((C_i \cap O') \times (C_i \cap O'))\rangle$ of G induced by $C_i \cap O'$ is acyclic, then there is a total ordering $o_1, \ldots, o_n$ of O' such that $app_{o_1;\ldots;o_n}(s) = app_{O'}(s)$.*

Proof: Let the indices of $C_1, \ldots, C_m$ be such that for all $i \in \{1, \ldots, m-1\}$ and $j \in \{i+1, \ldots, m\}$ there are no edges from an operator in C_i to an operator in C_j. Such a numbering exists because the sets C_i are strong components of G (the strong components always form a tree.) Since the subgraph induced by $C_i \cap O'$ is acyclic for every $i \in \{1, \ldots, m\}$, we can impose an ordering $o_1 <_i \ldots <_i o_{n_i}$ on $C_i \cap O'$ so that if $o <_i o'$ then there is no edge from o to o', that is, o does not affect o'.

Now we can construct a total order $o_1 < \cdots < o_n$ on O' as follows. For all $\{o, o'\} \in O'$, $o < o'$ if $\{o, o'\} \subseteq C_i$ for some $i \in \{1, \ldots, m\}$ and $o <_i o'$, or $o \in C_i$ and $o' \in C_j$ and $i < j$. Now for all $\{o, o'\} \subseteq O'$, if $o < o'$ then o does not affect o'. Hence $app_{o_1;\ldots;o_n}(s) = app_{O'}(s)$ by Lemma 3.72. □

Note that acyclicity is a sufficient but not a necessary condition for a set of operators to be executable in some order, even for minimal disabling graphs. This is because the edges are independent of the state, exactly like the notion of interference in Definition 3.52. As in Example 3.60 two operators may form a cycle in the disabling graph but can nevertheless be executed in any order with the same results. However, for STRIPS operators and minimal disabling graphs acyclicity exactly coincides with executability in some order, as we show in Lemma 3.84. This fact was already implicitly used in Theorem 3.74.

Lemma 3.84 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system and $\langle O, E\rangle$ a disabling graph for π such that $\langle o, o'\rangle \in E$ only if o affects o'. Let s be a state reachable from I by some sequence of operators in O and let $S = \{o_1, \ldots, o_n\}$ be a set of STRIPS operators so that $app_{o_1;\ldots;o_n}(s)$ and $app_S(s)$ are defined for some ordering $o_1, \ldots, o_n$ of S. Then the subgraph of $\langle O, E\rangle$ induced by S is acyclic.*

Proof: Fact A: Since $app_S(s)$ is defined, there are no $\{\langle p, e, \emptyset\rangle, \langle p', e', \emptyset\} \subseteq S$ and $a \in A$ so that $a \in e$ and $\neg a \in e'$.

Since $app_{o_1;\ldots;o_n}(s)$ is defined, there are no $i \in \{1, \ldots, n-1\}$ and $j \in \{i+1, \ldots, n\}$ such that o_i affects o_j. If there were, o_i would make one of the literals in the precondition of o_j false and by Fact A no operator $o_k, k \in \{i+1, \ldots, j-1\}$ could make the precondition true again, and hence $app_{o_1;\ldots;o_j}(s)$ would not be defined. As no operator in S affects a later operator, and there is an edge from an operator to another only if the former affects the latter, the subgraph of $\langle O, E\rangle$

[3] In ∀-step semantics simultaneous application is allowed only if the subgraph induced by all applied operators does not have *any* edges.

induced by S is acyclic. □

Next we discuss three ways of deriving constraints that guarantee that operators occupying one SCC of a disabling graph can be totally ordered to a valid plan.

Encoding of size $\mathcal{O}(n^3)$

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.

We use auxiliary propositional variables $c_t^{i,j}$ for all operators with indices i and j indicating that the operators $o^i, o^1, o^2, \ldots, o^n, o^j$ are applied and each operator affects its immediate successor in the sequence. 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 $o^{i'}$. Then we have the formulae $(o_t^i \wedge o_t^{i'}) \rightarrow c_t^{i,i'}$ and $(o_t^i \wedge c_t^{i',j}) \rightarrow c_t^{i,j}$ for all j such that $i' \neq j \neq i$. Further we have formulae $\neg(o_t^i \wedge c_t^{i',i})$ for preventing the completion of a cycle.

There is a cubic number of formulae, each having a constant size (two or three variable occurrences). The number of propositional variables $c_t^{i,j}$ is quadratic in the number of operators in an SCC. Some problems have SCCs of hundreds or thousands of operators, and this would mean millions or billions of formulae, which would often make the encoding impractical.

Theorem 3.85 *Let $\pi = \langle A, I, O, G\rangle$ be a transition system. There is a $\exists$-step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{1lin,1}$ is satisfiable.*

Proof: Let v be a valuation such that $v \models \Phi_{\pi,l} \wedge \Phi_{O,l}^{1lin,1}$. Define for all $i \in \{0, \ldots, l\}$ the state s_i as the valuation of A such that $s_i(a) = v(a_i)$ for every $a \in A$. Define $S_j = \{o \in O | v(o_j) = 1\}$ for all $j \in \{0, \ldots, l-1\}$. By Theorem 3.75 we only have to test the condition that for $\langle S_0, \ldots, S_{l-1}\rangle$, its execution $s_0, \ldots, s_l$ and every $i \in \{0, \ldots, l-1\}$ there is a total ordering $o_1, \ldots, o_n$ of S_i such that $app_{o_1;\ldots;o_n}(s_i) = app_{S_i}(s_i)$.

By Lemma 3.83 it suffices to show that the subgraph of the disabling graph induced by $S_i \cap C$ for every SCC C of the disabling graph is acyclic. For the sake of argument assume that the subgraph has a cycle. Hence there are operators $o'^1, \ldots, o'^m$ in S_i such that o'^j affects o'^{j+1} for all $j \in \{1, \ldots, m-1\}$ and o'^m affects o'^1. But the formulae

$$o_i'^{m-1} \wedge o_i'^m \rightarrow c_i^{m-1,m}, o_i'^{m-2} \wedge c_i^{m-1,m} \rightarrow c_i^{m-2,m}, \ldots, o_i'^1 \wedge c_i^{2,m} \rightarrow c_i^{1,m}, \neg(o_i'^m \wedge c_i^{1,m})$$

together with $o_i'^1, \ldots, o_i'^m$ are inconsistent. Since these formulae are conjuncts of $\Phi^{1lin,1}$, there can be no cycle in the subgraph induced by $S_i \cap C$. □

Encoding of size $\mathcal{O}(e \log_2 n)$

A more compact encoding is obtained by assigning a $\log_2 n$-bit binary number to each of the n operators and by requiring that the number of operator o is lower than that of o' if there is an edge from o' to o in the disabling graph.[4] The size of the encoding is $\mathcal{O}(e \log_2 n)$ where e is the number of edges in the disabling graph and n is the number of operators.

[4]This encoding has also been independently discovered by Victor Khomenko [2005].

For every operator o and time point t we introduce the propositional variables $i_t^{o,0}, \ldots i_t^{o,k}$ where $k = \lceil \log_2 n \rceil - 1$ for encoding o's index at time point t.

So, for any operators o and o' so that o' affects o use the following formula for guaranteeing that the edges are always from an operator with a higher index to a lower index.

$$(o_t \wedge o'_t) \rightarrow GT(i_t^{o',0}, \ldots, i_t^{o',k}; i_t^{o,0}, \ldots, i_t^{o,k}) \tag{3.13}$$

Above $GT(i_t^{o',0}, \ldots, i_t^{o',k}; i_t^{o,0}, \ldots, i_t^{o,k})$ is a formula comparing two k-bit binary numbers. There are such formulae that have a size that is linear in the number of bits.

Theorem 3.86 *Let $\pi = \langle A, I, O, G \rangle$ be a transition system. There is a $\exists$-step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{1lin,2}$ is satisfiable.*

Proof: Similarly to the proof of Theorem 3.85, we have to show that the subgraph induced by every set of simultaneous operators is acyclic. Formula 3.13 guarantees that the index of an operator to which there is an edge from another operator is lower than the index of the latter. The existence of a cycle would mean that there are also edges from an operator with a lower index to an operator to a higher index but as such edges do not exist, there are no cycles in the graph. □

Note that given a set of literals describing which operators are applied at a given time point, for the encoding in Section 3.8.4 unit resolution is sufficient for determining whether there is a cycle, but not for the encoding in Section 3.8.4.

A linear-size encoding based on a fixed ordering of operators

Our third encoding does not allow all the parallelism allowed by the preceding encodings but it leads to small formulae and seems to be very efficient in practice. With this encoding the set of formulae constraining parallel application *is a subset* of those for the less permissive $\forall$-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 $o^1, \ldots, o^n$ in an SCC and to allow parallel application of two operators o^i and o^j such that o^i affects o^j only if $i \geq j$. Of course, this restriction to one given linearization may rule out many sets of parallel operators that could be applied simultaneously according to some other linearization than the fixed one.

A trivial implementation of this idea (similar to the $\forall$-step semantics encoding in Section 3.8.2) has a quadratic size because of the worst-case quadratic number of pairs of operators that may not be simultaneously applied. However, we may use one half of the implications in the linear-size encoding for $\forall$-step semantics from Section 3.8.2. The linear-size encoding for the constraints for $\exists$-step semantics is thus simply the conjunction of formulae

$$\mathit{chain}(o^1, \ldots, o^n; E_m; R_m; m)$$

for every literal m where E_m is the set of operators that may falsify m ($\overline{m}$ occurs as an atomic effect) and R_m is the set of operators that may require m to remain true (m occurs in the antecedent of a conditional effect or positively in the precondition).

Theorem 3.87 *Let $\pi = \langle A, I, O, G \rangle$ be a transition system. There is a $\exists$-step plan of length l for π if $\Phi_{\pi,l} \wedge \Phi_{O,l}^{1lin,3}$ is satisfiable.*

Proof: Let v be a valuation such that $v \models \Phi_{\pi,l} \wedge \Phi_{O,l}^{1lin,3}$. Define for all $i \in \{0,\ldots,l\}$ the state s_i as the valuation of A such that $s_i(a) = v(a_i)$ for every $a \in A$. Define $S_j = \{o \in O | v(o_j) = 1\}$ for all $j \in \{0,\ldots,l-1\}$. Consider an SCC C of the disabling graph and the fixed ordering $o'^1,\ldots,o'^{n'}$ of the operators in $C \cap S_i$ for some $i \in \{0,\ldots,l-1\}$.

The formulae $\mathit{chain}(o^1,\ldots,o^n;E_m;R_m;m)$ in $\Phi_{O,l}^{1lin,3}$ for all literals m guarantee that if o'^j affects o'^k, then $k < j$. By Lemma 3.72 $\mathit{app}_{S_i}(s_i) = \mathit{app}_{o'^1;\ldots;o'^{n'}}(s_i)$. Hence the definition of $\exists$-step plans is satisfied. □

3.8.5 Experiments

The shortest encodings of the three semantics in Sections 3.8.2, 3.8.3 and 3.8.4 have sizes that are linear in the size of the problem instance and the number of time points, and are therefore asymptotically optimal. The question arises whether the potentially much smaller number of time points makes $\exists$-step semantics more efficient than $\forall$-step semantics and whether the potentially much smaller number of plans makes the process semantics more efficient than $\forall$-step semantics. In this section we answer these questions by comparing the different encodings of the three semantics with respect to a number of planning problems.

We consider two problem classes. First, as a way of measuring the efficiency of the encodings on "average" problem instances, we sample problem instances from the space of all problem instances characterized by certain parameter values, following Bylander [1996] and Rintanen [2004d]. The problem instances we consider are rather small, 40 state variables and up to 280 operators, but rather challenging in the phase transition region.

Second, we consider some of the benchmarks used by the planning community. These problem instances have a simple interpretation in terms of real-world planning tasks, like simple forms of transportation planning. In contrast to the problem instances in the phase transition study, the numbers of state variables and operators in these problems are much higher (up to several thousands of state variables and tens of thousands of operators), and most of these problems can be solved rather easily by domain-specific polynomial time algorithms when no optimality criteria (for example minimal number of operators in a plan) have to be satisfied.

Implementation details

We briefly discuss details of the implementation of our translator from the planning domain description language PDDL [Ghallab *et al.*, 1998] into propositional formulae in conjunctive normal form.

The planning domain description language PDDL allows describing schematic operators that are instantiated with a number of objects. For some of the standard benchmark problems the number of operators produced by a naïve instantiation procedure is astronomic, and indeed all practical planner implementations rely on heuristic techniques for avoiding the generation of ground operators that could never be part of a plan because no state satisfying the precondition of the operator can be reached.

After instantiating the schematic PDDL operators, we perform a simple polynomial-time reachability analysis for the possible values of state variables to identify operators that can never be applied. For example, in the 1998 and 2000 AIPS planning competition logistics problems there are operators for driving trucks between locations outside the truck's home city, but the truck can

never leave its home city. Hence the state variables indicating that the truck's location is a non-home city can never be true. This analysis allows eliminating many irrelevant operators and it is similar to the reachability analysis performed by the GraphPlan [Blum and Furst, 1997] and BLACKBOX [Kautz and Selman, 1999] planners.

Similarly to BLACKBOX [Kautz and Selman, 1999] and other implementations of satisfiability planning, our translation includes formulae $l_t \vee l'_t$ for invariants $l \vee l'$ as produced by the algorithm by Rintanen [1998]. This algorithm is defined for STRIPS operators only but can be generalized to arbitrary operators (Section 3.5).

In the experiments we use disabling graphs that are not necessarily minimal but can be computed in polynomial time. The test of whether two operators can be simultaneously applied in some state is not exact: we only test whether the unconditional effects contradict directly or through an invariant and whether the preconditions have conjuncts that are complementary literals or contradict through an invariant. For STRIPS operators the graphs are minimal whenever the invariants are sufficient for determining whether a state in which two operators are both applicable is reachable.

The orderings in the $\exists$-step encoding of Section 3.8.4 were the ones in which the operators came out of our PDDL front-end. Better orderings that minimize the number of pairs of operators o and o' such that o precedes and affects o' could be produced by heuristic methods. They can potentially increase parallelism and improve runtimes.

The AIPS 2000 planning competition Schedule benchmarks contain conditional effects $m \rhd \overline{m}$, sometimes simultaneously with effects m. The purpose of this is apparently to make it difficult for planners like GraphPlan [Blum and Furst, 1997] or BLACKBOX [Kautz and Selman, 1999] to apply several operators in parallel. Replacing effects $m \rhd \overline{m}$ by preconditions $\overline{m}$ whenever also m is an unconditional effect and by effects $\overline{m}$ whenever m is not an unconditional nor a conditional effect of the operator, is a transformation that preserves the semantics of the operators exactly and for this benchmark allows much more parallelism. The front-end of our translator performs this transformation.

The SAT solvers we use only accept formulae in conjunctive normal form (CNF) as input. Therefore all the propositional formulae have to be transformed to CNF. We use a simple scheme for doing this. For any subformula of the form $(\phi_1 \wedge \phi_2) \vee \psi$ we introduce an auxiliary variable x, replace the subformula by $x \vee \psi$ and add $x \rightarrow \phi_1$ and $x \rightarrow \phi_2$ to our set of formulae. Note that almost all of the formulae in our encodings are already in CNF (modulo equivalences like $\neg(\phi \wedge \psi) \leftrightarrow \neg\phi \vee \neg\psi$.) Exceptions to this are the precondition axioms for operators with disjunctive preconditions and effect axioms for operators with conditional effects.

For effect axioms $o_t \rightarrow e_t$ we only include those effects in e that are not consequences of other effects and invariants. For example, many operators in the standard benchmarks have effects *at(A,L1)*$\wedge\neg$*at(A,L2)* for representing the movement of an object from location 2 to location 1. Then $\neg$*at(A,L1)*$\vee\neg$*at(A,L2)* is an invariant that is included in the problem encoding. Since $\neg$*at(A,L2)* is a consequence of the invariant together with *at(A,L1)*, the effect axiom 3.2 does not have to state this explicitly. This reduces the size of the formulae slightly and has a small effect on runtimes.

Experimental setting

For the experiments we use a 3.6 GHz Intel Xeon processor with 512 KB internal cache and the Siege SAT solver version 4 by Ryan of the Simon Fraser University.

In addition to Siege V4, we ran tests with the May 13, 2004 version of zChaff. The runtimes are

close to the ones for Siege, often worse but in some cases slightly better. We could solve some of the biggest structured instances (Section 3.8.5) in a reasonable time only with Siege. BerkMin and some of the best solvers in the 2005 SAT solver competition are also rather good on the planning problems.

As Siege V4 uses randomization, its runtimes vary, in some cases considerably. For the structured problems the tables give the average runtimes over 100 runs and 95 percent confidence intervals for the average runtimes. As it is not known what the distribution of Siege runtimes on a given instance is, we calculate the confidence intervals by using a standard bootstrapping procedure [Efron and Tibshirani, 1986; 1993]. From the sample of 100 runtimes we resample (with replacement) 4000 times a sample of 100 runtimes and then look at the distribution of these averages. The 95 percent confidence interval is obtained as the 2.5 and 97.5 percentiles of this distribution.

Problem instances sampled from the phase transition region

We considered problem instances with $|A| = 40$ state variables, corresponding to state spaces with $2^{40} \sim 10^{12}$ states, and STRIPS operators with 3 literals in the preconditions and 2 literals in the effect, following Model A of Rintanen [2004d] in which precondition literals are chosen randomly and independently, and effect literals are chosen randomly so that each propositional variable has about the same number of occurrences in an atomic effect, both negatively and positively. In the initial state all state variables are false and in the unique goal state all state variables are true. We generated about 1000 soluble problem instances for ratios $\frac{|O|}{|A|}$ of operators to state variables varying from 1.85 to 5 at an interval of about 0.3. The number of operators then varied from 74 to 280. To find 1000 soluble instances for the smaller ratios we had to generate up to 45000 instances most of which are insoluble. Since we did not have a complete insolubility test, we do not know how many of the instances that we could not solve within our limits on plan length (60 time points) and CPU time (3 minutes per formula) are really insoluble. For the $\forall$-step and the process semantics the number of instances solved within the time limit was slightly smaller than for the $\exists$-step semantics, so the actual performance difference is slightly bigger than what the diagram suggests.

Figure 3.6 depicts the average runtimes of Siege with the $\exists$-step (the linear-size encoding from Section 3.8.4), $\forall$-step (the linear-size encoding from Section 3.8.2) and process semantics (the linear-size encoding from Section 3.8.3 based on the linear-size $\forall$-step encoding from Section 3.8.2). There are two sources of imprecision in the runtime comparison, the variation of runtimes of Siege due to randomization and the random variation in the properties of problem instances sampled from the space of all problem instances. For this reason we give estimates on the accuracy of the averages of runtimes. The diagrams depicting the runtimes give error bars indicating the 95 percent confidence intervals for the runtimes. Note that the scale of the runtime diagram is logarithmic.

Figure 3.7 depicts the average numbers of operators in the plans. Figure 3.8 depicts the average number of time points in the plans. The process and $\forall$-step semantics share the curve because the shortest number of time points of a plan for any problem instance is the same for both.

As is apparent from the diagrams, the $\exists$-step semantics is by far the most efficient of the three. The efficiency is directly related to the fact that with $\exists$-step semantics the shortest plans often have less time points than with the $\forall$-step and process semantics. The encoding for the process semantics is the slowest, most likely because of the higher number of propositional variables and

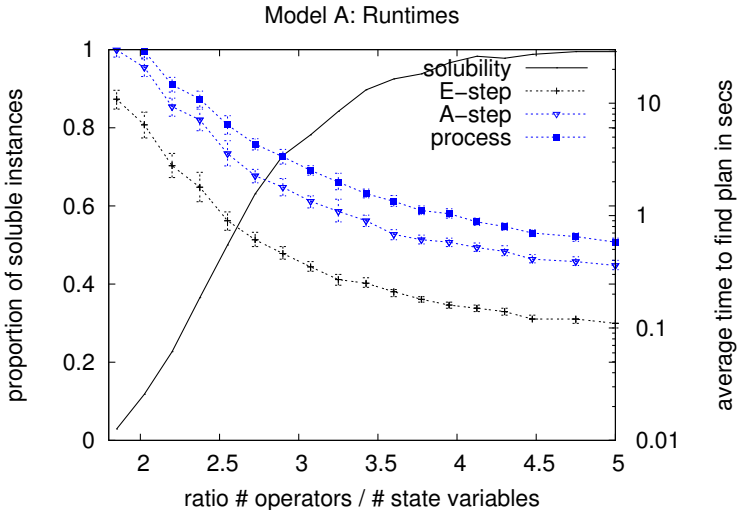


Figure 3.6: Runtimes of ∃-step, ∀-step and process semantics on problem instances with 40 state variables sampled from the phase transition region.

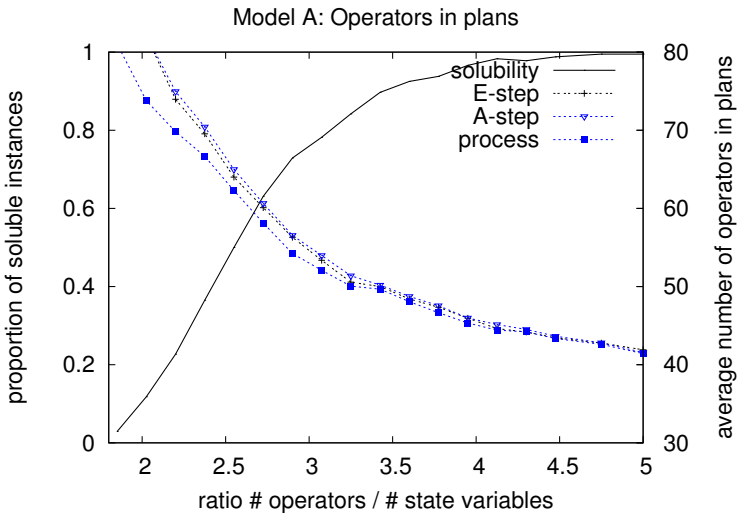


Figure 3.7: Numbers of operators in plans for ∃-step, ∀-step and process semantics on problem instances with 40 state variables sampled from the phase transition region.

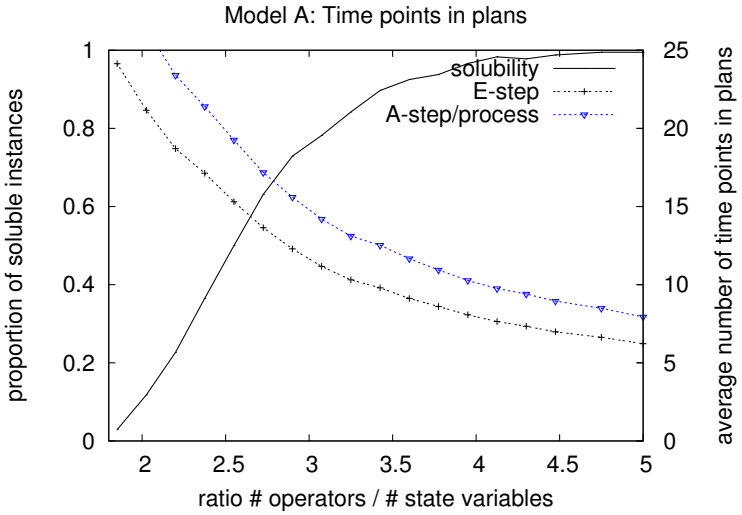


Figure 3.8: Numbers of time points in plans for ∃-step, ∀-step and process semantics on problem instances with 40 state variables sampled from the phase transition region. For ∀-step and process semantics the number of time points is always the same.

clauses and the ineffectiveness of the process constraints on these problems.

Interestingly, the number of operators in the ∃-step and ∀-step plans is almost exactly the same despite the fact that the ∀-step semantics needs more time points. On the other hand, process semantics imposes stricter constraints on the plans than the ∀-step semantics, and the number of operators is therefore slightly smaller.

Structured problem instances

We evaluate the different semantics on a number of benchmarks from the AIPS planning competitions of years 1998, 2000 and 2002. For a discussion of these benchmarks and their properties see Section 3.9.5. We also tried the benchmarks from the year 2004 competition, but, although most of them are easy to solve, they result in very big formulae, and the relative behavior of the encodings of the different semantics on them is similar to the benchmarks we report in this paper. Hence we did not run exhaustive tests with them.

For all other benchmarks we use the STRIPS version, but for the Schedule benchmark we use the ADL version because with the STRIPS version our translator has problems with the very high number of operators. However, the simplification mentioned in Section 3.8.5 transforms also these operators to STRIPS operators.

In Tables 3.1, 3.8, 3.9, 3.10, 3.11 3.12, 3.13, 3.14 and 3.15 (all but the first are in the appendix) we present for each problem instance the runtimes for the formulae corresponding to the highest number of time points without a plan (truth value F) and the first satisfiable formula corresponding to a plan (truth value T). The rows marked with the question mark indicate that none of the runs successfully terminated and we therefore do not know whether the formulae are satisfiable or unsatisfiable. The column ∃-*step* is for the ∃-step semantics encoding in Section 3.8.4, the column *process* for the process semantics encoding in Section 3.8.3, the column ∀-*step* for the worst-case

instance	len	val	∃-step	process	∀-step	∀-step l.
log-16-0	7	F	$0.01^{0.01}_{0.01}$			
log-16-0	8	T	$0.03^{0.03}_{0.04}$			
log-16-0	12	F		$0.62^{0.57}_{0.67}$	$0.30^{0.27}_{0.33}$	$0.79^{0.73}_{0.86}$
log-16-0	13	T		$7.46^{6.96}_{7.98}$	$1.35^{1.19}_{1.52}$	$2.27^{2.04}_{2.50}$
log-17-0	8	F	$0.15^{0.14}_{0.15}$			
log-17-0	9	T	$0.02^{0.02}_{0.02}$			
log-17-0	13	F		$3.06^{2.93}_{3.19}$	$1.97^{1.89}_{2.05}$	$2.25^{2.15}_{2.35}$
log-17-0	14	T		$14.40^{13.71}_{15.11}$	$3.22^{2.93}_{3.55}$	$4.48^{4.07}_{4.91}$
log-18-0	8	F	$0.13^{0.13}_{0.14}$			
log-18-0	9	T	$0.33^{0.26}_{0.40}$			
log-18-0	14	F		$8.18^{7.74}_{8.67}$	$5.83^{5.50}_{6.17}$	$6.77^{6.47}_{7.08}$
log-18-0	15	T		–	$7.84^{6.78}_{9.09}$	$14.95^{13.13}_{16.78}$
log-19-0	8	F	$0.23^{0.22}_{0.25}$			
log-19-0	9	T	$0.33^{0.22}_{0.46}$			
log-19-0	14	F		$10.23^{9.54}_{10.95}$	$11.22^{10.47}_{11.98}$	$13.39^{12.68}_{14.12}$
log-19-0	15	T		-	$29.10^{25.33}_{33.05}$	–
log-20-0	8	F	$0.25^{0.24}_{0.26}$			
log-20-0	9	T	$0.88^{0.64}_{1.17}$			
log-20-0	14	F		$12.30^{11.76}_{12.83}$	$10.63^{9.96}_{11.32}$	$12.01^{11.36}_{12.67}$
log-20-0	15	T		-	–	$41.17^{36.81}_{45.80}$

Table 3.1: Runtimes of Logistics problems

instance	len	∃-step	process	∀-step	∀-step l.
depot-16-4398	8	$53.00\,^{53}_{53}$	$43.22\,^{36}_{48}$	$43.40\,^{38}_{50}$	$38.97\,^{35}_{44}$
driver-4-4-8	9	$54.19\,^{50}_{61}$			
driver-4-4-8	11		$55.45\,^{50}_{61}$	$52.32\,^{50}_{58}$	$51.47\,^{50}_{58}$
gripper-3	8	$23.21\,^{23}_{24}$			
gripper-3	15		$23.00\,^{23}_{23}$	$23.00\,^{23}_{23}$	$23.00\,^{23}_{23}$
log-16-0	8	$122.74\,^{105}_{131}$			
log-16-0	13		$146.47\,^{125}_{167}$	$123.91\,^{106}_{141}$	$125.32\,^{108}_{143}$
freecell5-4	13	$32.99\,^{30}_{35}$	$32.65\,^{30}_{33}$	$34.02\,^{31}_{35}$	$32.46\,^{30}_{33}$
elev-str-f24	17	$58.38\,^{51}_{63}$			
elev-str-f24	32		$40.00\,^{40}_{40}$	$40.00\,^{40}_{40}$	$40.00\,^{40}_{40}$
satel-17	4	$191.55\,^{82}_{274}$			
satel-17	6		$95.00\,^{83}_{106}$	$122.14\,^{92}_{158}$	$96.73\,^{85}_{105}$
sched-30-0	11	$43.88\,^{40}_{50}$	$50.79\,^{45}_{53}$	$45.06\,^{39}_{53}$	$41.63\,^{38}_{46}$
zeno-5-10	4	$34.36\,^{34}_{35}$			
zeno-5-10	6		$43.32\,^{38}_{48}$	$46.80\,^{38}_{58}$	$40.63\,^{35}_{46}$

Table 3.2: Numbers of operators in plans

quadratic ∀-step semantics encoding in Section 3.8.2, and the column ∀-*step l.* for the linear ∀-step semantics encoding in Section 3.8.2.

Runtimes for ∃-step semantics are in most cases reported on their own lines because its shortest plan lengths differ from the other semantics. Each runtime is followed by the upper and lower bounds of the 95 percent confidence intervals. We indicate by a dash - the formulae for which not all runs finished within 180 seconds.

In Table 3.2 we compare the semantics in terms of the number of operators in plans. Blocks World problems are sequential (only one operator can be applied at a time) and plan lengths equal the number of time points. The average number of operators is followed by the lowest and the highest number of operators any plan we found had.

In Table 3.3 we present data on formula sizes.

The lowest runtimes are usually obtained with the ∃-step semantics. It is often one or two orders of magnitude faster. For problem instances that are more difficult than those depicted in the tables the runtime differences are still bigger. Most of the benchmark problems allow parallelism, and in most of these cases ∃-step semantics allows more operators in parallel than the ∀-step semantics. For example in many of the problems involving transportation of objects by vehicles, with ∃-step semantics a vehicle can leave a location simultaneously with loading or unloading an object to or from the vehicle. The smaller parallel plan lengths directly lead to much faster planning.

For the Schedule benchmark ∃-step semantics does not allow more parallelism than the ∀-step semantics. The linear-size ∃-step semantics is as efficient as the linear-size ∀-step semantics encoding, and slightly less efficient than the quadratic-size ∀-step semantics encoding as far as the unsatisfiable formulae are concerned. Interestingly, the relative efficiency of the encodings reverses for satisfiable formulae corresponding to plans. As shown in Table 3.4, for satisfiable formulae the SAT solver runtimes more closely reflect the relative sizes of the encodings: the linear-size ∃-step encoding is the fastest, followed by the linear-size ∀-step encoding and the quadratic

instance	len	∃-step			process			∀-step			∀-step l.		
		$\frac{P}{10^3}$	$\frac{C}{10^3}$	MB	$\frac{P}{10^3}$	$\frac{C}{10^3}$	MB	$\frac{P}{10^3}$	$\frac{C}{10^3}$	MB	$\frac{P}{10^3}$	$\frac{C}{10^3}$	MB
block-18-0	58	58.9	696.8	10.9	264.9	1218.8	24.9	58.9	696.8	10.9	201.0	1120.1	18.9
block-20-0	60	74.9	937.8	14.8	338.4	1607.2	32.9	74.9	937.8	14.8	257.7	1482.0	25.2
block-22-0	72	108.3	1431.0	22.9	490.5	2406.9	49.5	108.3	1431.0	22.9	374.6	2225.5	38.4
depot-17-6587	7	24.1	256.3	3.9	154.7	611.6	12.8	24.1	269.8	4.1	144.2	586.2	9.7
depot-18-1916	12	75.7	864.9	13.7	484.2	2052.0	45.4	75.7	899.4	14.2	457.7	1968.3	34.2
depot-15-4534	20	93.0	882.8	14.5	594.8	2360.2	53.7	93.0	918.8	15.0	550.2	2243.9	39.4
driver-2-3-6e	12	25.4	110.6	1.6	66.2	206.4	3.7	21.5	157.0	2.3	42.9	174.1	2.6
driver-3-3-6b	11	22.3	93.4	1.4	54.9	178.2	3.3	18.0	144.9	2.1	39.2	153.5	2.3
driver-4-4-8	11	48.5	210.7	3.3	117.0	401.3	7.7	37.7	382.5	5.8	89.4	352.3	5.4
gripper-2	11	1.0	4.7	0.1	2.9	8.8	0.1	1.0	5.3	0.1	1.5	7.2	0.1
gripper-3	15	1.8	8.7	0.1	5.0	16.1	0.3	1.8	9.7	0.1	2.7	13.2	0.2
gripper-4	17	2.4	12.5	0.2	6.9	23.1	0.4	2.4	13.9	0.2	3.7	19.0	0.3
log-16-0	13	18.7	105.4	1.5	46.6	174.3	3.1	18.7	139.1	2.0	27.0	146.3	2.2
log-20-0	15	29.1	174.8	2.5	72.4	284.6	5.1	29.1	236.6	3.5	42.5	240.6	3.6
log-24-0	15	37.8	240.8	3.5	94.3	385.1	6.9	37.8	333.0	4.9	55.9	328.2	5.0
elev/str-f8	12	1.0	2.4	0.0	4.1	8.1	0.1	1.0	3.0	0.0	2.1	5.7	0.1
elev/str-f12	14	2.4	5.8	0.1	10.3	21.4	0.3	2.4	7.7	0.1	5.7	15.6	0.2
elev/str-f16	22	6.4	15.7	0.2	27.8	60.7	1.1	6.4	21.0	0.3	16.2	44.7	0.7
elev/str-f20	26	11.5	28.4	0.4	50.5	112.5	2.0	11.5	38.3	0.6	30.2	83.7	1.3
elev/str-f24	28	17.5	43.4	0.7	77.5	174.7	3.1	17.5	58.9	0.9	47.2	131.0	2.0
satel-14	8	37.7	129.6	2.0	108.1	347.0	6.7	37.7	267.0	4.1	98.5	309.4	4.8
satel-15	8	49.0	168.5	2.7	142.0	454.0	9.2	49.0	327.3	5.1	130.1	405.3	6.6
satel-16	6	46.8	161.5	2.6	136.6	430.1	8.6	46.8	333.7	5.2	125.7	386.3	6.3
satel-17	6	54.0	185.6	3.0	160.6	500.1	10.1	54.0	346.7	5.4	148.5	449.8	7.5
satel-18	8	31.7	108.5	1.7	91.3	290.2	5.5	31.7	221.1	3.4	82.4	258.1	4.0
sched-10-0	7	7.3	40.2	0.6	16.5	58.4	1.1	3.4	73.5	1.0	11.3	53.0	0.8
sched-20-0	9	18.2	101.5	1.6	40.7	148.4	3.0	8.4	285.0	3.9	28.5	134.8	2.1
sched-30-0	11	32.9	185.3	3.0	72.9	271.7	5.5	15.1	700.0	10.3	51.4	246.6	3.9
sched-40-0	15	58.8	334.3	5.4	129.6	492.4	10.9	27.0	1595.3	24.6	91.8	445.9	7.1
sched-50-0	17	82.7	480.3	7.8	182.0	704.7	15.9	38.0	2720.7	42.0	129.2	638.5	10.9
zeno-3-8b	6	9.1	49.0	0.7	42.0	139.4	2.6	9.1	144.1	2.0	39.5	130.7	2.0
zeno-5-10	6	39.2	220.8	3.6	195.2	653.8	13.9	39.2	814.8	12.3	190.1	618.9	10.5
zeno-5-15	6	59.0	332.7	5.5	291.0	979.0	21.1	59.0	1639.5	25.0	283.6	926.6	16.0
zeno-5-15b	6	78.0	309.5	5.5	391.9	1182.9	26.3	78.0	2111.3	32.6	383.6	1114.4	19.5

Table 3.3: Sizes of formulae under the different encodings. The column $\frac{P}{10^3}$ gives the number of propositional variables in thousands, the column $\frac{C}{10^3}$ the number of clauses in thousands, and the column MB the size of the DIMACS encoded formulae in CNF in megabytes. The data are on the satisfiable formulae corresponding to the length of shortest existing plans under ∀-step semantics. The shortest ∃-step plans are in many cases shorter, and the required formulae then correspondingly smaller.

size $\forall$-step encoding.

Numbers of operators in plans for the different encodings do not seem to follow any regular pattern. In same cases the process semantics plans have the most operators, in others the $\forall$-step or the $\exists$-step plans.

Contrary to our expectations based on the earlier results by Heljanko [2001] on Petri net deadlock detection problems, process semantics does not provide an advantage over $\forall$-step semantics on these problems 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.

Differences to the results by Heljanko are likely to be because of differences between the application area and the type of SAT solvers and encodings used. The problem with the planning problems would appear to be the high number of long clauses that usually do not lead to pruning the search space and just add an overhead.

It is interesting to make a comparison between the quadratic and linear size encodings of the $\forall$-step semantics constraints respectively discussed in Sections 3.8.2 and 3.8.2. Even though the worst-case formula sizes are smaller with the linear encoding, this did not directly translate into smaller formulae and improved runtimes. First of all, even though the encoding from Section 3.8.2 is worst-case quadratic, the number of clauses $\neg o \vee \neg o'$ is often small because not all pairs of operators interfere. Also many pairs of interfering operators cannot be simultaneously applied, and hence the corresponding clauses are not needed.

The only benchmark series in which the linear-size encoding substantially improves on the worst-case quadratic-size encoding is Schedule. This is because in this benchmark there is a very high number of pairs of interfering operators that can be applied simultaneously, and the quadraticity therefore very clearly shows up. Hence the linear-size encoding leads to much smaller formulae. Better runtimes are however obtained only for plan lengths higher than the shortest existing plans, as shown in Figure 3.4. On still bigger instances the differences become still more pronounced. These differences between the linear and quadratic size encodings often mean much bigger differences in total runtimes on planners that use more sophisticated evaluation algorithms than the standard sequential one, for example the algorithm we consider in Section 3.9.3.

Some of the sizes of SCCs of disabling graphs are depicted in Table 3.5. We only give the SCC sizes for one instance of each benchmark series because the SCC sizes for all instances of each series are similar. For example, all SCCs of all instances of the Blocks World, Depot, Gripper, Elevator, Logistics, Satellite and ZenoTravel have size 1. For the other benchmarks, the SCC sizes are a function of some of the problem parameters, like the number of vehicles.

Only few or no constraints on parallel operators are needed if all the strong components of the disabling graphs are small. This directly contributes to the small size of the formulae for the $\exists$-step semantics. However, it is not clear whether this per se is a reason for the efficiency of $\exists$-step semantics. For problems in which shortest $\exists$-step and shortest $\forall$-step plans have the same length, for example the blocks world problems, $\exists$-step encoding is not more efficient than the corresponding $\forall$-step semantics encoding.

The BLACKBOX planner by Kautz and Selman [1999] is the best-known planner that implements the planning as satisfiability paradigm[5]. Our quadratic encoding of the $\forall$-step semantics (Section 3.8.2) is closest to the planning graph based encoding used in the BLACKBOX planner.

[5]Surprisingly, the SAT encodings of planning by the SATPLAN04 planner of Kautz et al. (unpublished work) which participated in the 2004 planning competition are for many benchmark problems much slower than the BLACKBOX encodings, and only in few cases it is somewhat faster.

instance	len	val	∃-step	∀-step	∀-step l.
sched-35-0	13	T	$3.43^{2.65}_{4.33}$	$3.86^{3.13}_{4.65}$	$3.14^{2.46}_{3.95}$
sched-35-0	14	T	$2.10^{1.78}_{2.44}$	$3.08^{2.63}_{3.62}$	$1.63^{1.37}_{1.90}$
sched-35-0	15	T	$1.39^{1.20}_{1.57}$	$2.81^{2.41}_{3.26}$	$1.83^{1.58}_{2.13}$
sched-35-0	16	T	$1.41^{1.22}_{1.62}$	$2.30^{1.99}_{2.65}$	$1.43^{1.22}_{1.69}$
sched-35-0	17	T	$1.28^{1.13}_{1.43}$	$3.08^{2.66}_{3.53}$	$1.43^{1.24}_{1.63}$
sched-35-0	18	T	$1.22^{1.07}_{1.37}$	$3.95^{3.28}_{4.82}$	$1.52^{1.26}_{1.86}$
sched-35-0	19	T	$1.20^{1.04}_{1.37}$	$5.62^{4.73}_{6.56}$	$1.40^{1.25}_{1.54}$
sched-35-0	20	T	$1.31^{1.17}_{1.46}$	$4.77^{4.18}_{5.39}$	$1.41^{1.24}_{1.59}$
sched-35-0	21	T	$1.04^{0.90}_{1.19}$	$4.80^{4.26}_{5.36}$	$1.07^{0.93}_{1.24}$
sched-35-0	22	T	$1.37^{1.20}_{1.58}$	$14.97^{13.44}_{16.58}$	$1.38^{1.23}_{1.54}$
sched-35-0	23	T	$1.16^{1.02}_{1.31}$	$6.17^{5.36}_{7.05}$	$1.26^{1.10}_{1.44}$
sched-35-0	24	T	$1.64^{1.44}_{1.83}$	$10.14^{8.89}_{11.51}$	$2.13^{1.85}_{2.42}$
sched-35-0	25	T	$1.68^{1.47}_{1.90}$	$20.52^{18.31}_{22.69}$	$1.83^{1.58}_{2.12}$
sched-35-0	26	T	$1.54^{1.37}_{1.71}$	$17.65^{15.64}_{19.71}$	$2.11^{1.82}_{2.42}$
sched-35-0	27	T	$1.77^{1.53}_{2.02}$	$13.46^{11.74}_{15.36}$	$1.56^{1.34}_{1.80}$
sched-35-0	28	T	$1.56^{1.38}_{1.76}$	$22.96^{20.09}_{26.10}$	$2.22^{1.86}_{2.64}$

Table 3.4: Runtimes for the satisfiable formulae for different plan lengths

instance	SCCs
block-34-0	2312×1
depot-22-1817	22252×1
grip-5	98×1
elev/str-f60	3600×1
log-41-0	7812×1
satel-20	4437×1
zeno-5-25b	31570×1
driver-4-4-8	16×10 16×9 32×8 48×7 16×6 32×5 32×4 1312×1
sched-51-0	1×1173 1×51 1×1
freecell8-4	1×6882 99×1

Table 3.5: Sizes of SCCs of Disabling Graphs: $n \times m$ means that there are n SCCs of size m.

We give a comparison between the runtimes for our quadratic $\forall$-step semantics encoding and the encoding used by BLACKBOX in Table 3.6,[6] and between the formula sizes in Table 3.7.

The planning graph [Blum and Furst, 1997] is a data structure that represents constraints $\neg o_t \vee \neg o'_t$ for pairs of interfering operators, 2-literal invariants, as well as 1-literal and 2-literal clauses that indicate that certain values of state variables and application of certain operators are not possible at given time points. The 2-literal clauses in planning graphs are called *mutexes*. A peculiarity of planning graphs is the NO-OP operators which are used as a marker for the fact that a given state variable does not change its value. The problem encoding used by BLACKBOX is based on translating the contents of planning graphs into 1-literal and 2-literal clauses.

For some of the easiest problems the BLACKBOX encoding is more efficient than the quadratic $\forall$-step semantics encoding (the Logistics problems and some instances of the Depot problem), but in many cases it is much less efficient, most notably on the Blocks World, Driver and Gripper problems. We believe that BLACKBOX's efficiency on the easier problems is due to the explicit reachability information in the planning graph that with our $\forall$-step semantics encoding has to be inferred, and the inefficiency in general is due to the bigger formulae BLACKBOX produces.

The BLACKBOX encoding results in much bigger formulae than the quadratic $\forall$-step encoding, for the biggest instances by factors up to 25. The main reason for this is the very straightforward translation of planning graphs into propositional formulae BLACKBOX uses. This includes many redundant interference mutexes for operators that can also be otherwise inferred not to be simultaneously applicable as well as many mutexes between NO-OPs and operators.

The $\forall$-step semantics formulae often have almost twice as many propositional variables as the BLACKBOX formulae. This is due to the reachability information in the planning graphs that allows to infer that only certain operators are applicable and that only certain state variable values are possible at some of the early time points. Roughly the same reduction could be obtained for our $\forall$-step semantics formulae by performing unit resolution and then eliminating all occurrences of propositional variables occurring in a unit clause by subsumption.

We conclude that the BLACKBOX encoding is roughly comparable to our quadratic encoding for the $\forall$-step semantics and hence in many cases much less efficient than our encoding for the $\exists$-step semantics. Further, the formulae for the BLACKBOX encoding are often several times bigger.

3.9 Evaluation algorithms

Earlier research on classical planning that splits plan search into finding plans of given fixed lengths, for instance the GraphPlan algorithm [Blum and Furst, 1997] and planning as satisfiability [Kautz and Selman, 1996] and related approaches [Rintanen, 1998; Kautz and Walser, 1999; Wolfman and Weld, 1999; van Beek and Chen, 1999; Do and Kambhampati, 2001], have without exception adopted a sequential strategy for plan search. This strategy starts with (parallel) plan length 0, and if no such plans exist, continues with length 1, length 2, and so on, until a plan is found.

This standard sequential strategy is guaranteed to find a plan with the minimal number of time points. If only one operator is applied at every time point then the plans are also guaranteed to contain the minimal number of operators.

It seems that for finding a plan with the minimal number of time points the sequential strategy

[6] We were not able to test all the benchmarks with BLACKBOX because of certain bugs in BLACKBOX.

instance	len	val	∀-step	blackbox
block-12-1	33	F	$0.06\,^{0.06}_{0.06}$	$0.20\,^{0.20}_{0.21}$
block-12-1	34	T	$0.05\,^{0.05}_{0.05}$	$0.22\,^{0.21}_{0.23}$
block-14-1	35	F	$0.35\,^{0.34}_{0.35}$	$18.02^{16.91}_{19.29}$
block-14-1	36	T	$0.12\,^{0.11}_{0.14}$	$5.65\,^{4.97}_{6.39}$
block-16-1	53	F	$0.65\,^{0.63}_{0.68}$	$33.38^{31.10}_{35.66}$
block-16-1	54	T	$0.38\,^{0.36}_{0.40}$	$13.85^{12.52}_{15.25}$
block-18-0	57	F	$2.29\,^{2.22}_{2.36}$	–
block-18-0	58	T	$1.07\,^{0.98}_{1.17}$	$24.15^{21.61}_{26.90}$
log-17-0	13	F	$1.97\,^{1.89}_{2.05}$	$0.42\,^{0.40}_{0.44}$
log-17-0	14	T	$3.22\,^{2.91}_{3.56}$	$1.06\,^{0.91}_{1.22}$
log-18-0	14	F	$5.83\,^{5.50}_{6.18}$	$3.25\,^{2.98}_{3.55}$
log-18-0	15	T	$7.84\,^{6.74}_{9.07}$	$2.21\,^{1.86}_{2.59}$
log-19-0	14	F	$11.22^{10.52}_{12.01}$	$4.55\,^{4.30}_{4.82}$
log-19-0	15	T	$29.10^{25.25}_{33.00}$	$13.74^{11.99}_{15.53}$
log-20-0	14	F	$10.63\,^{9.96}_{11.36}$	$7.88\,^{7.52}_{8.26}$
log-20-0	15	T	–	$15.94^{13.97}_{18.01}$
depot-14-7654	11	F	$1.41\,^{1.34}_{1.48}$	$0.30\,^{0.28}_{0.31}$
depot-14-7654	12	T	$3.48\,^{3.19}_{3.78}$	$1.17\,^{1.06}_{1.29}$
depot-16-4398	7	F	$0.01\,^{0.01}_{0.01}$	$0.01\,^{0.01}_{0.01}$
depot-16-4398	8	T	$0.07\,^{0.06}_{0.07}$	$0.01\,^{0.01}_{0.01}$
depot-18-1916	11	F	$0.17\,^{0.16}_{0.17}$	$28.41^{23.75}_{33.44}$
depot-18-1916	12	T	–	–
driver-2-3-6d	15	F	$19.09^{18.22}_{19.98}$	$43.44^{40.04}_{46.98}$
driver-2-3-6d	16	T	$8.04\,^{7.16}_{9.00}$	$18.94^{17.72}_{20.16}$
driver-2-3-6e	11	F	$1.13\,^{1.07}_{1.19}$	$0.60\,^{0.56}_{0.63}$
driver-2-3-6e	12	T	$1.27\,^{1.10}_{1.45}$	$1.51\,^{1.28}_{1.73}$
driver-3-3-6b	10	F	$0.82\,^{0.77}_{0.87}$	$0.60\,^{0.56}_{0.65}$
driver-3-3-6b	11	T	$1.07\,^{0.90}_{1.25}$	$0.76\,^{0.64}_{0.89}$
driver-4-4-8	10	F	$1.30\,^{1.26}_{1.33}$	$0.56\,^{0.52}_{0.61}$
driver-4-4-8	11	T	$5.92\,^{5.34}_{6.50}$	$19.35^{17.88}_{20.89}$
gripper-2	10	F	$0.08\,^{0.08}_{0.09}$	$0.34\,^{0.32}_{0.36}$
gripper-2	11	T	$0.02\,^{0.01}_{0.02}$	$0.12\,^{0.10}_{0.15}$
gripper-3	14	F	$3.91\,^{3.49}_{4.35}$	$41.07^{36.15}_{45.84}$
gripper-3	15	T	$0.32\,^{0.19}_{0.46}$	$2.82\,^{2.08}_{3.63}$

Table 3.6: Runtimes of the quadratic ∀-step semantics encoding vs. the BLACKBOX encoding

instance	len	∀-step			blackbox		
		$\frac{P}{10^3}$	$\frac{C}{10^3}$	MB	$\frac{P}{10^3}$	$\frac{C}{10^3}$	MB
block-12-1	34	15.71	152.4	2.23	13.12	1035.3	14.80
block-14-1	36	22.44	233.7	3.47	24.88	2938.5	44.59
block-16-1	54	43.54	485.1	7.51	42.73	6012.7	94.72
block-18-0	58	58.88	696.8	10.92	61.79	11091.9	176.58
log-17-0	14	20.12	149.8	2.16	10.41	431.8	6.15
log-19-0	15	29.08	236.6	3.46	15.47	897.1	12.98
log-21-0	16	30.98	252.3	3.70	19.97	1301.6	19.43
log-23-0	16	40.22	355.1	5.29	24.13	1973.5	29.97
log-25-0	15	56.66	556.3	8.42	28.70	3419.9	52.70
depot-14-7654	12	30.99	357.5	5.52	12.79	1952.6	27.96
depot-16-4398	8	13.72	143.5	2.10	4.12	237.7	3.33
depot-18-1916	12	75.67	899.4	14.18	33.42	14599.4	230.82
driver-2-3-6d	16	23.00	168.5	2.51	15.60	1809.6	26.44
driver-2-3-6e	12	21.52	157.0	2.32	11.45	1432.2	20.47
driver-3-3-6b	11	17.97	144.9	2.12	8.86	972.9	13.87
driver-4-4-8	11	37.73	382.5	5.81	15.54	3406.7	49.92
gripper-2	11	1.01	5.3	0.06	1.15	15.2	0.19
gripper-3	15	1.76	9.7	0.12	2.13	36.7	0.48
gripper-4	19	2.72	15.5	0.20	3.39	71.6	0.97

Table 3.7: Formula sizes of the quadratic ∀-step semantics encodings vs. the BLACKBOX encoding

cannot in general be improved. For example, a strategy that increases the plan length by more than one until a satisfiable formula is found and then performs a binary search to find the shortest plan does not typically improve runtimes because the cost of evaluating the unsatisfiable formulae usually increases exponentially as the plan length increases.

However, when the objective is to find any plan, not necessarily with the minimal number of time points, we can use more efficient search strategies for plan search. The standard sequential strategy is often inefficient because the satisfiability tests for the last unsatisfiable formulae are often much more expensive than for the first satisfiable formulae. Consider the diagrams in Figures 3.9 and 3.10 that represent some standard benchmarks problems as well as the diagrams in Figure 3.11 that represent two difficult problem instances with 20 state variable sampled from the phase transition region [Rintanen, 2004d]. Each diagram shows the cost of detecting the satisfiability or unsatisfiability of formulae that represent the existence of plans of different lengths. Black bars depict unsatisfiable formulae and grey bars satisfiable formulae.

When the plan quality (the number of time points) is not a concern, we would like to run a satisfiability algorithm with the satisfiable formula for which the runtime of the algorithm is the lowest. Of course, we do not know which formulae are satisfiable and which have the lowest runtime. With an infinite number of processors we could find in the smallest possible time a satisfying assignment for one of the formulae: just let each processor $i \in \{0, 1, 2, \ldots\}$ test the satisfiability of the formula for i time points. However, we do not have an infinite number of processors, and we cannot even simulate an infinite number of processors running at the same speed by a finite number of processors. But we can approximate this scheme.

Our first algorithm uses a finite number n of processes/processors. Our second algorithm uses

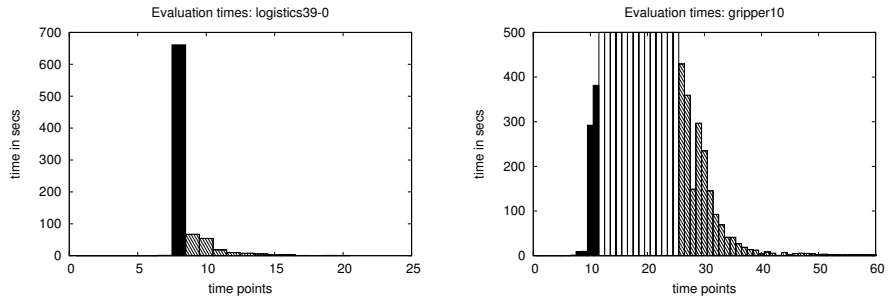


Figure 3.9: SAT solver runtimes for two problem instances and different plan lengths

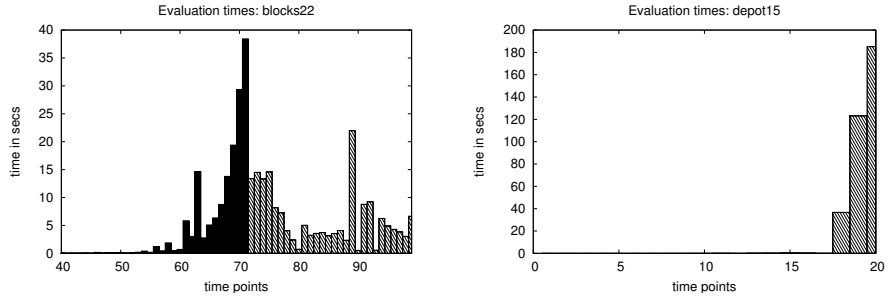


Figure 3.10: SAT solver runtimes for two problem instances and different plan lengths

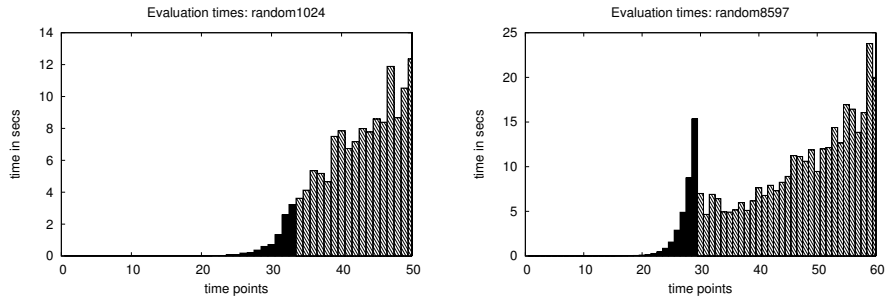


Figure 3.11: SAT solver runtimes for two problem instances and different plan lengths

instance	len	val	∃-step	process	∀-step	∀-step l.
gripper-2	5	F	$0.01\,^{0.01}_{0.01}$			
gripper-2	6	T	$0.01\,^{0.01}_{0.01}$			
gripper-2	10	F		$0.14\,^{0.13}_{0.15}$	$0.08\,^{0.08}_{0.09}$	$0.12\,^{0.12}_{0.13}$
gripper-2	11	T		$0.04\,^{0.03}_{0.04}$	$0.02\,^{0.01}_{0.02}$	$0.05\,^{0.04}_{0.05}$
gripper-3	7	F	$0.23\,^{0.23}_{0.24}$			
gripper-3	8	T	$0.17\,^{0.16}_{0.18}$			
gripper-3	14	F		$9.39\,^{8.43}_{10.47}$	$3.91\,^{3.48}_{4.35}$	$8.84\,^{7.84}_{9.93}$
gripper-3	15	T		$1.72\,^{1.18}_{2.34}$	$0.32\,^{0.19}_{0.47}$	$0.69\,^{0.36}_{1.08}$
gripper-4	9	F	$12.87^{11.61}_{14.26}$			
gripper-4	10	T	$0.85\,^{0.70}_{1.02}$			
gripper-4	16	F		–	–	–
gripper-4	17	?				
gripper-4	18	?				
gripper-4	19	T		–	–	–

Table 3.8: Runtimes of Gripper problems

one process/processor to simulate an infinite number of processes but the simulation runs the processes at variable rates so that for every formula ϕ_t and every $k \geq 0$ there is a time point when k seconds of CPU time has been spent for testing the satisfiability of ϕ_t. If all processes were run at the same rate, this property could not be fulfilled.

Except for the rightmost diagram in Figure 3.10 and the leftmost diagram in Figure 3.11, the diagrams depict steeply growing costs of determining unsatisfiability of a sequence of formulae followed by small costs of determining satisfiability of formulae corresponding to plans. This pattern could be abstracted as the diagram in Figure 3.12. The strategy implemented by our first algorithm distributes the computation to n concurrent processes and initially assigns the first n formulae to the n processes. Whenever a process finds its formula satisfiable, the computation is terminated. Whenever a process finds its formula unsatisfiable, the process is given the first unevaluated formula to evaluate. This strategy can avoid completing the evaluation of many of the expensive unsatisfiable formulae, thereby saving a lot of computation effort.

An inherent property of the problem is that unsatisfiable (respectively satisfiable) formulae later in the sequence are in general more expensive to evaluate than earlier unsatisfiable (respectively satisfiable) formulae. The difficulty of the unsatisfiable formulae increases as i increases because the formulae become less constrained, contradictions are not found as quickly, and search trees grow exponentially. The increase in the difficulty of satisfiable formulae is less clear. For example, for the first satisfiable formula ϕ_s there may be few plans while for later formulae there may be many plans, and the formulae would be less constrained and easier to evaluate. However, as formula sizes increase, the possibility of getting lost in parts of the search space that do not contain any solutions also increases. Therefore an increase in plan length also later leads to an increase in difficulty.

The new algorithms are useful if a peak of difficult formulae precedes easier satisfiable formulae, for example when it is easier to find a plan of length n than to prove that no plans of length $n-1$ exists, or if the first strongly constrained satisfiable formulae corresponding to the shortest plans are more difficult to evaluate than some of the later less constrained ones. The experiments

instance	len	val	∃-step	process	∀-step	∀-step l.
satel-14	4	F	$9.43^{8.81}_{10.12}$			
satel-14	5	T	$1.79^{1.66}_{1.91}$			
satel-14	7	F		$38.20^{36.42}_{40.13}$	$29.59^{28.17}_{31.12}$	$30.95^{29.59}_{32.37}$
satel-14	8	T		$6.20^{5.83}_{6.61}$	$4.38^{4.06}_{4.73}$	$5.82^{5.31}_{6.37}$
satel-15	4	F	$10.44^{9.36}_{11.66}$			
satel-15	5	T	$1.60^{1.45}_{1.75}$			
satel-15	7	F		$33.04^{30.92}_{35.37}$	$26.58^{25.32}_{27.87}$	$28.11^{26.72}_{29.59}$
satel-15	8	T		$7.53^{7.17}_{7.90}$	$4.83^{4.58}_{5.10}$	$6.23^{5.93}_{6.55}$
satel-16	3	F	$1.73^{1.54}_{1.93}$			
satel-16	4	T	$3.36^{3.16}_{3.57}$			
satel-16	5	F		$20.34^{18.68}_{22.04}$	$8.80^{8.10}_{9.53}$	$20.09^{18.61}_{21.74}$
satel-16	6	?				
satel-16	7	T		$8.87^{8.21}_{9.55}$	$7.88^{7.42}_{8.39}$	$7.81^{7.35}_{8.29}$
satel-17	3	F	$0.28^{0.25}_{0.30}$			
satel-17	4	T	$2.85^{2.81}_{2.90}$			
satel-17	5	F		$2.74^{2.46}_{3.08}$	$1.45^{1.32}_{1.63}$	$1.72^{1.66}_{1.78}$
satel-17	6	T		$3.46^{3.22}_{3.71}$	$2.22^{2.10}_{2.35}$	$2.53^{2.37}_{2.69}$
satel-18	4	F	$0.07^{0.07}_{0.07}$			
satel-18	5	T	$0.22^{0.20}_{0.24}$			
satel-18	7	F		$0.60^{0.57}_{0.63}$	$0.30^{0.29}_{0.31}$	$0.54^{0.52}_{0.57}$
satel-18	8	T		$1.18^{1.08}_{1.27}$	$0.54^{0.49}_{0.58}$	$0.86^{0.78}_{0.93}$

Table 3.9: Runtimes of Satellite problems

in Section 3.9.5 show that for many problems one or both of these conditions hold.

We discuss the standard sequential algorithm and the two new algorithms in detail next.

3.9.1 Algorithm S: sequential evaluation

The standard algorithm for finding plans in the satisfiability and related approaches to planning tests the satisfiability of formulae for plan lengths 0, 1, 2, and so on, until a satisfiable formula is found [Blum and Furst, 1997; Kautz and Selman, 1996]. This algorithm is given in Figure 3.13. This algorithm, like the ones discussed next, can be extended so that it terminates whenever no plans exist. This is by the observation that with n Boolean state variables there are at most 2^n reachable states and hence if a plan exists, then a plan of length less than 2^n exists. This, however, provides only an impractical termination test. More practical tests exist [Sheeran *et al.*, 2000; McMillan, 2003; Mneimneh and Sakallah, 2003].

3.9.2 Algorithm A: multiple processes

The first new algorithm (Figure 3.14) which we call Algorithm A is based on parallel or interleaved evaluation of a fixed number n of formulae by n processes. As the special case $n = 1$ we have Algorithm S. Whenever a process finishes the evaluation of a formula, it is given the first uneval-

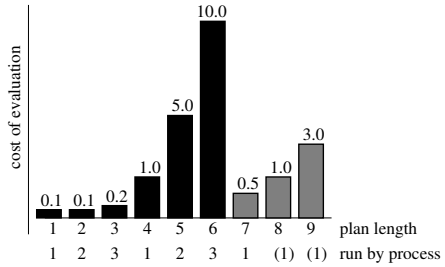


Figure 3.12: Evaluation cost of the unsatisfiable formulae for plan lengths 1 to 6 and the satisfiable formulae for plan length 7 and higher. With 3 processes, process 1 finds the first plan (satisfying assignment) after evaluating the formulae for plan lengths 1, 4 and 7 in 0.1+1+0.5 = 1.6 seconds. This is $3 \times 1.6 = 4.8$ seconds of total CPU time. The sequential strategy needs $0.1 + 0.1 + 0.2 + 1 + 5 + 10 + 0.5 = 16.9$ seconds. With 4 processes a plan would be found by process 3 in $0.2 + 0.5 = 0.7$ seconds, which is $4 \times 0.7 = 2.8$ seconds of total CPU time.

```
1: procedure AlgorithmS()
2: i := 0;
3: repeat
4:    test satisfiability of φ_i;
5:    if φ_i is satisfiable then terminate;
6:    i := i + 1;
7: until 1=0;
```

Figure 3.13: Algorithm S

instance	len	val	∃-step	process	∀-step	∀-step l.
block-12-1	33	F	$0.06\,^{0.05}_{0.06}$	$0.17\,^{0.16}_{0.18}$	$0.06\,^{0.06}_{0.06}$	$0.16\,^{0.16}_{0.17}$
block-12-1	34	T	$0.05\,^{0.04}_{0.05}$	$0.36\,^{0.35}_{0.37}$	$0.05\,^{0.05}_{0.05}$	$0.19\,^{0.18}_{0.20}$
block-14-1	35	F	$0.34\,^{0.34}_{0.35}$	$1.45\,^{1.38}_{1.53}$	$0.35\,^{0.34}_{0.35}$	$1.01\,^{0.98}_{1.05}$
block-14-1	36	T	$0.14\,^{0.12}_{0.15}$	$1.18\,^{1.10}_{1.26}$	$0.12\,^{0.11}_{0.14}$	$0.50\,^{0.46}_{0.53}$
block-16-1	53	F	$0.67\,^{0.65}_{0.69}$	$3.82\,^{3.66}_{4.00}$	$0.65\,^{0.63}_{0.68}$	$1.77\,^{1.69}_{1.85}$
block-16-1	54	T	$0.35\,^{0.33}_{0.37}$	$4.95\,^{4.63}_{5.30}$	$0.38\,^{0.36}_{0.40}$	$1.86\,^{1.76}_{1.98}$
block-18-0	57	F	$1.91\,^{1.85}_{1.98}$	$15.20\,^{14.32}_{16.11}$	$2.29\,^{2.22}_{2.36}$	$6.56\,^{6.33}_{6.80}$
block-18-0	58	T	$0.94\,^{0.87}_{1.01}$	$8.04\,^{7.41}_{8.67}$	$1.07\,^{0.98}_{1.17}$	$3.42\,^{3.19}_{3.66}$
block-20-0	59	F	$2.49\,^{2.37}_{2.61}$	$8.34\,^{8.04}_{8.66}$	$2.57\,^{2.43}_{2.74}$	$5.37\,^{5.09}_{5.66}$
block-20-0	60	T	$1.86\,^{1.79}_{1.92}$	$9.55\,^{9.25}_{9.87}$	$1.80\,^{1.74}_{1.85}$	$4.93\,^{4.66}_{5.21}$
block-22-0	71	F	$38.15\,^{36.82}_{39.48}$	–	$38.49\,^{37.28}_{39.76}$	$51.64\,^{49.58}_{53.78}$
block-22-0	72	T	$14.34\,^{12.88}_{15.82}$	–	$14.32\,^{13.03}_{15.66}$	$26.72\,^{24.83}_{28.64}$

Table 3.10: Runtimes of Blocks World problems

uated formula to evaluate. The constant ϵ determines the coarseness of CPU time division during the evaluation. The *for each* loop in this algorithm and in the next algorithm can be implemented so that several processors are used in parallel.

There is a simple improvement to the algorithm: when formula ϕ_i is found unsatisfiable, the algorithm terminates the evaluation of all ϕ_j for $j < i$ because they must all be unsatisfiable. However, this modification does not usually have any effect because of the monotonically increasing evaluation cost of the unsatisfiable formulae: ϕ_j would already have been found unsatisfiable when ϕ_i with $i > j$ is found unsatisfiable. We ignore this improvement in the following.

3.9.3 Algorithm B: geometric division of CPU use

In Algorithm A the choice of n is determined by the (assumed) width and height of the peak preceding the first satisfiable formulae, and our experiments indicate that small differences in n may make a substantial difference in the runtimes: consider for example the problem instance logistics39-0 in Figure 3.9 for which runtime of Algorithm A with $n = 1$ is more than 10 times the runtime with $n = 2$.

Our second algorithm which we call Algorithm B addresses the difficulty of choosing the value n in Algorithm A. Algorithm B evaluates in an interleaved manner an unbounded number of formulae. The amount of CPU given to each formula depends on its index: if formula ϕ_k is given t seconds of CPU during a certain time interval, then a formula $\phi_i, i \geq k$ is given $\gamma^{i-k}t$ seconds. This means that every formula gets only slightly less CPU than its predecessor, and the choice of the exact value of the constant $\gamma \in]0, 1[$ is far less critical than the choice of n for Algorithm A.

Algorithm B is given in Figure 3.15. Variable t, which is repeatedly increased by δ, characterizes the total CPU time $\frac{t}{1-\gamma}$ available so far. As the evaluation of ϕ_i proceeds only if it has been evaluated for at most $t\gamma^i - \epsilon$ seconds, CPU is actually consumed less than $\frac{t}{1-\gamma}$, and there will be at time $\frac{t}{1-\gamma}$ only a finite number $j \leq \log_\gamma \frac{\epsilon}{t}$ of formulae for which evaluation has commenced.

In a practical implementation of the algorithm, the rate of increase δ of t is increased as the computation proceeds; otherwise the inner *foreach* loop will later often be executed without eval-

```
 1: procedure AlgorithmA(n)
 2: P := {φ_0, ..., φ_{n-1}};
 3: uneval := n;
 4: repeat
 5:   P' := P;
 6:   for each φ ∈ P' do
 7:     continue evaluation of φ for ε seconds;
 8:     if φ was found satisfiable then goto finish;
 9:     if φ was found unsatisfiable then
10:       P := P ∪ {φ_uneval}\{φ};
11:       uneval := uneval + 1;
12:     end if
13:   end do
14: until 0=1
15: finish:
```

Figure 3.14: Algorithm A

```
 1: procedure AlgorithmB(γ)
 2: t := 0;
 3: for each i ≥ 0 do done[i] = false;
 4: for each i ≥ 0 do time[i] = 0;
 5: repeat
 6:   t := t + δ;
 7:   for each i ≥ 0 such that done[i] = false do
 8:     if time[i] + nε ≤ tγ^i for some maximal n ≥ 1 then
 9:       continue evaluation of φ_i for nε seconds;
10:       if φ_i was found satisfiable then goto finish;
11:       time[i] := time[i] + nε;
12:       if φ_i was found unsatisfiable then done[i] := true; end if
13:     end if
14:   end do
15: until 0=1
16: finish:
```

Figure 3.15: Algorithm B

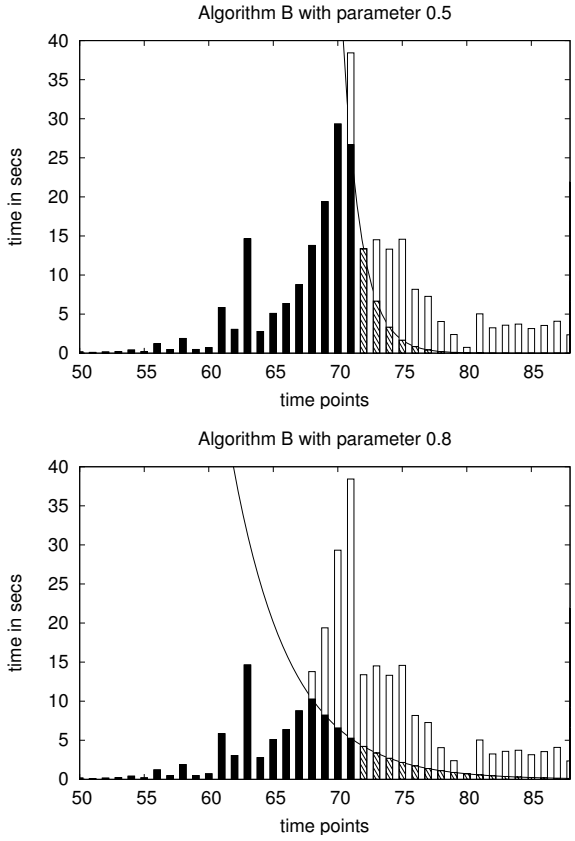


Figure 3.16: Illustration of two runs of Algorithm B. When $\gamma = 0.5$ most CPU time is spent evaluating the first formulae, and when the first satisfiable formula is detected also the unsatisfiability of most of the preceding unsatisfiable formulae has been detected. With $\gamma = 0.8$ more CPU is spent for the later easier satisfiable formulae, and the expensive unsatisfiability tests have not been completed before finding the first satisfying assignment.

instance	len	val	∃-step	process	∀-step	∀-step l.
driver-2-3-6b	4	F	$0.01\,{}^{0.01}_{0.01}$			
driver-2-3-6b	5	T	$0.01\,{}^{0.01}_{0.01}$			
driver-2-3-6b	6	F		$0.04\,{}^{0.04}_{0.04}$	$0.01\,{}^{0.01}_{0.01}$	$0.02\,{}^{0.02}_{0.02}$
driver-2-3-6b	7	T		$0.09\,{}^{0.08}_{0.09}$	$0.03\,{}^{0.02}_{0.03}$	$0.04\,{}^{0.04}_{0.05}$
driver-2-3-6c	6	F	$0.01\,{}^{0.01}_{0.01}$			
driver-2-3-6c	7	T	$0.01\,{}^{0.01}_{0.01}$			
driver-2-3-6c	8	F		$0.03\,{}^{0.03}_{0.03}$	$0.03\,{}^{0.03}_{0.03}$	$0.03\,{}^{0.03}_{0.04}$
driver-2-3-6c	9	T		$0.24\,{}^{0.22}_{0.27}$	$0.10\,{}^{0.09}_{0.11}$	$0.14\,{}^{0.13}_{0.16}$
driver-2-3-6d	12	F	$0.44\,{}^{0.42}_{0.46}$			
driver-2-3-6d	13	T	$0.63\,{}^{0.57}_{0.69}$			
driver-2-3-6d	15	F		$34.14^{32.82}_{35.45}$	$19.09^{18.23}_{20.00}$	$26.27^{25.27}_{27.27}$
driver-2-3-6d	16	T		$17.79^{15.95}_{19.67}$	$8.04\,{}^{7.12}_{8.97}$	$9.59^{8.30}_{11.00}$
driver-2-3-6e	7	F	$0.01\,{}^{0.01}_{0.02}$			
driver-2-3-6e	8	T	$0.04\,{}^{0.04}_{0.04}$			
driver-2-3-6e	11	F		$2.14\,{}^{2.04}_{2.24}$	$1.13\,{}^{1.08}_{1.19}$	$1.55\,{}^{1.47}_{1.62}$
driver-2-3-6e	12	T		$2.54\,{}^{2.29}_{2.80}$	$1.27\,{}^{1.10}_{1.45}$	$1.25\,{}^{1.09}_{1.41}$
driver-3-3-6b	8	F	$0.16\,{}^{0.15}_{0.17}$			
driver-3-3-6b	9	T	$0.08\,{}^{0.07}_{0.09}$			
driver-3-3-6b	10	F		$2.15\,{}^{1.99}_{2.31}$	$0.82\,{}^{0.77}_{0.87}$	$1.40\,{}^{1.31}_{1.51}$
driver-3-3-6b	11	T		$3.26\,{}^{2.77}_{3.83}$	$1.07\,{}^{0.90}_{1.26}$	$1.43\,{}^{1.21}_{1.69}$
driver-4-4-8	8	F	$0.14\,{}^{0.13}_{0.15}$			
driver-4-4-8	9	T	$0.15\,{}^{0.13}_{0.16}$			
driver-4-4-8	10	F		$4.68\,{}^{4.49}_{4.87}$	$1.30\,{}^{1.26}_{1.33}$	$2.84\,{}^{2.75}_{2.94}$
driver-4-4-8	11	T		$23.69^{21.93}_{25.60}$	$5.92\,{}^{5.35}_{6.49}$	$13.08^{11.94}_{14.26}$

Table 3.11: Runtimes of DriverLog problems

uating any of the formulae further. We could choose δ for example so that the first unfinished formula ϕ_i is evaluated further at every iteration ($\delta = \frac{\epsilon}{\gamma^i}$).

The constants n and γ respectively for Algorithms A and B are roughly related by $\gamma = 1 - \frac{1}{n}$: of the CPU capacity $\frac{1}{n} = 1 - \gamma$ is spent evaluating the first unfinished formula, and the lower bound for Algorithm B is similarly related to the lower bound for Algorithm A. Algorithm S is the limit of Algorithm B when γ goes to 0.

3.9.4 Properties of the algorithms

We analyze the properties of the algorithms.

Definition 3.88 (Speed-up) *The* speed-up *of an algorithm X (with respect to Algorithm S) is the ratio of the runtimes of Algorithm S and the Algorithm X.*

If the speed-up is greater than 1, then the algorithm is faster than Algorithm S.

instance	len	val	∃-step	process	∀-step	∀-step l.
sched-10-0	6	F	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
sched-10-0	7	T	$0.01^{0.01}_{0.01}$	$0.07^{0.05}_{0.10}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
sched-15-0	8	F	$8.74^{7.40}_{10.17}$	$14.57^{13.27}_{15.88}$	$3.44^{2.86}_{4.10}$	$11.52^{10.21}_{12.85}$
sched-15-0	9	T	$0.16^{0.12}_{0.22}$	$0.23^{0.19}_{0.27}$	$0.14^{0.11}_{0.16}$	$0.36^{0.27}_{0.44}$
sched-20-0	8	F	$1.11^{1.06}_{1.16}$	$1.42^{1.37}_{1.47}$	$0.46^{0.44}_{0.48}$	$1.30^{1.25}_{1.35}$
sched-20-0	9	T	$0.14^{0.11}_{0.18}$	$0.24^{0.21}_{0.28}$	$0.19^{0.19}_{0.20}$	$0.10^{0.09}_{0.11}$
sched-25-0	8	F	$7.85^{6.87}_{8.96}$	$15.47^{14.61}_{16.37}$	$2.14^{1.96}_{2.35}$	$8.53^{7.57}_{9.56}$
sched-25-0	9	T	$0.29^{0.23}_{0.35}$	$0.68^{0.55}_{0.82}$	$0.19^{0.18}_{0.21}$	$0.69^{0.56}_{0.84}$
sched-30-0	10	F	–	–	$8.12^{5.93}_{10.45}$	–
sched-30-0	11	T	$1.05^{0.78}_{1.36}$	$2.63^{2.22}_{3.04}$	$1.07^{0.88}_{1.29}$	$0.90^{0.65}_{1.19}$
sched-35-0	10	F	$26.22^{24.78}_{27.71}$	$34.35^{32.89}_{35.87}$	$10.26^{9.72}_{10.80}$	$30.09^{28.35}_{31.88}$
sched-35-0	13	T	$3.43^{2.66}_{4.37}$	$3.53^{3.02}_{4.09}$	$3.86^{3.15}_{4.68}$	$3.14^{2.49}_{3.96}$

Table 3.12: Runtimes of Schedule problems

In our analysis we assume that the constant ϵ in Algorithm A is infinitesimally small, and hence, after a process finishes with one formula, the evaluation of the next formula starts immediately, and the algorithm terminates immediately after a satisfiable formula is found.

If there is no peak because the last unsatisfiable formulae are not more difficult than some of the first satisfiable ones, then Algorithm A with $n \geq 2$ may need n times more CPU than Algorithm S because $n-1$ satisfiable formulae are evaluated unnecessarily. We formally establish worst-case bounds for Algorithm A.

Theorem 3.89 *The speed-up of Algorithm A with n processes is at least $\frac{1}{n}$. This lower bound is strict.*

Proof: The worst case $\frac{1}{n}$ can show up in the following situation. Assume the first satisfiable formula is evaluated in time t, the preceding unsatisfiable formulae are evaluated in time 0, and the following satisfiable formulae are evaluated in time $\geq t$. Then the total runtime of Algorithm A is tn, while the total runtime of Algorithm S is t.

Assume the runtimes (CPU time) for the formulae are $t_0, t_1, \ldots, t_s, \ldots$, and ϕ_s is the first satisfiable formula. The total runtime of Algorithm S is $\sum_{i=0}^{s} t_i$. This is also an upper bound on the CPU time consumed by Algorithm A on $\phi_0, \ldots, \phi_s$. Additionally, Algorithm A may spend CPU evaluating $\phi_{s+1}, \phi_{s+2}, \ldots$. The evaluation of these formulae starts at the same time or later than the evaluation of the first satisfiable formula ϕ_s. As $n-1$ processes may spend all their time evaluating these formulae after the evaluation of ϕ_s has started, the total CPU time spent evaluating them may be at most $(n-1)t_s$. Hence Algorithm A spends CPU time at most

$$\sum_{i=0}^{s} t_i + (n-1)t_s$$

in comparison to

$$\sum_{i=0}^{s} t_i$$

instance	len	val	∃-step	process	∀-step	∀-step l.
zeno-3-7b	3	F	$0.01^{0.01}_{0.01}$			
zeno-3-7b	4	T	$0.01^{0.01}_{0.01}$			
zeno-3-7b	5	F		$0.10^{0.10}_{0.10}$	$0.02^{0.02}_{0.02}$	$0.05^{0.05}_{0.06}$
zeno-3-7b	6	T		$0.11^{0.10}_{0.12}$	$0.02^{0.02}_{0.02}$	$0.06^{0.05}_{0.06}$
zeno-3-8	3	F	$0.01^{0.01}_{0.01}$			
zeno-3-8	4	T	$0.01^{0.01}_{0.01}$			
zeno-3-8	5	F		$0.08^{0.08}_{0.09}$	$0.02^{0.02}_{0.02}$	$0.05^{0.05}_{0.05}$
zeno-3-8	6	T		$0.49^{0.45}_{0.53}$	$0.06^{0.06}_{0.07}$	$0.30^{0.27}_{0.33}$
zeno-3-8b	3	F	$0.01^{0.01}_{0.01}$			
zeno-3-8b	4	T	$0.02^{0.01}_{0.02}$			
zeno-3-8b	5	F		$0.17^{0.16}_{0.17}$	$0.03^{0.02}_{0.03}$	$0.11^{0.11}_{0.12}$
zeno-3-8b	6	T		$0.54^{0.47}_{0.61}$	$0.16^{0.16}_{0.16}$	$0.31^{0.27}_{0.36}$
zeno-3-10	4	F	$0.05^{0.05}_{0.05}$			
zeno-3-10	5	T	$0.02^{0.02}_{0.02}$			
zeno-3-10	6	F		$1.77^{1.71}_{1.84}$	$0.51^{0.50}_{0.51}$	$1.17^{1.13}_{1.21}$
zeno-3-10	7	T		$2.68^{2.43}_{2.95}$	$0.76^{0.68}_{0.86}$	$1.79^{1.57}_{2.01}$
zeno-5-10	3	F	$0.10^{0.10}_{0.10}$			
zeno-5-10	4	T	$0.23^{0.19}_{0.28}$			
zeno-5-10	5	F		$2.23^{2.13}_{2.33}$	$1.03^{1.03}_{1.04}$	$1.47^{1.41}_{1.53}$
zeno-5-10	6	T		$9.34^{8.78}_{9.89}$	$3.17^{2.79}_{3.57}$	$6.53^{6.04}_{7.08}$
zeno-5-15	5	F	–			
zeno-5-15	6	T	$21.34^{17.83}_{25.10}$			
zeno-5-15	5	F		$3.52^{3.30}_{3.75}$	$1.76^{1.75}_{1.76}$	$2.32^{2.18}_{2.48}$
zeno-5-15	6	?				
zeno-5-15	7	T		-	$39.34^{35.65}_{43.34}$	-

Table 3.13: Runtimes of ZenoTravel problems

with Algorithm S. The speed-up is therefore at least

$$\frac{\sum_{i=0}^{s} t_i}{\sum_{i=0}^{s} t_i + (n-1)t_s} = \frac{1}{1 + (n-1)\frac{t_s}{\sum_{i=0}^{s} t_i}} \geq \frac{1}{1+n-1} = \frac{1}{n}.$$

□

In the other direction, there is no finite upper bound on the speed-up of Algorithm A in comparison to Algorithm S for any number of processes $n \geq 2$. Consider a problem instance with evaluation time t_0, t_1 and t_2 respectively for the first three formulae, the first two of which are unsatisfiable and the third satisfiable. Let $t_0 = t_2$ and $t_1 = ct_2$. The constant c could be arbitrarily high. Algorithm S runs in $(c+2)t_2$ time, while Algorithm A with $n = 2$ runs in $2t_2$ time. Hence the speed-up $\frac{c+2}{2}$ can be arbitrarily high.

Next we analyze the properties of Algorithm B assuming that the constants δ and ϵ are infinitesimally small and the evaluation of all of the formulae ϕ_i therefore proceeds continuously at rate

instance	len	val	∃-step	process	∀-step	∀-step l.
depot-13-5646	7	F	$0.01\ {}^{0.01}_{0.01}$			
depot-13-5646	8	T	$0.01\ {}^{0.01}_{0.01}$			
depot-13-5646	8	F		$0.02\ {}^{0.02}_{0.02}$	$0.01\ {}^{0.01}_{0.01}$	$0.01\ {}^{0.01}_{0.01}$
depot-13-5646	9	T		$0.27\ {}^{0.26}_{0.29}$	$0.04\ {}^{0.04}_{0.05}$	$0.08\ {}^{0.08}_{0.09}$
depot-14-7654	9	F	$0.05\ {}^{0.05}_{0.06}$			
depot-14-7654	10	T	$0.10\ {}^{0.09}_{0.11}$			
depot-14-7654	11	F		$3.07\ {}^{2.95}_{3.19}$	$1.41\ {}^{1.34}_{1.48}$	$2.17\ {}^{2.07}_{2.29}$
depot-14-7654	12	T		$8.18\ {}^{7.55}_{8.82}$	$3.48\ {}^{3.19}_{3.78}$	$4.26\ {}^{3.87}_{4.66}$
depot-16-4398	7	F	$0.01\ {}^{0.01}_{0.01}$			
depot-16-4398	8	T	$0.01\ {}^{0.01}_{0.01}$			
depot-16-4398	7	F		$0.03\ {}^{0.03}_{0.03}$	$0.01\ {}^{0.01}_{0.01}$	$0.02\ {}^{0.01}_{0.02}$
depot-16-4398	8	T		$0.43\ {}^{0.41}_{0.46}$	$0.07\ {}^{0.06}_{0.07}$	$0.12\ {}^{0.11}_{0.13}$
depot-17-6587	5	F	$0.01\ {}^{0.01}_{0.01}$			
depot-17-6587	6	T	$0.01\ {}^{0.01}_{0.01}$			
depot-17-6587	6	F		$0.24\ {}^{0.23}_{0.26}$	$0.02\ {}^{0.02}_{0.02}$	$0.13\ {}^{0.11}_{0.14}$
depot-17-6587	7	T		$0.69\ {}^{0.65}_{0.74}$	$0.03\ {}^{0.03}_{0.03}$	$0.27\ {}^{0.25}_{0.29}$
depot-18-1916	11	F	$0.29\ {}^{0.28}_{0.29}$			
depot-18-1916	12	T	$5.80\ {}^{5.02}_{6.61}$			
depot-18-1916	11	F		$1.12\ {}^{1.04}_{1.20}$	$0.17\ {}^{0.16}_{0.17}$	$0.51\ {}^{0.48}_{0.54}$
depot-18-1916	12	T		–	–	–

Table 3.14: Runtimes of Depot problems

γ^i.

Theorem 3.90 *The speed-up of Algorithm B is at least* $1-\gamma$*. This lower bound is strict.*

Proof: As with Algorithm A the worst case is reached when all unsatisfiable formulae preceding the first satisfiable formula ϕ_s are evaluated and the evaluation of many of the satisfiable ones has proceeded far. The disadvantage in comparison to Algorithm S is the unnecessary evaluation of many of the satisfiable formulae. Hence Algorithm B spends CPU time at most

$$\sum_{i=0}^{s} t_i + \sum_{i\geq 1} t_s\gamma^i = \sum_{i=0}^{s} t_i + \frac{1}{1-\gamma}t_s - t_s$$

in comparison to

$$\sum_{i=0}^{s} t_i$$

with Algorithm S. The speed-up is therefore at least

$$\begin{aligned}\frac{\sum_{i=0}^{s} t_i}{\sum_{i=0}^{s} t_i + \frac{1}{1-\gamma}t_s - t_s} &= \frac{1}{1+\frac{\frac{1}{1-\gamma}t_s - t_s}{\sum_{i=0}^{s} t_i}} \geq \frac{1}{1+\frac{\frac{1}{1-\gamma}t_s - t_s}{t_s}} \\ &= \frac{1}{1+\frac{1}{1-\gamma}-1} = 1-\gamma.\end{aligned}$$

instance	len	val	∃-step	process	∀-step	∀-step l.
freecell2-4	4	F	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
freecell2-4	5	T	$0.01^{0.01}_{0.01}$	$0.02^{0.01}_{0.02}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
freecell3-4	7	F	$0.45^{0.43}_{0.47}$	$0.77^{0.74}_{0.81}$	$0.25^{0.25}_{0.25}$	$0.53^{0.50}_{0.55}$
freecell3-4	8	T	$0.13^{0.11}_{0.15}$	$0.25^{0.22}_{0.28}$	$0.18^{0.17}_{0.18}$	$0.11^{0.10}_{0.13}$
freecell4-4	6	F	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$	$0.01^{0.01}_{0.01}$
freecell4-4	7	T	$0.05^{0.04}_{0.05}$	$0.12^{0.11}_{0.13}$	$0.02^{0.02}_{0.02}$	$0.08^{0.07}_{0.08}$
freecell5-4	12	F	$13.60^{12.94}_{14.29}$	$17.34^{16.54}_{18.14}$	$6.75^{6.39}_{7.17}$	$9.19^{8.84}_{9.55}$
freecell5-4	13	T	$59.57^{52.44}_{66.68}$	$63.78^{57.28}_{70.33}$	$35.70^{32.55}_{39.06}$	$53.59^{47.44}_{60.17}$

Table 3.15: Runtimes of FreeCell problems

This lower bound is strict: if ϕ_i is satisfiable, evaluation times for $\phi_j, j < i$ are 0, and evaluation times for $\phi_i, i > 1$ are not lower than that of ϕ_1, then the speed-up is only $1 - \gamma$. □

The worst-case speed-ups of these algorithms are the same if we observe the equation $\gamma = 1 - \frac{1}{n}$ relating their parameters.

Algorithm B does not have plan quality guarantees but Algorithm A has.

Theorem 3.91 *If a plan exists, Algorithm A with parameter $n \geq 1$ is guaranteed to find a plan that is at most $n - 1$ steps longer than the shortest existing one.*

Proof: So assume Algorithm A finds a plan with t steps. This means that the process for formula ϕ_t determined that the formula is satisfiable. There are at most $n - 1$ processes for formulae ϕ_s with $s < t$, and all formulae ϕ_s for $s < t$ for which a process terminated are unsatisfiable.

All formulae preceding an unsatisfiable formula are unsatisfiable. Consider formula ϕ_{t-n}.

If the process evaluating ϕ_{t-n} has terminated, the formula must have been unsatisfiable, and hence the plan from ϕ_t is at most $n - 1$ steps longer than the shortest existing one which much have length over $t - n$.

If the process evaluating ϕ_{t-n} has not terminated, then the evaluation of one of the $n - 1$ formulae $\phi_{t-n+1}, \ldots, \phi_{t-1}$ must already have been terminated, because there are n processes and two of them were evaluating ϕ_{t-n} and ϕ_t. Since ϕ_t was the first one found satisfiable, one of the formulae $\phi_{t-n+1}, \ldots, \phi_{t-1}$ that was evaluated was unsatisfiable, and hence the formula ϕ_{t-n} must also be unsatisfiable, yielding the same lower bound for the plan length. □

3.9.5 Experiments

Algorithms A and B increase efficiency for problem instances sampled from the set of all problem instances (Section 3.8.5). The improvements in comparison to Algorithm S are biggest for easy problem instances right of the hardest part of the phase transition region with 100 state variables or more. For the most difficult instances in the middle of the phase transition region the satisfiable formulae are often as difficult as the unsatisfiable ones and hence Algorithms A and B do not seem to bring as much benefit. We did not carry out exhaustive experimentation because of the extremely high computational resource consumption and the difficulty to derive exact characterizations of the

performance improvement when most of the problem instances could not be practically solved by using Algorithm S.

We illustrate properties of the algorithms on a collection of problems from the AIPS planning competitions. Plans for most of these problems can be found in polynomial time by simple domain-specific algorithms, and planners using heuristic search [Bonet and Geffner, 2001] have excelled on these problems, while they had been considered difficult for planners based on satisfiability testing or CSP techniques. In this section we demonstrate that the our new algorithms change this situation.

Many of these benchmark problems follow the same scheme in which objects are transported with vehicles from their initial locations to their target locations (Logistics, Depots, DriverLog, ZenoTravel, Gripper, Elevator), with one of them (Depots) combining transportation with stacking objects as in the well-known Blocks World problem. Some others (Satellite, Rovers) are variations of the transportation scenario in which different locations are visited to carry out some tasks. Some of the benchmark problems have the form of a scheduling problem (Elevator, Schedule) but do not involve any restrictions on resource consumption and therefore only test the property of feasibility which for these problems can be tested in low polynomial time by a simple algorithm.

To demonstrate the usefulness of the algorithms for a wider range of problems, in addition to the planning competition problems which are solvable by simple domain-specific algorithms in polynomial time, we also consider hard instances of an NP-hard planning problem. The planning competition problems are easy because they do not make restrictions on resource consumption and satisfying one subgoal never makes it more difficult to satisfy another. Hence we also consider a planning problem with critical restrictions on resource consumption. We call this problem the Mechanic problem. The objective is to perform a maintenance operation to a fleet of n aircraft. The aircraft fly according to some schedule and visit one of three airports five times during a time period of t days. A mechanic/equipment can be present at one of the three airports on any given day, and can perform the maintenance operation to all aircraft visiting that airport that day. This problem can be viewed as a form of a set covering problem but we make it a sequential decision making problem so that we can talk about completing the maintenance within $t' \leq t$ first days of the time period. We solve the problem for a $t = 30, 40, 50, ...$ and set $n = 3t$. With $n = 3t$ the problem is rather strongly constrained but still usually soluble. For low n it is easier to find a plan because there are only few aircraft, and for much higher n there are too many aircraft and no plan necessarily exists.

For each problem instance we generate formulae for plan lengths up to 10 or 30 beyond the first (assumed) satisfiable formula according to the $\exists$-step semantics encoding in Section 3.8.4. We use the linear-size encoding of the parallelism constraints if it is less than half of the size of the quadratic encoding that does not require introducing auxiliary propositional variables to avoid exceeding Siege's upper bound of 524288 propositional variables.

Then we test the satisfiability of every formula and cancel the run if the satisfiability is not determined in 60 minutes of CPU time. Like in the experiments in Section 3.8.5, we use the Siege V4 SAT solver by Lawrence Ryan of the Simon Fraser University on a 3.6 GHz Intel Xeon computer.

Then we compute from the runtimes of all these formulae the total runtimes under algorithms A and B with different values for the parameters n and γ. Algorithm S is the special case $n = 1$ of Algorithm A. The constants ϵ and δ determining the granularity of CPU time division are set infinitesimally small. Formulae that are beyond the plan-length horizon or that take over 60 minutes to evaluate are considered as having infinite evaluation time. The times do not include

generation of the formulae. The two expensive parts of the formula generation are the computation of the invariants and the disabling graph. For most of the benchmark problem instances these both take a fraction of a second, but for some of the biggest instances of the Logistics, Depot, and Driver problems 10 or 20 seconds, and a total of 6 minutes for the biggest ZenoTravel instance and 3 minutes for the biggest Logistics instance. A more efficient implementation would bring these times down to seconds.

The runtimes for a number of problems from the AIPS planning competitions of 1998, 2000 and 2002 and for the Mechanic problem are given in Table 3.16. For most benchmarks we give the runtimes of the most difficult problem instances, which in some cases are the last ones in the series, as well as some of the easier ones. Most of the runtimes that are not given in the table are below one second for every evaluation strategy. Some of the benchmark series cannot be solved until the end efficiently, and we give data just for some of the most difficult instances that can be solved. We discuss these benchmark problems below.

The column "easiest" gives the shortest time it took to determine the satisfiability of any of the satisfiable formulae corresponding to a plan. These times in almost all cases are very low, even when the total runtime of Algorithms S, A and B is high. Hence in almost all cases the total time it takes to find a plan is strongly dominated by the unsatisfiability tests.

	Algorithm A with n					Algorithm B with γ				easiest	HSP	FF
instance	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375			
block-18-0	8.6	7.8	7.6	**5.8**	6.6	8.0	7.9	**7.7**	9.3	0.1	6.4	–
block-20-0	**11.3**	12.2	13.8	16.9	15.5	13.0	16.5	20.1	18.3	0.2	82.1	0.0
block-22-0	122.4	106.9	96.7	77.0	**35.4**	106.0	62.2	33.5	**27.0**	0.3	79.6	0.5
block-24-0	2877.5	2675.7	1854.0	829.0	**167.4**	2087.3	583.3	284.8	**246.8**	0.7	–	–
block-26-0	5347.5	5000.0	4640.1	3103.9	**539.0**	4116.7	1140.0	242.6	**126.3**	0.9	46.8	0.0
block-28-0	3447.8	3413.4	3246.8	1984.3	**813.3**	2867.0	1746.1	1027.6	**336.4**	1.1	–	27.2
block-30-0	–	–	13949.9	7541.0	**6349.1**	13934.0	6577.4	1717.4	**503.9**	1.9	–	0.0
block-32-0	–	–	–	28695.4	**14326.9**	$> 27h$	36417.3	8182.8	**2245.7**	11.3	–	0.0
block-34-0	227.6	227.8	224.2	231.5	**208.8**	238.4	248.2	264.6	**188.5**	1.9	–	0.1
driver-4-4-8	**0.5**	0.6	**0.3**	0.5	0.8	0.6	0.6	0.7	1.1	0.1	2.9	0.1
driver-5-5-10	731.2	549.5	631.6	**237.7**	440.2	969.8	507.0	**472.4**	651.1	27.5	–	–
driver-5-5-15	72.4	**36.1**	50.4	100.4	200.6	**56.0**	72.7	120.5	219.8	12.5	72.21	–
driver-5-5-20	1018.2	690.1	792.4	940.7	**17.8**	967.5	148.2	35.4	**24.0**	0.5	1428.0	–
driver-5-5-25	–	6433.9	**2218.9**	3542.3	4132.2	4553.4	**4100.7**	5800.5	7865.5	258.2	–	609.5
driver-8-6-25	–	–	13333.9	**11081.4**	22162.6	27447.3	24120.5	**22377.1**	31375.3	1385.2	859.0	–
satel-12	31.1	5.1	**1.4**	1.8	2.7	4.0	**2.5**	3.1	4.6	0.2	3.3	0.1
satel-13	**14.8**	**14.2**	18.2	14.9	17.9	21.0	29.0	24.1	22.8	0.5	10.1	0.3
satel-19	45.1	28.4	21.6	**5.0**	5.6	42.3	13.1	**9.4**	10.1	0.3	8.8	0.3
satel-20	–	1806.4	266.6	**33.0**	35.0	187.1	69.3	**55.3**	63.5	2.1	23.2	4.9
gripper-5	3443.2	1053.7	35.5	7.2	**5.0**	31.7	16.2	2.1	**0.9**	0.0	0.0	0.0
gripper-6	–	–	2679.6	23.4	**10.4**	121.9	45.6	4.1	**1.7**	0.0	0.0	0.0
gripper-7	–	–	–	491.3	**28.3**	1968.0	128.2	7.9	**2.8**	0.0	0.0	0.0
gripper-8	–	–	–	13285.5	**293.1**	57298.9	790.1	27.3	**4.7**	0.0	0.0	0.0
gripper-9	–	–	–	–	**832.6**	$> 27h$	589.7	37.7	**13.0**	0.1	0.1	0.0
gripper-10	–	–	–	–	**216.3**	31496.5	569.3	126.8	**17.1**	0.1	0.1	0.0
gripper-11	–	–	–	–	–	$> 27h$	87479.2	2308.0	**335.4**	0.8	0.1	0.0
gripper-12	–	–	–	–	–	$> 27h$	$> 27h$	8306.4	**1117.5**	0.8	0.1	0.0
zeno-5-10	0.3	0.3	**0.2**	0.2	0.5	0.3	**0.2**	0.3	0.6	0.0	24.2	0.1
zeno-5-15	154.2	77.1	8.7	**2.3**	4.5	17.7	5.1	**4.8**	6.6	0.3	18.5	0.3
zeno-5-15b	40.5	25.3	**7.1**	9.4	9.1	24.4	**14.6**	17.6	17.7	0.5	104.5	0.4
zeno-5-20	–	–	9036.9	6422.6	**2896.0**	16459.9	1364.2	126.8	**64.8**	1.1	188.4	1.4
zeno-5-20b	–	–	–	**10822.9**	18744.6	87164.6	23385.8	**21683.0**	30471.3	1171.5	411.5	1.4
zeno-5-25	–	–	–	**12987.1**	25914.9	$> 27h$	37341.0	**29810.9**	39109.3	1619.7	332.4	4.4
sched-33-0	79.0	53.7	13.0	**5.0**	6.7	22.8	10.9	**10.1**	11.3	0.2	–	0.7
sched-35-0	2225.2	1435.5	19.5	3.6	**2.9**	14.3	7.8	**4.9**	5.2	0.2	–	0.7
sched-37-0	346.2	184.4	92.8	**8.6**	9.6	80.4	24.2	**19.4**	19.5	0.6	–	0.5
sched-39-0	–	–	–	592.2	**140.3**	5889.8	1084.6	437.6	**221.9**	1.9	–	1.7
sched-41-0	–	–	–	479.1	**35.4**	3040.7	237.1	91.7	**80.7**	1.3	–	1.0
sched-43-0	–	1565.2	23.9	**11.6**	17.4	47.3	**20.0**	21.4	23.7	0.4	–	2.2
sched-45-0	–	–	1398.1	109.5	**41.6**	786.6	257.8	100.2	**73.3**	1.5	–	1.0
sched-47-0	–	–	–	14066.9	**245.0**	62768.3	1708.6	607.0	**215.4**	2.2	–	4.3
sched-49-0	–	–	–	9511.7	**561.6**	24913.2	2609.9	426.4	**169.2**	2.1	–	6.0
sched-51-0	–	–	–	–	**1151.2**	$> 27h$	8327.0	1692.6	**889.2**	7.6	–	3.1
depot-09-5451	**14.1**	24.8	43.9	85.8	171.5	24.8	46.3	89.1	174.8	10.7	–	0.7
depot-12-9876	**255.4**	509.7	1018.6	2036.9	4073.6	509.9	1019.1	2037.5	4074.2	254.6	–	3.1
depot-15-4534	**42.8**	79.3	154.8	305.4	609.9	80.9	157.1	309.6	614.4	38.1	–	3.4
depot-18-1916	**5.9**	11.2	21.9	43.5	86.9	11.4	22.2	43.9	87.4	5.4	–	0.8
depot-19-6178	**0.2**	0.2	0.3	0.4	0.6	0.3	0.3	0.5	0.8	0.0	–	0.2
depot-20-7615	34.2	66.7	131.9	262.1	**10.4**	67.0	35.6	**18.5**	18.7	0.4	–	14.3
depot-21-8715	0.2	**0.1**	0.2	0.3	0.6	**0.2**	0.3	0.4	0.7	0.0	51.4	0.3
depot-22-1817	**27.1**	50.8	98.9	194.8	389.4	51.4	100.1	197.5	392.2	24.3	–	55.3
log-20-0	3.4	2.8	0.6	0.7	**0.5**	1.3	1.1	0.9	**0.8**	0.0	2.2	0.1
log-24-0	0.9	**0.4**	0.6	1.0	1.6	**0.6**	0.8	1.2	1.8	0.0	3.1	0.1
log-28-0	87.7	53.3	13.8	**1.9**	3.5	15.0	4.1	**3.8**	3.9	0.1	28.0	1.2
log-32-0	–	53.1	18.9	37.4	**16.3**	37.9	33.7	26.6	**16.6**	0.3	43.4	4.5
log-36-0	–	101.1	20.2	30.1	**11.8**	58.7	46.5	29.2	**14.5**	0.2	81.1	2.6
log-40-0	–	–	111.2	**4.6**	7.2	37.5	10.6	**9.9**	13.5	0.4	267.8	4.5
log-41-0	–	–	52.4	20.0	**5.4**	175.3	14.8	**9.1**	9.8	0.3	247.1	4.2
mechanic-30-90	0.5	0.4	**0.3**	0.4	0.3	**0.4**	0.4	0.4	0.5	0.0	4.1	0.1
mechanic-40-120	0.5	**0.4**	0.5	0.6	0.5	**0.5**	0.6	0.6	0.6	0.0	14.1	0.2
mechanic-50-150	1.1	**1.0**	1.0	1.7	2.9	**1.2**	1.4	2.0	2.7	0.1	226.7	0.4
mechanic-60-180	13.1	6.9	3.3	2.0	**1.0**	4.3	1.6	**1.2**	1.3	0.0	56.46	0.7
mechanic-70-210	4.5	2.9	2.4	**1.0**	1.3	3.3	1.8	**1.5**	1.8	0.1	–	–
mechanic-80-240	2.0	3.0	**1.1**	1.5	2.4	2.4	**1.7**	2.0	2.8	0.1	213.9	3.1
mechanic-90-270	15.1	**2.1**	2.4	3.2	2.9	**3.0**	3.1	3.9	3.9	0.1	339.0	3.6
mechanic-100-300	77.0	47.2	52.2	17.3	**8.0**	64.5	37.5	25.6	**14.7**	0.2	–	–
mechanic-110-330	162.0	192.5	117.8	81.4	**42.2**	164.7	90.2	64.6	**40.3**	0.6	–	–
mechanic-120-360	991.4	717.5	273.5	61.3	**30.6**	185.8	72.8	**56.6**	62.4	0.9	–	–

Table 3.16: Runtimes of Algorithms A and B. Column $n = 1$ is Algorithm S. Dash indicates a missing upper bound on the runtime when some formulae were not evaluated in 60 minutes. The best runtimes for Algorithms A and B are highlighted for each problem instance (sometimes this is the special case Algorithm S.) The column "easiest" shows the lowest runtime for any of the satisfiable formulae.

	Algorithm A with n					Algorithm B with ϵ			
instance	1	2	4	8	16	0.500	0.750	0.875	0.938
blocks-34-0	124 / 124	125 / 124	125 / 124	125 / 124	125 / 124	125 / 124	125 / 124	135 / 129	135 / 129
driver-8-6-25	–	–	12 / 193	14 / 206	14 / 206	11 / 178	14 / 206	14 / 206	14 / 206
satell-20	–	10 / 230	11 / 166	12 / 285	17 / 321	11 / 166	12 / 285	15 / 309	15 / 309
gripper-10	–	–	–	–	25 / 65	25 / 65	25 / 65	49 / 150	49 / 150
zeno-5-15b	5 / 87	5 / 87	7 / 98	9 / 140	14 / 191	7 / 98	7 / 98	14 / 191	14 / 191
sched-37-0	13 / 60	13 / 60	13 / 60	16 / 57	20 / 69	14 / 52	16 / 57	16 / 57	20 / 69
depot-19-6178	10 / 98	10 / 98	10 / 98	10 / 98	10 / 98	10 / 98	10 / 98	10 / 98	10 / 98
depot-20-7615	14 / 153	14 / 153	14 / 153	14 / 153	23 / 170	14 / 153	23 / 170	23 / 170	29 / 193
log-20-0	9 / 176	10 / 151	11 / 199	12 / 163	20 / 249	11 / 199	12 / 163	20 / 249	20 / 249
log-28-0	9 / 243	10 / 298	11 / 288	13 / 309	15 / 340	13 / 309	13 / 309	13 / 309	24 / 443

Table 3.17: Numbers of time points and operators in plans found by Algorithms A and B. Column $n = 1$ is Algorithm S. Dash indicates missing data when some formulae were not evaluated in 60 minutes.

Table 3.17 shows the numbers of time points and operators in the plans obtained for some of the benchmark problems reported in Table 3.16. In many cases the easiest satisfiable formulae are not the first ones, and these formulae typically have satisfying assignments that correspond to plans having many useless operators, which for algorithms A and B can lead to plans with many more operators than for algorithm S. However, the benchmarks have a simple structure and these plans with more operators are usually not genuinely different: the additional operators are either irrelevant for reaching the goals or contain pairs of operators and their inverses. It would be easy to eliminate these types of useless operators by a simple postprocessing step.

The Movie, MPrime and Mystery benchmarks from the 1998 competition and Rovers from 2002 are very easy for every evaluation strategy (fraction of a second in most cases) but we cannot produce the biggest MPrime instance because of a memory restriction.

The Logistics (1998 and 2000) and Satellite (2002) series are solved completely. Proving inexistence of plans slightly shorter than the optimal plan length is in some cases difficult but the new evaluation algorithms handle this efficiently.

The Depots (2002) problems are also relatively easy but in contrast to the rest of the benchmarks the new evaluation algorithms in some cases increase the runtimes up to the theoretical worst case.

The DriverLog and ZenoTravel (2002) problems are solved quickly except for some of the biggest instances. We cannot find satisfiable formulae for the last ZenoTravel problem within our time limit[7], and finding plans for the preceding two instances of ZenoTravel and the last two of DriverLog is also slow. The difficulty lies in finding tight lower bounds for plan lengths by determining the unsatisfiability of formulae.

Blocks World (2000) problems lead to very big formulae (size over 100 MB and over 524288 propositions), and we can solve only two thirds of the series.

Elevator (2000), Schedule (2000) and Gripper (1998) are a challenge because only very loose lower bounds on plan length are easy to prove. Finding plans corresponding to a given satisfiable formula is very easy (some seconds at most) but locating these formulae is very expensive. Increasing parameters n and γ improves runtimes.

The formulae generated for FreeCell (2002) are too big (hundreds of megabytes) for the current SAT solvers to solve them efficiently. This benchmark series along with the blocks world problems are the only ones that are not solved almost entirely.

[7]The number of propositions in formulae for plan lengths much higher than the presumed shortest plan length exceeds Siege's upper bound 524288.

All in all, it seems that a conservative use of the new algorithms (especially Algorithm B with $\gamma \in [0.7..0.9]$) leads to a general improvement in the runtimes in comparison to Algorithm S.

Decrease in plan quality is indirectly related to decrease in runtime. This depends on whether the first satisfiable formulae are the easiest ones. In general, satisfying valuations that are found for plan lengths much higher than the shortest plan length correspond to plans with more operators, but not always.

3.10 Literature

Progression and regression were used early in planning research [Rosenschein, 1981]. Our definition of regression in Section 3.1.2 is related to the weakest precondition predicates for program synthesis [de Bakker and de Roever, 1972; Dijkstra, 1976]. Instead of using the general definition of regression we presented, earlier work on planning with regression and a definition of operators that includes disjunctive preconditions and conditional effects has avoided all disjunctivity by producing only goal formulae that are conjunctions of literals [Anderson *et al.*, 1998]. Essentially, these formulae are the disjuncts of $regr_o(\phi)$ in DNF, although the formulae $regr_o(\phi)$ are not generated. The search algorithm then produces a search tree with one branch for every disjunct of the DNF formula. In comparison to the general definition, this approach often leads to a much higher branching factor and an exponentially bigger search tree.

The use of algorithms for the satisfiability problem of the classical propositional logic in planning was pioneered by Kautz and Selman, originally as a way of testing satisfiability algorithms, and later shown to be more efficient than other planning algorithms at the time [Kautz and Selman, 1992; 1996]. In addition to Kautz and Selman [1996], parallel plans were used by Blum and Furst in their Graphplan planner [Blum and Furst, 1997]. Parallelism in this context serves the same purpose as partial-order reduction [Godefroid, 1991; Valmari, 1991], reducing the number of orderings of independent actions to consider. Ernst et al. [1997] have considered translations of planning into the propositional that utilize the regular structure of sets of operators obtained from schematic operators. Planning by satisfiability has been extended to model-checking for testing whether a finite or infinite execution satisfying a given Linear Temporal Logic (LTL) formula exists [Biere *et al.*, 1999]. This approach to model-checking is called *bounded model-checking*.

It is trickier to use a satisfiability algorithm for showing that no plans of any length exist than for finding a plan of a given length. To show that no plans exist all plan lengths up to $2^n - 1$ have to be considered when there are n state variables. In typical planning applications n is often some hundreds or thousands, and generating and testing the satisfiability of all the required formulae is practically impossible. That no plans of a given length $n < 2^{|A|}$ do not exist does not directly imply anything about the existence of longer plans. Some other approaches for solving this problem based on satisfiability algorithms have been recently proposed [McMillan, 2003; Mneimneh and Sakallah, 2003].

The use of general-purpose heuristic search algorithms has recently got a lot of attention. The class of heuristics currently in the focus of interest was first proposed by McDermott [1999] and Bonet and Geffner [2001]. The distance estimates $\delta_I^{max}(\phi)$ and $\delta_I^{+}(\phi)$ in Section 3.4 are based on the ones proposed by Bonet and Geffner [2001]. Many other distance estimates similar to Bonet and Geffner's exist [Haslum and Geffner, 2000; Hoffmann and Nebel, 2001; Nguyen *et al.*, 2002]. The $\delta_I^{rlx}(\phi)$ estimate generalizes ideas proposed by Hoffmann and Nebel [2001].

Other techniques for speeding up planning with heuristic state-space search include symmetry

reduction [Starke, 1991; Emerson and Sistla, 1996] and partial-order reduction [Godefroid, 1991; Valmari, 1991; Alur *et al.*, 1997], both originally introduced outside planning in the context of reachability analysis and model-checking in computer-aided verification. Both of these techniques address the main problem in heuristic state-space search, high branching factor (number of applicable operators) and high number of states.

The algorithm for invariant computation was originally presented for simple operators without conditional effects [Rintanen, 1998]. The computation parallels the construction of planning graphs in the Graphplan algorithm [Blum and Furst, 1997], and it would seem to us that the notion of planning graph emerged when Blum and Furst noticed that the intermediate stages of invariant computation are useful for backward search algorithms: if a depth-bound of n is imposed on the search tree, then formulae obtained by m regression steps (suffixes of possible plans of length m) that do not satisfy clauses C_{n-m} cannot lead to a plan, and the search tree can be pruned. A different approach to find invariants has been proposed by Gerevini and Schubert [1998].

Some researchers extensively use Graphplan's planning graphs [Blum and Furst, 1997] for various purposes but we do not and have not discussed them in more detail for certain reasons. First, the graph character of planning graphs becomes inconvenient when preconditions of operators are arbitrary formulae and effects are conditional. As a result, the basic construction steps of planning graphs become unintuitive. Second, even when the operators have the simple form, the practically and theoretically important properties of planning graphs are not graph-theoretic. We can equivalently represent the contents of planning graphs as sequences of sets of literals and 2-literal clauses, as we have done in Section 3.5. In general it seems that the graph representation does not provide advantages over more conventional logic-based and set-based representations and is primarily useful for visualization purposes.

The algorithms presented in this section cannot in general be ordered in terms of efficiency. The general-purpose search algorithms with distance heuristics are often very effective in solving big problem instances with a sufficiently simple structure. This often entails better runtimes than in the SAT/CSP approach because of the high overheads with handling big formulae or constraint nets in the latter. Similarly, there are problems that are quickly solved by the SAT/CSP approach but on which heuristic state-space search fails.

There are few empirical studies on the behavior of different algorithms on planning problems in general or average. Bylander [1996] gives empirical results suggesting the existence of hard-easy pattern and a phase transition behavior similar to those found in other NP-hard problems like propositional satisfiability [Selman *et al.*, 1996]. Bylander also demonstrates that outside the phase transition region plans can be found by a simple hill-climbing algorithm or the inexistence of plans can be determined by using a simple syntactic test. Rintanen [2004d] complemented Bylander's work by analyzing the behavior of different types of planning algorithms on difficult problems inside the phase transition region, suggesting that current planners based on heuristic state space search are outperformed by satisfiability algorithms on difficult problems.

The PSPACE-completeness of the plan existence problem for deterministic planning is due to Bylander [1994]. The same result for another succinct representation of graphs had been established earlier by Lozano and Balcazar [1990].

Any computational problem that is NP-hard – not to mention PSPACE-hard – is considered too difficult to be solved in general. As planning even in the deterministic case is PSPACE-hard there has been interest in finding restricted special cases in which efficient (polynomial-time) planning is always guaranteed. Syntactic restrictions have been investigated by several researchers [Bylander, 1994; Bäckström and Nebel, 1995] but the restrictions are so strict that very few interesting

problems can be represented.

Schematic operators increase the conciseness of the representations of some problem instances exponentially and lift the worst-case complexity accordingly. For example, deterministic planning with schematic operators is EXPSPACE-complete [Erol *et al.*, 1995]. If function symbols are allowed, encoding arbitrary Turing machines becomes possible and the plan existence problem is undecidable [Erol *et al.*, 1995].

Chapter 4

Conditional planning: complexity

In this chapter we analyze the complexity of non-probabilistic planning with observation restrictions under two objectives, *reachability* and *maintenance* goals. Reachability goals are the most commonly considered objective a plan has to fulfill: reach any of a number of designated goal states. Plan executions in this case always have a finite (though possibly unbounded) length. Our interpretation of nondeterminism requires that the probability of every nondeterministic effect (which is not made explicit in our formalization of planning) is non-zero. Although plan executions may be arbitrarily long, the longer an execution is the smaller is its probability. Infinite execution that do not reach a goal state are possible but they have 0 probability. Maintenance objective is defined in terms of infinite plan executions. A plan satisfies the objective if all executions have an infinite length and only visit goal states.

Many of the results of this chapter are established by using a third objective which generalizes both reachability and maintenance and which we call *repeated reachability*. Under this objective plan executions are infinite and a plan has to repeatedly visit a goal state. Like for reachability goals, there is no upper bound on the number of execution steps between visits of a goal state but very long executions that do not visit a goal state have an extremely small probability.

We analyze the complexity of the plan existence problem under different observability restrictions and for the general nondeterministic actions and two subclasses of deterministic actions. For the more general deterministic class of actions the values of all state variables in the successor state are a function of the action and the current state. In the more restricted class of state-independent deterministic actions actions the set of state variables that are assigned a value by an action is only a function of the action, not of the current state. These two classes correspond to natural syntactic restrictions of operators describing the actions. Planning with these two classes of deterministic actions is in many cases substantially easier than in the general nondeterministic case.

In Section 4.2 we establish a number of complexity lower bounds in the form of hardness results for complexity classes PSPACE, EXP and 2-EXP of planning under the reachability objective. In the end of the section we explain briefly how the proofs of these lower bounds can be easily adapted to maintenance to obtain the same lower bound for it.

In Section 4.3 we establish corresponding complexity upper bounds in the form of resource-bounded algorithms under the repeated reachability, reachability and maintenance objectives. These algorithms in some cases are of only theoretical interest and in Chapter 5 we introduce some more practical approaches to solve these planning problems.

The lower and upper bound coincide and as a results we have proofs of PSPACE, EXP and 2-EXP-completeness of the planning problems.

4.1 Preliminaries

4.1.1 Alternative observation models

In Definition 2.8 we have defined observability in terms of a subset of the state variables that are observable. More generally, observations could be arbitrary formulae instead of state variables, and it could be possible to observe the values of these formulae without being able to observe the values of their subformulae. Also, what is observable could be determined by the last operator that has been applied. This is usually how *sensing actions* are formalized. To make certain observation an action that enables the observation has to be taken.

In this section we show that the observation model we have adopted is as powerful as models with non-atomic observations and sensing actions. The reduction from the more general model requires state-dependent effects.

In the more general model, a succinct transition system $\langle A, I, O, G, V\rangle$ does not have a fixed set of observable state variables. Instead, V map each $o \in O$ to a set of formulae over A: after applying an operator $o \in O$ the truth-values of the formulae $V(o)$ can be observed. It can also be that the value of a formula ϕ can be observed but not the values of its subformulae.

Let $\Pi = \langle A, I, O, G, V\rangle$ be a succinct transition system under this more general definition. We give a reduction to the basic definition. Let $\beta_1, \ldots, \beta_n$ be the formulae $\beta \in V(o)$ for some $o \in O$. We introduce new auxiliary state variables $a_{\beta_1}, \ldots, a_{\beta_n}$, z, and z_i for every $i \in \{1, \ldots, m\}$ where m is the number of operators in O. State variable a_β will have the same value as the respective observation β right after applying an operator that allowed observing it. The state variables z_i control the application of operators that evaluate the values of observable formulae. Define the succinct transition system $C(\Pi) = \langle A, I \wedge z \wedge \neg z_1 \wedge \cdots \wedge \neg z_m, O', G, \{a_{\beta_1}, \ldots, a_{\beta_n}\}\rangle$ where O' consists of

$$\langle z \wedge c, \neg z \wedge z_i \wedge e\rangle$$
$$\langle z_i, z \wedge \neg z_i \wedge \nu\rangle$$

for every operator $\langle c, e\rangle \in O$ (operator's index is i). The first operator replaces the old operator and the effect ν in the second operator evaluates the observations β and copies their values to the respective state variables a_β.

$$\nu \quad = \quad \bigwedge\{(\beta \rhd a_\beta) \wedge (\neg\beta \rhd \neg a_\beta) | \beta \in V(o)\}$$

The state variable z indicates that any operator can be applied, and z_i indicates that operator i has been applied and the corresponding observations are to be evaluated.

Theorem 4.1 *Let Π be a succinct transition system. Then Π has a plan if and only if $C(\Pi)$ has.*

Proof: We only give a brief proof sketch.

The reduction guarantees that the value of a_β coincides with the value of β after an operator with observation β has been applied.

Translations from plans for Π into plans for $C(\Pi)$ and vice versa do not require changing the structure of the plans, only one operator is interchanged with a sequence of two operators, or vice versa.

Plans for Π can be translated into plans for $C(\Pi)$ by replacing observations of β by observations of a_β and replacing every operator with the corresponding two new operators.

Plans for $C(\Pi)$ can be transformed into plans for Π by replacing observations a_β by β and making the converse replacement of operators. The two operators in $C(\Pi)$ obtained from one operator

in Π always appear together, and any branch between these two operators can be moved before the first operator because the first operator does not make any observable information available. $\square$

4.1.2 Plans

Plans determine which actions are executed. We formalize plans as a form of directed graphs. Each node is assigned an operator and information on zero or more successor nodes.

Definition 4.2 *Let $\Pi = \langle A, I, O, G, V\rangle$ be a succinct transition system. A* plan *for Π is a triple $\langle N, b, l\rangle$ where*

1. *N is a finite set of nodes,*
2. *$b \subseteq \mathcal{L} \times N$ maps initial states to starting nodes, and*
3. *$l : N \to O \times 2^{\mathcal{L}\times N}$ is a function that assigns each node an operator and a set of pairs $\langle \phi, n'\rangle$ where ϕ is a formula over the observable state variables V and $n' \in N$ is a node.*

   Nodes n with $l(n) = \langle o, \emptyset\rangle$ for some $o \in O$ are terminal nodes.

By ignoring the operators and branch formulae in a plan π we can construct a graph $G(\pi) = \langle N, E\rangle$ with $E \subseteq N \times N$ such that $\langle n, n'\rangle \in E$ iff $\langle \phi, n'\rangle \in B$ for $l(n) = \langle o, B\rangle$ and some ϕ. A plan π is *acyclic* if there is no non-trivial path starting and ending at the same node in $G(\pi)$.

Plan execution starts from a node $n \in N$ and state s such that $\langle \phi, n\rangle \in b$ and $s \models I \wedge \phi$. If there is no such $\langle \phi, n\rangle \in b$, then plan execution ends immediately. Execution in node n with $l(n) = \langle o, B\rangle$ proceeds by executing the operator o and then testing for each $\langle \phi, n'\rangle \in l(n)$ whether ϕ is true in all possible current states, and if it is, continuing execution from plan node n'. At most one ϕ may be true for this to be well-defined. Plan execution ends when none of the branch labels matches the current state. In a terminal node plan execution necessarily ends.

We define the satisfaction of plan objectives in terms of the transition system that is obtained when the original transition system is being controlled by a plan.

Definition 4.3 (Execution graph of a plan) *Let $\langle A, I, O, G, V\rangle$ be a succinct transition system and $\pi = \langle N, b, l\rangle$ be a plan. Define* the execution graph *of π as a pair $\langle M, E\rangle$ where*

1. *$M = S \times (N \cup \{\bot, \square\})$, where S is the set of Boolean valuations of A,*
2. *$E \subseteq M \times M$ has an edge from $\langle s, n\rangle \in S \times N$ to $\langle s', n'\rangle \in S \times N$ if and only if $l(n) = \langle o, B\rangle$ and for some $\langle \phi, n'\rangle \in B$*

   (a) $s' \in img_o(s)$ and

   (b) $s' \models \phi$,

   and an edge from $\langle s, n\rangle \in S \times N$ to $\langle s', \bot\rangle$ if and only if

   (a) $l(n) = \langle o, B\rangle$,

   (b) $s' \in img_o(s)$, and

   (c) there is no $\langle \phi, n'\rangle \in B$ such that $s' \models \phi$,

and an edge from $\langle s, n\rangle \in S \times N$ *to* $\langle s, \Box\rangle$ *if and only if*

(a) $l(n) = \langle o, B\rangle$, *and*

(b) $img_o(s) = \emptyset$.

We need the node $\bot$ to make all terminating plan executions explicit in the execution graph. All nodes in the execution graph with $\bot$ are terminal nodes. Without $\bot$ there would be no simple graph-theoretic definition of terminating executions. Consider a node n with label $l(n) = \langle o, \{\langle \phi, n'\rangle\}\rangle$ so that $img_o(s) = \{s_1, s_2\}$ and $s_1 \models \phi$ and $s_2 \not\models \phi$. The execution graph has an edge from $\langle n, s\rangle$ to $\langle s_1, n'\rangle$ and $\langle s_2, \bot\rangle$. If we had no edge to $\langle s_2, \bot\rangle$ it would seem that all executions from $\langle s, n\rangle$ would continue to $\langle s_1, n'\rangle$.

The node $\Box$ is needed for handling plans that attempt to apply an operator that is not applicable.

4.1.3 Decision problems

There are different types of objectives the plans may have to fulfill. The most basic one which is widely used in AI planning research is the reachability of a goal state. In this case each plan execution has a finite length. Also problems with infinite plan executions are meaningful. A plan does not reach a goal and terminate, but is a continuing process that has to repeatedly reach goal states or avoid visiting bad states. Examples of these are different kinds of maintenance tasks: keep a building clean and transport mail from location A to location B.

We define two objectives in terms of infinite plan executions. One is maintenance goals, the other is repeated reachability. Infinite plan executions are also used in connection with Markov decision processes and the average-reward and discounted geometric reward objectives [Puterman, 1994].

The plan objectives we use in this work are the following.

1. RG reachability (finite horizon)

   The objective is to reach one of predefined goal states starting from the initial state. Length of plan executions in this case is finite but for a given plan the length may still be unbounded if some of the actions are nondeterministic and loops are allowed.

2. MG Maintenance goals (infinite horizon)

   The objective is to take actions to stay indefinitely within the set of goal states: a plan has to maintain a certain property. Plan executions in this case are infinite.

3. RRG repeated reachability (infinite horizon)

   The objective is to repeatedly reach one of predefined goal states, possibly intermittently visiting non-goal states. After a goal state has been reached, plan execution is continued and a goal state has to be reached again.

In Section 4.1.4 we show that reachability goals and maintenance goals are special cases of repeated reachability.

Note that repeated reachability is not a trivial extension of reachability goals because in the first case plan execution can end only after reaching a belief state consisting of goal states only whereas in the second case it might never be known whether the current state is a goal state. We illustrate this by an example.

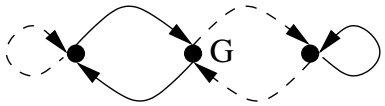


Figure 4.1: Reason why plans cannot be defined as mappings from belief states to actions

Example 4.4 Consider the transition graph in Figure 4.1. The belief state initially and after taking any action consists of all three states. To satisfy the repeated reachability criterion the two actions, one depicted with a dotted line and the other with a continuous line, have to be alternated. In this example Repeated Reachability objective is satisfied although at no point of time is it known that a goal state is occupied. ■

The example shows how the dependencies between consecutive belief states become important in the repeated reachability problem. The belief state which is the set of possible current states does not alone carry enough information for deciding whether the objective is fulfilled or not. As a consequence plans cannot be defined as mappings from belief states (understood as sets of states) to actions, unlike for maintenance or reachability goals.

Definition 4.5 (Reachability goals RG) *A plan $\pi = \langle N, b, l \rangle$ solves a succinct transition system $\langle A, I, O, G, V \rangle$ under the* Reachability *(RG) criterion if the corresponding execution graph fulfills the following.*

1. *For every state s such that $s \models I$ either $s \models G$ or there is $\langle \phi, n \rangle \in b$ such that $s \models \phi$.*
2. *For all states s and $\langle \phi, n \rangle \in b$ such that $s \models I \wedge \phi$, for every (s', n') to which there is a path from (s, n) there is a path from (s', n') of length ≥ 0 to some (s'', n'') such that $s'' \models G$, (s'', n'') has no successor nodes and $n'' \neq \Box$.*

This plan objective with unbounded looping can be interpreted probabilistically. For every nondeterministic choice in an operator we have to assume that each of the alternatives has a non-zero probability. For goal reachability, a plan with unbounded looping is simply a plan that has no finite upper bound on the length of its executions, but that with probability 1 eventually reaches a goal state. A non-looping plan also reaches a goal state with probability 1 but there is a finite upper bound on the execution length.

Definition 4.6 (Maintenance goals MG) *A plan $\pi = \langle N, b, l \rangle$ solves a succinct transition system $\langle A, I, O, G, V \rangle$ under the* Maintenance *(MG) criterion if the corresponding execution graph fulfills the following.*

1. *For every state s such that $s \models I$ there is $\langle \phi, n \rangle \in b$ such that $s \models \phi$.*
2. *For every state s and $\langle \phi, n \rangle \in b$ such that $s \models I \wedge \phi$, if there is a path of length ≥ 0 from (s, n) to some (s', n'), then $s' \models G$ and (s', n') has a successor node.*

Definition 4.7 (Repeated reachability goals RRG) *A plan $\pi = \langle N, b, l \rangle$ solves a succinct transition system $\langle A, I, O, G, V \rangle$ under the* Repeated Reachability *(RRG) criterion if the corresponding execution graph fulfills the following.*

1. *For every state s such that $s \models I$ there is $\langle \phi, n \rangle \in b$ such that $s \models \phi$.*

2. *For every state s and $\langle \phi, n \rangle \in b$ such that $s \models I \wedge \phi$, if there is a path from (s, n) to some (s', n'), then there is a path of length ≥ 1 from (s', n') to some (s'', n'') such that $s'' \models G$.*

The decision problem addressed in this work is whether a certain transition system has a plan that satisfies the goal objective.

We primarily consider the two objectives reachability goals (RG) and maintenance goals (MG). In some cases give a result for both simultaneously in terms of the repeated reachability goals (RRG) objective. Each problem is analyzed under full observability (FO), without observability (UO) and in the general case of partial observability (PO). In addition to arbitrary nondeterministic operators (ND), we also analyze problems restricted to the subclass of deterministic operators (D), and the subclass of deterministic state-independent operators (ID).

We assign names like PLANSAT-PO-ND-RRG to the decision problems. This problem is the most general problem we consider, with partial observability, nondeterministic operators and the repeated reachability criterion.

Some of our results are about the existence of acyclic plans, as defined in Section 4.1.2. Execution graphs of acyclic plans are acyclic. In these cases the name of the decision problem has the form PLANSAT-PO-ND-acyclicRG. Note that for unobservable problems with reachability goals we can always restrict to acyclic plans.

4.1.4 Reductions between objectives

Reachability goals and maintenance goals are easily reducible to repeated reachability. These reductions are useful for proving lower bounds on plan existence problems for repeated reachability, and for proving upper bounds on plan existence problems for reachability and maintenance goals.

Theorem 4.8 *Let $\Pi = \langle A, I, O, G, V \rangle$ be a succinct transition system. Then Π has a solution for RG if and only if Π' has a solution for RRG, where $\Pi' = \langle A \cup \{a\}, I \wedge \neg a, O \cup \{\langle G, a \rangle, \langle a, \top \rangle\}, a, V \rangle$ and $a \notin A$.*

Proof: Sketch: Assume a plan for Π under RG exists. Any terminal node n can be extended to first apply $\langle G, a \rangle$ and then repeat $\langle a, \top \rangle$ indefinitely. This results in a plan that eventually reaches a state that satisfies G, then reaches $G \wedge a$, and then stays in that goal state indefinitely.

Assume a plan for Π' under RRG exists. For the RRG criterion to be satisfied a state satisfying a is visited on every execution, and therefore the plan has one or more occurrences of the operator $\langle G, a \rangle$. In the plan we delete all children of nodes that apply $\langle G, a \rangle$ and make those nodes terminal nodes. Since every execution of Π until a state that satisfies G is a prefix of an execution of Π', and every execution of Π' eventually reaches a state that satisfies a, every execution of Π eventually reaches a state that satisfies G. □

Theorem 4.9 *Let $\Pi = \langle A, I, O, G, V \rangle$ be a succinct transition system. Then Π has a solution for MG if and only if Π' has a solution for RRG, where $\Pi' = \langle A, I, \{\langle c \wedge G, e \rangle | \langle c, e \rangle \in O\}, G, V \rangle$.*

Proof: Assume there is a plan that satisfies the maintenance objective for Π. The plan that is obtained by replacing every operator $\langle c, e \rangle$ by $\langle c \wedge G, e \rangle$ satisfies the repeated reachability objective for Π': all operators of the latter plan are always applicable because every state reached during execution satisfies G.

Assume there is a plan that satisfies the repeated reachability objective for Π'. Since G is true always when applying an operator, G is true in all states reached during plan execution. Hence the plan obtained by replacing every $\langle c \wedge G, e\rangle$ by $\langle c, e\rangle$ satisfies the maintenance objective. □

Note that in both of these proofs all the operators in Π are deterministic (deterministic state-independent) if and only if all those in Π' are deterministic (deterministic state-independent). This allows us to use – under the three classes of operators ND, D and ID– algorithms and complexity upper bound proofs for repeated reachability also for reachability and maintenance, and complexity lower bound proofs for reachability and maintenance also for repeated reachability.

4.2 Lower bounds of complexity

First we discuss the complexity of plan synthesis for reachability goals. These results also yield lower bounds for the repeated reachability objective, and the hardness results for maintenance goals are in most cases simple modifications of the Turing machine simulations presented in this section.

For deterministic planning with only one initial state observability is not relevant because the state after executing any sequence of operators can always be unambiguously predicted. This problem is closely related to the s-t-reachability problem of succinctly represented graphs [Papadimitriou and Yannakakis, 1986; Lozano and Balcázar, 1990] which is PSPACE-hard, too.

Theorem 4.10 ([Bylander, 1994]) *Planning with one initial state and deterministic operators, even for state-independent operators (PLANSAT-??-ID-RG), is PSPACE-hard.*

The proof of this theorem is a simulation of polynomial-space deterministic Turing machines. It can be obtained from the proof of Theorem 4.11 by restricting it to deterministic Turing machines. It is easy to generalize the simulation to nondeterministic NPSPACE=PSPACE Turing machines.

4.2.1 Planning with full observability

When nondeterministic (with or without probabilities) operators are allowed, the complexity of plan existence problems increases. Littman [1997] reduces the EXP-complete existence problem of winning strategies in the game G_4 [Stockmeyer and Chandra, 1979] to the plan existence problem of probabilistic planning. Probabilities do not play a role in G_4 or in the reduction, and hence also the plan existence problem of non-probabilistic planning with fully observability is EXP-hard.

Next we present a proof for the EXP-hardness of conditional planning with full observability by a direct simulation of polynomial-space alternating Turing machines. The result follows from the equality EXP=APSPACE. The proof of Theorem 4.10 by Bylander shows how PSPACE Turing machines are simulated. For APSPACE we also need to simulate alternation, which means that configurations of the TM may have several successor configurations, and that there are both $\forall$ and $\exists$ states.

For configurations with $\forall$ states all successor configurations must be accepting (final or non-final) configurations. For configurations with $\exists$ states at least one successor configuration must be an accepting (final or non-final) configuration. Both of these requirements can be represented in the nondeterministic planning problem.

The transitions from a configuration with a $\forall$ state will correspond to one nondeterministic operator. That all successor configurations must be accepting (final or non-final) configurations corresponds to the requirement in planning that from all successor states of a state a goal state must be reached.

Every transition from a configuration with $\exists$ state will correspond to a deterministic operator. Since only one of the operators is chosen to the plan it means that only one of the successor configurations needs to be accepting.

Theorem 4.11 *The problem of testing the existence of an acyclic plan for succinct transition systems with full observability (PLANSAT-FO-ND-acyclicRG) is EXP-hard, even with the restriction to only one initial state.*

Proof: Let $\langle \Sigma, Q, \delta, q_0, g \rangle$ be any alternating Turing machine with a polynomial space bound $p(x)$. Let σ be an input string of length n. We construct a succinct transition system $\langle A, I, O, G, A \rangle$ with full observability for simulating the Turing machine. The succinct transition system has a size that is polynomial in the size of the description of the Turing machine and the input string.

The set A of state variables in the succinct transition system consists of

1. $q \in Q$ for denoting the states of the ATM,
2. s_i for every symbol $s \in \Sigma \cup \{|, \Box\}$ and tape cell $i \in \{0, \ldots, p(n)\}$, and
3. h_i for the positions of the R/W head $i \in \{0, \ldots, p(n)\}$.

The subscripts are the indices of the tape cells with which the state variables are associated.

The succinct transition system has exactly one initial state which represents the initial configuration of the ATM. It is described by the formula I which is the conjunction of the following literals.

1. q_0
2. $\neg q$ for all $q \in Q \backslash \{q_0\}$.
3. s_i for all $s \in \Sigma$ and $i \in \{1, \ldots, n\}$ such that ith input symbol is s.
4. $\neg s_i$ for all $s \in \Sigma$ and $i \in \{1, \ldots, n\}$ such that ith input symbol is not s.
5. $\neg s_i$ for all $s \in \Sigma$ and $i \in \{0, n+1, n+2, \ldots, p(n)\}$.
6. $\Box_i$ for all $i \in \{n+1, \ldots, p(n)\}$.
7. $\neg \Box_i$ for all $i \in \{0, \ldots, n\}$.
8. $|_0$
9. $\neg |_i$ for all $n \in \{1, \ldots, p(n)\}$
10. h_1
11. $\neg h_i$ for all $i \in \{0, 2, 3, 4, \ldots, p(n)\}$

The goal is the following formula.

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}$$

Next we define the set O of operators corresponding to transitions of the ATM. We have to distinguish between transitions for universal $\forall$ and existential $\exists$ states.[1] For a given input symbol and a $\forall$ state, the transitions from this state correspond to one nondeterministic operator, whereas for a given input symbol and an $\exists$ state each transition corresponds to a different deterministic operator.

We first define the effects of the operators. For all $\langle s, q\rangle \in (\Sigma \cup \{|, \square\}) \times Q$, $i \in \{0, \ldots, p(n)\}$ and $\langle s', q', m\rangle \in (\Sigma \cup \{|\}) \times Q \times \{L, N, R\}$ define $\tau_{s,q,i}(s', q', m) = \alpha \wedge \kappa \wedge \theta$ where the effects α, κ and θ are defined below.

The effect α describes the changes to the tape symbol at the R/W head. If $s = s'$ then $\alpha = \top$ as nothing on the tape changes. Otherwise $\alpha = \neg s_i \wedge s'_i$ to denote that the new symbol in the ith tape cell is s' and not s.

The effect κ describes the change to the state of the ATM. We define $\kappa = \neg q$ when $i = p(n)$ and $m = R$ so that when the space bound is violated no accepting state is reached. Otherwise if $i < p(n)$ or $m \neq R$, define $\kappa = \top$ if $q = q'$ and $\kappa = \neg q \wedge q'$ if $q \neq q'$.

The effect θ describes the movement of the R/W head. Either the movement is to the left, there is no movement, or the movement is to the right. By definition of ATMs movement at the left end of the tape is always to the right.

$$\theta \quad = \quad \begin{cases} \neg h_i \wedge h_{i-1} & \text{if } m = L \\ \top & \text{if } m = N \\ \neg h_i \wedge h_{i+1} & \text{if } m = R \text{ and } i \leq p(n) \\ \top & \text{if } m = R \text{ and } i = p(n) \end{cases}$$

Next we define the operators separately for existential and universal ATM states. Let $\langle s, q\rangle \in (\Sigma \cup \{|, \square\}) \times Q$, $i \in \{0, \ldots, p(n)\}$ and $\delta(s, q) = \{\langle s^1, q^1, m^1\rangle, \ldots, \langle s^k, q^k, m^k\rangle\}$.

If $g(q) = \exists$, then define k deterministic operators

$$\begin{aligned} o_{s,q,i,1} &= \langle h_i \wedge s_i \wedge q, \tau_{s,q,i}(s^1, q^1, m^1)\rangle \\ o_{s,q,i,2} &= \langle h_i \wedge s_i \wedge q, \tau_{s,q,i}(s^2, q^2, m^2)\rangle \\ &\vdots \\ o_{s,q,i,k} &= \langle h_i \wedge s_i \wedge q, \tau_{s,q,i}(s^k, q^k, m^k)\rangle. \end{aligned}$$

For a given current symbol s, state q and tape cell i a plan determines which $\exists$ transition is chosen by applying the corresponding operator.

If $g(q) = \forall$, then define one nondeterministic operator

$$\begin{aligned} o_{s,q,i} = \langle h_i \wedge s_i \wedge q, \quad & (\tau_{s,q,i}(s^1, q^1, m^1)| \\ & \tau_{s,q,i}(s^2, q^2, m^2)| \\ & \vdots \\ & \tau_{s,q,i}(s^k, q^k, m^k))\rangle. \end{aligned}$$

Since the transition is chosen nondeterministically a plan has to reach the goals for all transitions from a $\forall$ configuration.

[1] There are no transitions for accepting and rejecting states.

For establishing a connection between the accepting computation trees of an ATM and valid plans define for an ATM configuration $c = \langle q, \sigma, \sigma' \rangle$ the state $z = CS(c)$ as follows.

1. $z(q) = 1$
2. $z(q') = 0$ for all $q' \in Q \backslash \{q\}$
3. $z(s_i) = 1$ iff $(\sigma\sigma')[i] = s$, for all $s \in \Sigma$ and $i \in \{1, \ldots, |\sigma\sigma'|\}$
4. $z(s_i) = 0$ for all $s \in \Sigma$ and $i \in \{|\sigma\sigma'|, \ldots, p(n)\}$
5. $z(|_0) = 1$
6. $z(|_i) = 0$ for all $i \in \{1, \ldots, p(n)\}$
7. $z(\square_i) = 0$ for all $i \in \{0, \ldots, |\sigma\sigma'| - 1\}$
8. $z(\square_i) = 1$ for all $i \in \{|\sigma\sigma'|, \ldots, p(n)\}$
9. $z(h_i) = 1$ iff $i = |\sigma|$, for all $i \in \{0, \ldots, p(n)\}$

Claim A: For configurations $\langle q, \sigma_1, \sigma_2 \rangle$ and $\langle q', \sigma_1', \sigma_2' \rangle$ such that $q, q' \in Q$ and $g(q) = \exists$, $\langle q, \sigma_1, \sigma_2 \rangle \vdash \langle q', \sigma_1', \sigma_2' \rangle$ if and only if $\mathit{img}_o(CS(\langle q, \sigma_1, \sigma_2 \rangle)) = \{CS(\langle q', \sigma_1', \sigma_2' \rangle)\}$ for some $o \in O$.

Claim B: For a configuration $\langle q, \sigma_1, \sigma_2 \rangle$ such that $g(q) = \forall$,

$$\mathit{img}_o(CS(\langle q, \sigma_1, \sigma_2 \rangle)) = \{CS(\langle q', \sigma_1', \sigma_2' \rangle) | \langle q, \sigma_1, \sigma_2 \rangle \vdash \langle q', \sigma_1', \sigma_2' \rangle\}$$

for some $o \in O$. Operators in $O \backslash \{o\}$ are not applicable in $CS(\langle q, \sigma_1, \sigma_2 \rangle)$.

We prove that $\langle A, I, O, G, A \rangle$ has a plan if and only if the alternating Turing machine $\langle \Sigma, Q, \delta, q_0, g \rangle$ with input string σ accepts without violating the space bound.

If the Turing machine violates the space bound then all state variables $q \in Q$ will be false and no further operator will be applicable. Hence no goal state can be reached.

Otherwise, we can inductively extract from a computation tree of an accepting ATM a plan that always reaches a goal state, and vice versa.

If the initial configuration $\langle q_0, |a, \sigma' \rangle$ is a 0-accepting configuration then the plan is $\langle \emptyset, \emptyset, \emptyset \rangle$. In this case the starting node of the execution graph is $\langle CS(\langle q_0, |a, \sigma' \rangle), \bot \rangle$ which is a goal node without successors.

We show how accepting computation trees are mapped to plans with execution graphs satisfying the RG criterion so that all paths in the graph have a finite length. We take the d-accepting configurations of the ATM for $d \geq 1$ to be the nodes in the plan. The construction proceeds inductively.

Base case $d = 1$: Let $c = \langle q, \sigma_1, \sigma_2 \rangle$ be an 1-accepting $\exists$-configuration with current symbol s, R/W-head position i, and 0-accepting final configuration c' reached by transition j. We define c as a plan node with $l(c) = \langle o_{s,q,i,j}, \emptyset \rangle$.

In the execution graph there are two nodes $\langle CS(c), c \rangle$ and $\langle CS(c'), \bot \rangle$ with an edge between them. Since c' is an accepting configuration $CS(c')$ is a goal state. Hence from the node $\langle CS(c), c \rangle$ there is a path of length 1 to a terminal node and this is the only maximal path starting in that node.

Let $c = \langle q, \sigma_1, \sigma_2 \rangle$ be an 1-accepting $\forall$-configuration with current symbol s, R/W-head position i, and only 0-accepting successor configurations $c_1, \ldots, c_m$. We define c as a plan node with $l(c) = \langle o_{s,q,i}, \emptyset \rangle$.

In the execution graph there is the node $\langle CS(c), c \rangle$ with edges to nodes $\langle CS(c_i), \bot \rangle$ where $CS(c_i)$ is a goal state for every $i \in \{1, \ldots, m\}$. Hence from the node $\langle CS(c), c \rangle$ all maximal paths have a length of 1 and end in a terminal node.

Inductive case $d \geq 2$:

1. Let c be a d-accepting $\exists$-configuration with current symbol s, current state q and R/W head position i, and let c' be a $d-1$-accepting successor configuration reached by transition $\langle s', q', m \rangle$.

   The configuration c is mapped to a plan node c with $l(c) = \langle o_{s,q,i,j}, \{\langle \top, c' \rangle\} \rangle$ where s is the current symbol, q is the current state, and j is the index of the transition $\langle s', q', m \rangle$.

   By Claim A in the execution graph there is an edge from $\langle CS(c), c \rangle$ to $\langle CS(c'), c' \rangle$.

   By the induction assumption all paths in the execution graph from $\langle CS(c'), c' \rangle$ have a length of at most $d-1$ and end in a node that corresponds to a goal state. Hence all paths starting from $\langle CS(c), c \rangle$ have length at most d and end in a goal node.

2. Let c be a d-accepting $\forall$-configuration with successor configurations $c_1, \ldots, c_k$, R/W head position i, current symbol s, and state q. Let $\langle s^1, q_1, m_1 \rangle, \ldots, \langle s^k, q_k, m_k \rangle$ be the transitions from c that respectively reach $c_1, \ldots, c_k$. For the plan node c define $l(c) = \langle o_{s,q,i}, \{\langle s_i^1 \wedge q_1 \wedge l_1, c_1 \rangle, \ldots, \langle s_i^k \wedge q_k \wedge l_k, c_k \rangle\} \rangle$ where for every $j \in \{1, \ldots, k\}$, $l_j = h_{i+1}$ if $m_j = R$, $l_j = h_{i-1}$ if $m_j = L$ and otherwise $l_j = h_i$.

   By Claim B $img_{o_{s,q,i}}(CS(c)) = \{CS(c_1), \ldots, CS(c_k)\}$. These states differ with respect to state variables representing the symbol that was written, the state of the TM, and the location of the R/W head. Branching in $l(c)$ distinguishes between all these states. Hence in the execution graph the edges from $\langle CS(c), c \rangle$ are to nodes $\langle CS(c_1), c_1 \rangle, \ldots, \langle CS(c_k), c_k \rangle$.

   By the induction assumption all paths from the latter nodes have a length of at most $d-1$ and end in a node that corresponds to a goal state, and at least one of the paths has length $d-1$. Hence all paths starting from $\langle CS(c), c \rangle$ have a length of at most d and end in a goal node.

The plan is $\langle N, b, l \rangle$ where N is the set of d-accepting configurations for $d \geq 1$ and $b = \{\langle \top, \langle q_0, |a, \sigma' \rangle \rangle\}$ where $\sigma = a\sigma'$ is the input string. All paths in the execution graph that start from the unique starting node $\langle CS(\langle q_0, |a, \sigma' \rangle), \langle q_0, |a, \sigma' \rangle \rangle$ are finite and end in a node that corresponds to a goal state.

Then we show how acyclic plans are mapped to accepting computation trees of the ATM. The proof is extended to cyclic plans in Section 4.2.4. The acyclicity of the plans entails the acyclicity of the execution graphs.

Given an execution graph that satisfies the reachability criterion we construct an accepting computation tree for the Turing machine. First we give a mapping from states to configurations. This mapping is defined only for states v that correspond to configurations in the sense that $v(q) = 1$ for exactly one $q \in Q$ and of the state variables representing the tape contents and the location of the R/W head exactly one is true for any tape location. Any state reachable from the unique initial state of the planning problem satisfies these requirements. Let $SC(v) = \langle q, \sigma, \sigma' \rangle$ where

1. q is the state in Q such that $v(q) = 1$,
2. k is the number of symbols that are left from the R/W head so that $v(h_k) = 1$,
3. σ is the symbol sequence of length k such that symbol $i \in \{0, \ldots, k\}$ is $s \in \Sigma$ iff $v(s_i) = 1$,
4. $\sigma' = \epsilon$ if $k = e(n)$ or $v(\square_k) = 1$, and otherwise $\sigma' = s'^1, \ldots, s'^m$ where m is such that $v(\square_{k+m}) = 0$ and $v(\square_{k+m+1}) = 1$ and $v(s'^i_{k+i}) = 1$ for all $i \in \{1, \ldots, m\}$ and some $s'^i \in \Sigma$.

Let s_I be the unique state such that $s_I \models I$. Let $\langle N, b, l\rangle$ be an acyclic plan for the succinct transition system. Let R be the set of nodes of the execution graph to which there is a path from the initial node $\langle s_I, n\rangle$. We will show that $T = \{SC(s)|\langle s, n\rangle \in R\}$ is an accepting computation tree of the ATM.

Induction hypothesis: Let $\langle s, n\rangle \in R$ be a node in the execution graph so that there is path from the initial node to $\langle s, n\rangle$ and the longest path to a terminal node has length d. Then $SC(s)$ is a d'-accepting configuration for some $d' \leq d$.

Base case $d = 0$: Let $\langle s, n\rangle \in R$ be a node in the execution graph with no successor nodes. Let $c = \langle q, \sigma, \sigma'\rangle = SC(s)$. As $\langle s, n\rangle$ has no successors must s be a goal state. Hence q is an accepting state and c is a 0-accepting final configuration.

Inductive case $d \geq 1$: Let $\langle s, n\rangle \in R$ be a node in the execution graph for which the longest path to a terminal node has length d. The configuration $c = \langle q, \sigma, \sigma'\rangle = SC(s)$ belongs to T. If c is an $\exists$-configuration, then by Claim A $\langle s, n\rangle$ has exactly one successor node $\langle s', n'\rangle$ and the longest path from $\langle s', n'\rangle$ to a terminal node has length $d - 1$. By the induction hypothesis $SC(\langle s', n'\rangle)$ is a $d' - 1$-accepting configuration of the Turing machine for somd $d' \leq d$. Hence c is a d'-accepting configuration for some $d' \leq d$.

If c is a $\forall$-configuration, then by Claim B $\langle s, n\rangle$ has successor nodes $\langle s_1, n_1\rangle, \ldots, \langle s_k, n_k\rangle$ from which the longest path to a terminal node has length $\leq d - 1$. By the induction hypothesis $SC(\langle s_1, n_1\rangle), \ldots, SC(\langle s_k, n_k\rangle)$ are $d' - 1$-accepting configurations for some $d' \leq d$. Hence c is an d'-accepting configuration for som $d' \leq d$.

Since T consists of accepting configurations only and it includes the initial configuration, the ATM accepts.

As all alternating Turing machines with a polynomial space bound can be translated into a nondeterministic planning problem in polynomial time, all decision problems in APSPACE are polynomial time many-one reducible to nondeterministic planning. Hence the plan existence problem is APSPACE-hard and EXP-hard. □

4.2.2 Planning without observability

The plan existence problem of conditional planning with unobservability is more complex than that of conditional planning with full observability. We give a new EXPSPACE-hardness proof for a result that was first proved by Haslum and Jonsson [2000] and later extend it to a 2-EXP-hardness proof for the more general partially observable case.

We show the EXPSPACE-hardness by a direct simulation of exponential space Turing machines. The first problem is the representation of the exponentially long tape. In the PSPACE- and APSPACE-hardness proofs of deterministic planning and conditional planning with full observability, it is possible to represent the polynomial number of tape cells as state variables of

the planning problem. With an exponential space bound an exponential number of state variables would be needed, and such a reduction would not be possible in polynomial time.

Hence we have to find a more clever way of encoding the tape contents. It turns out that we can use the uncertainty about the initial state for this purpose. When an execution of the plan that simulates the Turing machine is started, we randomly choose one of the tape cells to be the *watched* tape cell. This is the only cell of the tape for which the contents are represented in the state variables. Of all changes to the working tape that the simulated TM makes, only the changes of the watched tape cell are reflected in the state variables.

For guaranteeing that the plan corresponds to a simulation of the Turing machine, it is tested whether the transition the plan makes when the current tape cell is the watched tape cell is the one that assumes the current symbol to be the one that is stored in the state variables. If it is not, the plan is not a valid plan. Since the watched tape cell could be any of the exponential number of tape cells, all the transitions the plan makes are guaranteed to correspond to the contents of the current tape cell of the Turing machine. So if the plan does not simulate the Turing machine, the plan is not guaranteed to reach the goal states.

The proof uses several initial states and unobservability. Several initial states are needed for selecting the watched tape cell, and unobservability is needed so that the plan cannot cheat: if the plan can determine what the current tape cell is, it could choose transitions that correspond to the Turing machine transitions only on the watched tape cell. Alternatively, we can have only one initial state and select the watched tape cell by one nondeterministic operator in the beginning.

Theorem 4.12 *The problems of testing the existence of a plan for succinct transition systems with unobservability and deterministic operators (PLANSAT-UO-D-RG) and with nondeterministic operators and one initial state (PLANSAT-UO-ND-RG-1) are EXPSPACE-hard.*

Proof: We give a simulation of deterministic Turing machines with an exponential space bound. Nondeterministic Turing machines could be simulated for a NEXPSPACE-hardness proof, but this additional generality is not interesting because EXPSPACE = NEXPSPACE.

Below we give the proof for multiple initial states and deterministic operators, but we could instead use one initial state and force the plan to first apply a special operator that nondeterministically generates the required successor states, so the theorem holds under either assumption.

Let $\langle \Sigma, Q, \delta, q_0, g \rangle$ be any deterministic Turing machine with an exponential space bound $e(x)$. Let σ be an input string of length n. We denote the ith symbol of σ by σ_i.

The Turing machine may use space $e(n)$, and for encoding numbers from 0 to $e(n)$ corresponding to the tape cells we need $n^* = \lceil \log_2(e(n) + 1) \rceil$ Boolean state variables.

We construct a succinct transition system without observability for simulating the Turing machine. It has a size that is polynomial in the size of the description of the Turing machine and the input string.

Unlike in the proof of Theorem 4.11 we cannot represent the contents of the tape by having a state variable for every tape cell because an exponential number of state variables would be needed and the reduction would take exponential time.

It turns out that it suffices to keep track of only one tape cell (that we call the *watched tape cell*) that is randomly chosen in the beginning of every execution of the plan.

The set A of state variables in the succinct transition system consists of

1. $q \in Q$ for denoting the states of the TM,
2. w_i for $i \in \{0, \ldots, n^* - 1\}$ for the index of the watched tape cell,

3. s for every symbol $s \in \Sigma \cup \{|, \square\}$ for the contents of the watched tape cell,
4. h_i for $i \in \{0, \ldots, n^* - 1\}$ for the position of the R/W head.

The formula I which represents the initial states encodes the initial configuration of the TM requires that the variables w_i represent the index of one of the tape cells $0, \ldots, e(n)$. The formula I is the conjunction of the following formulae.

1. q_0
2. $\neg q$ for all $q \in Q \backslash \{q_0\}$.
3. $w < e(n) + 1$
4. Initialization of the state variables for the contents of the watched tape cell.

$$\begin{array}{rcl} | & \leftrightarrow & (w = 0) \\ \square & \leftrightarrow & (w > n) \\ s & \leftrightarrow & \bigvee_{i \in \{1,\ldots,n\}, \sigma_i = s} (w = i) \text{ for all } s \in \Sigma \end{array}$$

5. $h = 1$ for the initial position of the R/W head.

The initial state formula allows any valuation of the state variables w_i. A given input string and a valuation of variables w_i the formula determines the values of the state variables $s \in \Sigma$ uniquely. The expressions $w = i$, $w < i$, $w > i$ denote formulae that test integer equality and inequality of the numbers encoded by $w_0, \ldots, w_{n^*-1}$. Later we will also use effects $h := h + 1$ and $h := h - 1$ which represent incrementing and decrementing the number encoded by $h_0, \ldots, h_{n^*-1}$.

The goal formula is

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}.$$

To define the operators we first define effects corresponding to transitions.

For all $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\langle s', q', m \rangle \in (\Sigma \cup \{|\}) \times Q \times \{L, N, R\}$ define the effect $\tau_{s,q}(s', q', m)$ as $\alpha \wedge \kappa \wedge \theta$ where α, κ and θ are defined below.

The effect α describes what happens to the tape symbol under the R/W head. If $s = s'$ then $\alpha = \top$ as nothing on the tape changes. Otherwise $\alpha = ((h = w) \rhd (\neg s \wedge s'))$ so that if the current tape cell is the watched one then the change is recorded in the state variables for the watched tape cell contents.

The effect κ describes the change to the state of the TM. If the R/W movement is to the left or if there is no movement we define $\kappa = \neg q \wedge q'$ if $q \neq q'$ and $\kappa = \top$ otherwise. If the R/W head movement is to the right we define $\kappa = \neg q \wedge ((h < e(n)) \rhd q')$ if $q \neq q'$ and $\kappa = (h = e(n)) \rhd \neg q$ if $q = q'$. Hence if the space bound is violated no operator will be applicable and no accepting state can be reached.

The movement of the R/W head is described by θ.

$$\theta = \begin{cases} h := h - 1 & \text{if } m = L \\ \top & \text{if } m = N \\ h := h + 1 & \text{if } m = R \end{cases}$$

The operators are defined next. Let $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\delta(s, q) = \{\langle s', q', m \rangle\}$. If $g(q) \neq \exists$ there is no operator because the computation from accepting and rejecting states does not proceed further. If $g(q) = \exists$ then define

$$o_{s,q} = \langle ((h \neq w) \vee s) \wedge q, \tau_{s,q}(s', q', m) \rangle.$$

The precondition $(h \neq w) \vee s$ is critical for the correct simulation of the TM. If the current tape cell is the watched tape cell then the operator application is guaranteed to correspond to the TM transition.

We show that for a given accepting computation tree T of the Turing machine there is a corresponding plan and an execution graph that satisfies the RG criterion. First we define a mapping from TM configurations $c = \langle q, \sigma, \sigma' \rangle$ and indices $i \in \{0, \ldots, e(n)\}$ of watched tape cells to states of the transition system as $z = CS_i(c)$ where

1. $z(q) = 1$
2. $z(q') = 0$ for all $q' \in Q \backslash \{q\}$
3. $z(s) = 1$ iff $i \in \{1, \ldots, |\sigma\sigma'|\}$ and $(\sigma\sigma')[i] = s$, for all $s \in \Sigma$
4. $z(|) = 1$ iff $i = 0$
5. $z(\Box) = 0$ iff $i \in \{0, \ldots, |\sigma\sigma'|\}$
6. $z(w_j) = 1$ iff jth bit of i is 1, for all $j \in \{0, \ldots, m-1\}$
7. $z(h_j) = 1$ iff jth bit of $|\sigma|$ is 1, for all $j \in \{0, \ldots, n^* - 1\}$.

Let d be an integer such that the initial configuration c_I is d-accepting. If $d = 0$ (the initial configuration is an accepting final configuration) then define the plan $\pi = \langle \emptyset, \emptyset, \emptyset \rangle$. In the execution graph of this plan the starting nodes $\langle CS_i(c_I), \bot \rangle, i \in \{1, \ldots, e(n)\}$ are also terminal nodes and there are no other nodes reachable from the starting nodes. Hence the RG criterion is satisfied.

If $d \geq 1$ define $\pi = \langle N, b, l \rangle$ where $b = \{(\top, c_I)\}$. The plan nodes $N = N_1 \cup \cdots \cup N_d$ and their labels $l = l_1 \cup \cdots \cup l_d$ are defined below.

For every $i \in \{0, \ldots, d\}$ let N_i be the set of i-accepting configurations in T.

For every $c \in N_1$ define $l_1(c) = \langle o_{s,q}, \emptyset \rangle$ where s is the current symbol and q is the current state of c. The 1-accepting configurations are terminal nodes in the plan and after the operators in these nodes have been applied plan execution terminates.

For every $i \in \{2, \ldots, d\}$ and $c \in N_i$ define $l_i(c) = \langle o_{s,q}, \{(\top, c')\} \rangle$ where c' is the successor configuration of c in T and s is the current symbol and q is the current state of c. Hence the plan nodes that correspond to an i-accepting configuration for $i \geq 1$ apply the operator that corresponds to the transition in that configuration and continue from a plan node that corresponds to the unique successor configuration.

The execution graph corresponding to the plan is $X = \langle V, E \rangle$ where

$$
\begin{array}{rcl}
V_0 & = & \{\langle CS_j(c), \bot \rangle | 0 \leq j \leq e(n), c \in N_0\} \\
V_i & = & \{\langle CS_j(c), c \rangle | 0 \leq j \leq e(n), c \in N_i\} \text{ for all } i \in \{1, \ldots, d\} \\
E_1 & = & \{(\langle s, c \rangle, \langle s', \bot \rangle) \in V_1 \times V_0 | o \in O, s' \in \mathit{img}_o(s)\} \\
E_i & = & \{(\langle CS_j(c), c \rangle, \langle CS_j(c'), c' \rangle) \in V_i \times V_{i-1} | 0 \leq j \leq e(n), \{c, c'\} \subseteq T, c \vdash c'\} \\
 & & \text{for all } i \in \{2, \ldots, d\} \\
V & = & V_0 \cup \cdots \cup V_d \\
E & = & E_1 \cup \cdots \cup E_d.
\end{array}
$$

The execution graph consists of $e(n) + 1$ disjoint chains of length d. The nodes in V_d are the first nodes in the chains. They are the initial nodes in the execution graph. The nodes in V_0 are the last nodes in the chains. They are the terminal nodes of the execution graph.

A simple induction proof shows that there is a path from every initial node in the execution graph to a terminal node that corresponds to a goal state of the planning problem. Hence the RG criterion is satisfied by the plan.

Assume a plan for the succinct transition system exists. Since there are no observations this plan always executes the same sequence $o_1, \ldots, o_k$ of operators. We construct a sequence $c_0, \ldots, c_k$ of configurations that form an accepting computation tree of the Turing machine.

Let $c_0 = \langle q_0, |a, \sigma' \rangle$ where $\sigma = a\sigma'$ is the input string, and $c_i = NC_{o_i}(c_{i-1})$ for all $i \in \{1, \ldots, k\}$ where

$$NC_{o_{s,q}}(\langle q, \sigma a, \sigma' \rangle) = \begin{cases} \langle q', \sigma, a'\sigma' \rangle & \text{if } \delta(s,q) = \langle a', q', L \rangle \\ \langle q', \sigma a', \sigma' \rangle & \text{if } \delta(s,q) = \langle a', q', N \rangle \\ \langle q', \sigma a' b, \sigma'' \rangle & \text{if } \delta(s,q) = \langle a', q', R \rangle \text{ and } \sigma' = b\sigma'' \\ \langle q', \sigma a' \Box, \epsilon \rangle & \text{if } \delta(s,q) = \langle a', q', R \rangle \text{ and } \sigma' = \epsilon \end{cases}$$

It is easy to verify that the operator sequence corresponds to the transition sequence of the TM, assuming that when operator $o_{s,q}$ is executed the current tape symbol is indeed s. Assume that some $o_{s,q}$ is the first operator that misrepresents the tape contents and that $h = c$ for some tape cell location c. There is an execution of the plan so that $w = c$. But in this case the precondition of $o_{s,q}$ would not be satisfied, and the plan is not executable. Hence a valid plan cannot contain operators that misrepresent the tape contents. □

4.2.3 Planning with partial observability

We show that the plan existence problem of the general conditional planning problem with partial observability is 2-EXP-hard. The hardness proof is by a simulation of AEXPSPACE=2-EXP Turing machines.

The hardness proof is an extension of both the EXP-hardness proof of Theorem 4.11 and of the EXPSPACE-hardness proof of Theorem 4.12. From the first proof we have the simulation of alternation, and from the second proof the simulation of an exponentially long tape.

Theorem 4.13 *The problem of testing the existence of an acyclic plan for succinct transition systems with partial observability (PLANSAT-PO-ND-acyclicRG) is 2-EXP-hard. This holds even when there is only one initial state.*

Proof: Let $\langle \Sigma, Q, \delta, q_0, g \rangle$ be any alternating Turing machine with an exponential space bound $e(x)$. Let σ be an input string of length n. We denote the ith symbol of σ by σ_i.

The Turing machine may use space $e(n)$, and for encoding numbers from 0 to $e(n)$ corresponding to the tape cells we need $n^* = \lceil \log_2(e(n) + 1) \rceil$ Boolean state variables.

We construct a succinct transition system with full observability for simulating the Turing machine. The succinct transition system has a size that is polynomial in the size of the description of the Turing machine and the input string.

We use watched tape cells as in the proof of Theorem 4.12.

The set A of state variables in the succinct transition system consists of

1. $q \in Q$ for denoting the states of the TM,
2. w_i for $i \in \{0, \ldots, n^* - 1\}$ for the index of the watched tape cell,

3. s for every symbol $s \in \Sigma \cup \{|, \square\}$ for the contents of the watched tape cell,
4. s^* for every $s \in \Sigma \cup \{|\}$ for the symbol last written (important for nondeterministic transitions),
5. L, R and N for the last movement of the R/W head (important for nondeterministic transitions), and
6. h_i for $i \in \{0, \ldots, n^* - 1\}$ for the position of the R/W head.

The observable state variables are L, N and R, $q \in Q$, and s^* for $s \in \Sigma$. These are needed by the plan to decide how to proceed execution after a nondeterministic transition with a $\forall$ state.

Define the initial state formula I as the conjunction of the following formulae.

1. q_0
2. $\neg q$ for all $q \in Q \backslash \{q_0\}$.
3. $\neg s^*$ for all $s \in \Sigma \cup \{|\}$.
4. $w < e(n) + 1$
5. Formulae for having the contents of the watched tape cell in state variables $\Sigma \cup \{|, \square\}$.

$$\begin{array}{rcl} | & \leftrightarrow & (w = 0) \\ \square & \leftrightarrow & (w > n) \\ s & \leftrightarrow & \bigvee_{i \in \{1,\ldots,n\}, \sigma_i = s} (w = i) \text{ for all } s \in \Sigma \end{array}$$

6. $h = 1$ for the initial position of the R/W head.

Instead of encoding the choice of watched tape cells in the initial state formula, we could alternatively use only one initial state and a nondeterministic operator that is only applicable in the initial state and that would randomly select the watched tape cell. Hence the ATM simulation is also possible with succinct transition systems with only one initial state.

The goal is the following formula.

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}$$

To define the operators, we first define effects corresponding to all possible transitions.

For all $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\langle s', q', m \rangle \in (\Sigma \cup \{|\}) \times Q \times \{L, N, R\}$ define the effect $\tau_{s,q}(s', q', m)$ as $\alpha \wedge \kappa \wedge \theta$ where the effects α, κ and θ are defined as follows.

The effect α describes how the current tape symbol changes. If $s = s'$ then $\alpha = \top$ as nothing on the tape changes. Otherwise $\alpha = ((h = w) \rhd (\neg s \wedge s')) \wedge s'^* \wedge \neg s^*$ to denote that the new symbol in the watched tape cell is s' and not s, and to make it possible for the plan to detect which symbol was written onto the tape.

The effect κ describes the change to the state of the TM. If R/W head moves to the left or does not move define $\kappa = \neg q \wedge q'$ if $q \neq q'$ and $\kappa = \top$ if $q = q'$. If R/W head moves to the right $\kappa = \neg q \wedge ((h < e(n)) \rhd q')$ if $q \neq q'$ and $\kappa = (h = e(n)) \rhd \neg q$ if $q = q'$. Hence if the space bound is violated no operator will be applicable and no accepting state can be reached.

The movement of the R/W head is described by the effect

$$\theta \quad = \quad \begin{cases} (h := h-1) \wedge L \wedge \neg N \wedge \neg R & \text{if } m = L \\ N \wedge \neg L \wedge \neg R & \text{if } m = N \\ (h := h+1) \wedge R \wedge \neg L \wedge \neg N & \text{if } m = R \end{cases}$$

The observable state variables L, N and R are for the plan to make branching decisions.

Let $\langle s, q\rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\delta(s,q) = \{\langle s_1, q_1, m_1\rangle, \ldots, \langle s_k, q_k, m_k\rangle\}$. If $g(q) = \exists$, then define k deterministic operators

$$\begin{aligned} o_{s,q,1} &= \langle((h \neq w) \vee s) \wedge q, \tau_{s,q}(s_1, q_1, m_1)\rangle \\ o_{s,q,2} &= \langle((h \neq w) \vee s) \wedge q, \tau_{s,q}(s_2, q_2, m_2)\rangle \\ &\vdots \\ o_{s,q,k} &= \langle((h \neq w) \vee s) \wedge q, \tau_{s,q}(s_k, q_k, m_k)\rangle \end{aligned}$$

Hence, the plan determines which transition is chosen. If $g(q) = \forall$, then define one nondeterministic operator

$$\begin{aligned} o_{s,q} = \langle((h \neq w) \vee s) \wedge q, \quad & (\tau_{s,q}(s_1, q_1, m_1)| \\ & \tau_{s,q}(s_2, q_2, m_2)| \\ & \vdots \\ & \tau_{s,q}(s_k, q_k, m_k))\rangle. \end{aligned}$$

This operator nondeterministically chooses which transition takes place.

We show that the succinct transition system has an acyclic plan if and only if the Turing machine accepts without violating the space bound.

If the Turing machine violates the space bound then all state variables $q \in Q$ will be false and no further operator will be applicable. Hence no goal state can be reached.

Otherwise, it can be inductively shown that a computation tree of an accepting ATM can be mapped to a conditional plan that always reaches a goal state, and vice versa. These mappings are directly based on the mappings given in the proofs of Theorems 4.11 and 4.12 and we do not give them in detail.

A computation tree can be mapped to a plan of the same form.

1. A 1-accepting configuration is mapped to a terminal plan node.
2. An $\exists$ configuration is mapped to a node that applies the deterministic operator that corresponds to the transition in that configuration.
3. A $\forall$ configuration is mapped to a node that applies the nondeterministic operator that corresponds to the transitions that are possible and then uses the observations to branch and reach successor nodes corresponding to all the possible successor configurations.

The mapping from plans to computation trees is similar to the mapping in Theorem 4.11 except that the states in plan executions cannot be directly mapped to ATM configurations as they do not represent the tape contents explicitly. As in the proof of Theorem 4.12 the contents of the tape must be recovered from the transition sequence that leads to the configuration from the initial configuration.

Since all alternating Turing machines with an exponential space bound can be translated into a nondeterministic planning problem with partial observability in polynomial time, all decision

problems in AEXPSPACE are polynomial time many-one reducible to nondeterministic planning, and the plan existence problem is AEXPSPACE-hard and consequently 2-EXP-hard. □

Vardi and Stockmeyer [1985] established the 2-EXP-hardness of the satisfiability of the branching time temporal logic CTL$*$ by a simulation of exponential-time alternating Turing machines, but their proof idea is different from ours. Their proof encodes ATM configurations as sequences of states, each corresponding to one tape cell, and tests the correctness of the differences between two consecutive tape cells by CTL$*$ formulae. Our proof encodes the ATM configurations in the belief state – implicitly represented by a plan – by utilizing partial observability.

When operators are restricted to be deterministic or state-independent, complexity of the plan existence problem in the most general cases decreases.

Our proofs for the EXP-hardness and 2-EXP-hardness of plan existence for planning with full and partial observability use nondeterminism. Also the EXP-hardness proof of Littman [1997] uses nondeterminism. The question arises whether complexity is lower when all operators are deterministic. This turns out to be the case for planning with full (Theorem 4.16) and partial observability (Theorem 4.24), but without observability there is no difference: the EXPSPACE-hardness proof in this case uses deterministic operators only. In the partially observable case determinism guarantees that the size of the current belief state can never increase, and this leads to reduced complexity.

Interestingly, deterministic planning with one initial state is exactly as complex both with and without conditional effects, but for planning with restrictions on observability there is a difference in computational complexity. With deterministic state-independent operators the uncertainty about actual current state monotonically decreases, and the values of state variables never change without the new value being known, which leads to a belief space with a particularly simple structure.

4.2.4 Plans with loops

The complexity results of the above plan existence problems also hold when loops are allowed in the plans. Loops are needed for finitely representing repetitive strategies that do not have an upper bound on execution length. A typical problem would be to toss a die until its value is six. Assuming that the probability of the die falling on every side is non-zero, eventually getting six has probability 1, although the zero-probability event of unsuccessfully throwing the die infinitely many times is still possible.

In this section we discuss how the earlier results are extended to the case with loops in the plans. Looping is not possible in the unobservable case as it is impossible to decide when to stop looping, but for the fully and partially observable cases looping is applicable.

The problem that looping plans cause for the proofs of Theorems 4.11 and 4.13 is that a Turing machine computation of infinite length is not accepting but the corresponding infinite length zero-probability plan execution is allowed to be a part of a plan. Hence for acyclic plans there is this mismatch between plans and accepting computation trees.

To eliminate infinite plan executions we have to modify the Turing machine simulations given in the proofs of those theorems. The modification involves counting the length of the plan executions and rejecting when the plan has been executed so far that at least one configuration has been visited more than once. This modification makes looping plans ineffective, and in the absence of loops the simulation is faithful.

Theorem 4.14 *The problem of testing the existence of an acyclic plan for succinct transition systems with full observability (PLANSAT-FO-ND-RG) is EXP-hard, even with the restriction to only one initial state.*

Proof: This is an easy modification of the proof of Theorem 4.11. If there are n state variables, an acyclic plan exists if and only if a plan with execution length of at most 2^n exists because visiting any state more than once is unnecessary. Plans that rely on loops can be invalidated by counting the number of actions taken and failing when it exceeds 2^n. This counting can be done by having $n+1$ auxiliary state variables $c_0, \ldots, c_n$ which are initialized to false. Every operator $\langle p, e \rangle$ is extended to $\langle p, e \wedge t \rangle$ where t is an effect that increments the binary number encoded by $c_0, \ldots, c_n$ by one until the most significant bit c_n becomes one. The goal G is replaced by $G \wedge \neg c_n$.

Therefore, a plan exists if and only if an acyclic plan exists if and only if the alternating Turing machine accepts. □

In the fully observable case counting the execution length does not pose a problem because we only have to count an exponential number of execution steps. This can be represented by a polynomial number of state variables. In the partially observable case, however, we need to count a doubly exponential number of execution steps, as the number of visited belief states may be doubly exponential. A binary representation of these numbers requires an exponential number of bits. We cannot use an exponential number of state variables for representing them because the reduction to planning would not be polynomial time. However, partial observability with only a polynomial number of auxiliary state variables has the power to count doubly exponentially far.

Theorem 4.15 *The problem of testing the existence of a plan for succinct transition systems with partial observability (PLANSAT-PO-ND-RG) is 2-EXP-hard. This holds even when there is only one initial state.*

Proof: We extend the proof of Theorem 4.13 by a counting scheme that makes cyclic plans ineffective. We show how counting transitions can be achieved within a succinct transition system obtained from the alternating Turing machine and the input string in polynomial time.

Since representing the number of transitions as a binary number requires an exponential number of state variables, we cannot represent the number explicitly. Instead, we use a randomized technique to force the plan to implicitly count the number of Turing machine transitions so far. The technique has some resemblance to the idea we used in the simulation of exponentially long working tapes in proofs of Theorems 4.11 and 4.13 but now the idea is applied to incrementing a number that has exponentially many digits.

For a succinct transition system with n state variables (that represent the Turing machine configurations) executions that visit each belief state at most once may have a length of 2^{2^n}. Representing these numbers requires 2^n binary digits. We introduce $n+1$ new unobservable state variables $d_0, \ldots, d_n$ for representing the index of *one of the digits* and v_d for the value of this digit, and new state variables $c_0, \ldots, c_n$ through which the plan indicates changes to the count of Turing machine transitions. There is a set of operators by means of which the plan sets the values of these variables before applying an operator corresponding to a Turing machine transition.

The idea of the construction is the following. Whenever the count of TM transitions is incremented, one of the 2^n digits in the count changes from 0 to 1, and all of the less significant digits change from 1 to 0. The plan is forced to communicate the index of the digit that changes from 0 to 1 by the state variables $c_0, \ldots, c_n$. The unobservable state variables $d_0, \ldots, d_n, v_d$ store the

index and the value of one of the digits. We call this digit *the watched digit*. One watched digit is sufficient for verifying that the reporting of $c_0, \ldots, c_n$ by the plan is truthful because any of the digits could be watched one and false reporting of its value is detected on one execution of the plan, making the plan invalid. Plan execution fails when reporting is false or when the count exceeds 2^{2^n}. For this reason a plan for the succinct transition system exists if and only if an acyclic plan exists if and only if the Turing machine accepts the input string.

Next we define how the succinct transition systems defined in the proof of Theorem 4.13 are extended to prevent unbounded looping in plans by means of a counter.

The initial state description is extended with the conjunct $\neg d_v$ in order to signify that the watched digit is initially 0 (all the digits in the counter implicitly represented in the belief state are 0.) The state variables $d_0, \ldots, d_n$ may have any values, which means that the watched digit is chosen randomly. The state variables $d_v, d_0, \ldots, d_n$ are all unobservable so that the plan cannot know which digit is watched.

There is a state variable f that is initially set to false by having $\neg f$ in the initial states formula. It becomes true on plan executions that do not correspond to legal TM computations.

The goal G is extended to $G \wedge \neg f \wedge ((d_0 \wedge \cdots \wedge d_n) \rightarrow \neg d_v)$ to prevent executions that lead to setting f true or that have a length that exceeds $2^{2^{n+1}-1}$. The conjunct $(d_0 \wedge \cdots \wedge d_n) \rightarrow \neg d_v$ is false if the index of the watched digit is $2^{n+1} - 1$ and the digit is true, corresponding to an execution length $\geq 2^{2^{n+1}-1}$.

The new operators for indicating the changing digit are

$$\begin{array}{ll} \langle \top, c_i \rangle & \text{for all } i \in \{0, \ldots, n\} \text{ and} \\ \langle \top, \neg c_i \rangle & \text{for all } i \in \{0, \ldots, n\}. \end{array}$$

The operators for Turing machine transitions are extended with the randomized test that tests whether the digit the plan claims to change from 0 to 1 is the one that actually changes: every operator $\langle p, e \rangle$ defined in the proof of Theorem 4.13 is replaced by $\langle p, e \wedge t \rangle$ where t is the conjunction of the following effects.

$$\begin{array}{rcl} ((c = d) \wedge d_v) & \rhd & f \\ (c = d) & \rhd & d_v \\ ((c > d) \wedge \neg d_v) & \rhd & f \\ (c > d) & \rhd & \neg d_v \end{array}$$

Here $c = d$ denotes $(c_0 \leftrightarrow d_0) \wedge \cdots \wedge (c_n \leftrightarrow d_n)$, and $c > d$ encodes the greater-than test for the binary numbers encoded by $c_0, \ldots, c_n$ and $d_0, \ldots, d_n$.

The above effects do the following.

1. When the plan claims that the watched digit changes from 0 to 1 and the value of d_v is 1, fail.

2. When the plan claims that the watched digit changes from 0 to 1, change d_v to 1.

3. When the plan claims that a more significant digit changes from 0 to 1 and the value of d_v is 0, fail.

4. When the plan claims that a more significant digit changes from 0 to 1, set the value of d_v to 0.

The fact that these effects guarantee the invalidity of a plan that relies on unbounded looping is because the failure flag f will be set if the plan lies about the count, or the most significant bit with index $2^{n+1} - 1$ will be set if the count reaches $2^{2^{n+1}-1}$. Attempts of unfair counting are recognized, and consequently f is set to true because of the following.

Assume that the binary digit at index i changes from 0 to 1 (and therefore all less significant digits change from 1 to 0) and the plan incorrectly claims that it is the digit j that changes, and this is the first time on this execution that the plan lies (hence the value of d_v is the true value of the watched digit.)

If $j > i$, then i could be the watched digit (and hence $c > d$), and for j to change from 0 to 1 the less significant bit i should be 1, but we would know that it is not because d_v is false. Consequently on this plan execution the failure flag f would be set.

If $j < i$, then j could be the watched digit (and hence $c = d$), and the value of d_v would indicate that the current value of digit j is 1, not 0. Consequently on this plan execution the failure flag f would be set.

So, if the plan does not correctly report the digit that changes from 0 to 1, then the plan is not valid. Hence any valid plan correctly counts the execution length which cannot exceed $2^{2^{n+1}-1}$. □

The lower bounds we have established for the reachability objective also hold for the maintenance objective. The proofs are obtained from the proofs of Theorems 4.11, 4.12 and 4.13 by using a counter for the number of actions taken: before the counter reaches 2^n (for full observability) or 2^{2^n} (for partial observability) any state is a goal state, and after reaching the limit only the original goal states are goal states. Indefinitely staying in a goal state is made possible by a dummy operator that has no effects. This way the maintenance objective is satisfied if and only if the Turing machine accepts.

4.3 Upper bounds of complexity

In this section we establish complexity upper bounds for the goal reachability RG and maintenance MG criteria, in some cases by giving an algorithm for the most general repeated reachability criterion RRG. The complexities of the criteria are the same in all cases we consider.

4.3.1 Planning with full observability

In the simplest deterministic fully observable cases the plan existence problem for repeated reachability can be solved by identifying paths and cycles in the transition graph.

Theorem 4.16 *Testing plan existence for succinct transition systems with deterministic operators and only one initial state (PLANSAT-??-D-RRG-1) and for deterministic fully observable succinct transition systems with several initial states (PLANSAT-FO-D-RRG) under the repeated reachability criterion is in PSPACE.*

Proof: Let $\Pi = \langle A, I, O, G, A \rangle$ be a succinct transition system so that all operators in O are deterministic.

The fully observable problem with several initial states easily reduces to the one initial state case: iterate over all initial states and separately test for each whether a plan exists. Full observability is essential for this reduction to be correct because the plans for different initial states could

```
1:  procedure FOreach(A,O,s,s',m)
2:  if m ≤ 0 then
3:    if (s = s' or there is o ∈ O such that {s'} = img_o(s)) then return true
4:    else return false
5:  else
6:    for each valuation s'' of A do
7:      if FOreach(A,O,s,s'',m − 1) and FOreach(A,O,s'',s',m − 1) then return true
8:    end
9:  return false
```

Figure 4.2: Algorithm for finding a path of length $\leq 2^m$ from s to s'

not otherwise be chosen by observations.

For the case with one initial state and deterministic operators we give an algorithm for testing the existence of plans. Let s_I be the unique initial state. The plan existence problem can be solved by in polynomial space as follows. Iterate over all states to find a state s such that

1. $s \models G$,
2. s is reachable from s_I by at most $2^{|A|}$ steps, tested by FOreach(A,O,s_I,s,$|A|$), and
3. there is a non-trivial path from s to itself. This is by testing FOreach(A,O,s',s,$|A|$) for every $o \in O$ with $img_o(s) = \{s'\}$. If one of these calls returns true, a non-trivial path from s to itself exists.

If such a state s exists then there is a sequence of actions that reaches s from s_I and another sequence of 1 or more actions that reaches s from itself. Executing the second action sequence indefinitely satisfies the RRG criterion.

The reachability tests are for paths of a length of at most $2^{|A|}$ because the number of states is $2^{|A|}$ and if a path exists then a path of length at most $2^{|A|} - 1$ exists.

Iteration over all states s can be done in polynomial space, and FOreach(A,O,s,s',m) needs polynomial space in m and the sizes of the inputs. The procedure FOreach is given in Figure 4.2. □

Planning with full observability has properties that allow using a simpler definition of plans. Essentially, plans with several nodes are unnecessary as no memory about the prior steps in the plan execution need to be retained. Hence plans can be viewed as mappings from states to actions, or equivalently, each state can be taken as a plan node, and after executing the associated action the plan continues from the node corresponding to the successor state. In connection with Markov decision processes [Howard, 1960; Puterman, 1994] this kind of history-independent policies have been called *stationary*.

Theorem 4.17 *Given a plan $\pi = \langle N, b, l\rangle$ for a succinct transition system $\Pi = \langle A, I, O, G, A\rangle$ under the RRG criterion, there is a function $x : S \to O$ and a plan $\pi' = \langle S, b', l'\rangle$ for Π such that S is the set of all valuations of A (all states) and*

1. *$b = \{\langle U(s), s\rangle | s \in S, s \models I\}$, and*
2. *$l(s) = \langle x(s), \{\langle U(s'), s'\rangle | s' \in img_{x(s)}(s)\}\rangle$ for every $s \in S$,*

where $U(s) = \bigwedge(\{a \in A | s \models a\} \cup \{\neg a | a \in A, s \models \neg a\})$*, and its execution graph is* $\langle M, E\rangle$ *where*

1. $M = S \times S$
2. $E = \{(\langle s, s\rangle, \langle s', s'\rangle) \in M \times M | o \in O, s' \in img_{x(s)}(s)\}$

and $\langle M, E\rangle$ *satisfies the RRG criterion.*

Proof: The construction is in two stages. First we construct a plan in which every plan node uniquely identifies the state but there are several plan nodes corresponding to a state. Full observability is needed for this construction to work. Then plan nodes for each state are combined so that there is a one-to-one match between plan nodes and states.

Define $\pi'' = \langle N'', b'', l''\rangle$ where

- $N'' = N \times S$,
- $b'' = \{\langle U(s), (n, s)\rangle | \langle \phi, n\rangle \in b, s \in S, s \models \phi \wedge I\}$, and
- $l''((n, s)) = \langle o, \{\langle U(s'), (n', s')\rangle | \langle \phi, n'\rangle \in B, s' \in S, s' \models \phi\}\rangle$ where $l(n) = \langle o, B\rangle$, for every $n \in N$ and $s \in S$.

In the execution graph of π'' all the initial nodes and all nodes reachable from them have the form $\langle s, (n, s)\rangle$ where $s \in S$ and $n \in N$. The execution graph of π'' satisfies the RRG criterion if the execution graph of π does because for $s, s' \in S$ and $n, n' \in N$ there is an edge from $\langle s, (n, s)\rangle$ to $\langle s', (n', s')\rangle$ if and only if there is an edge from $\langle s, n\rangle$ to $\langle s', n'\rangle$ in the execution graph of π.

Let $s_1, \ldots, s_m$ be an enumeration of S. Next we combine for every s_i all nodes $\langle s_i, (n, s_i)\rangle$ while preserving the RRG criterion. Define a sequence of plans $\pi_0 = \langle N_0, b_0, l_0\rangle, \ldots, \pi_m = \langle N_m, b_m, l_m\rangle$ start from $\langle N_0, b_0, l_0\rangle = \pi''$.

Assume we have constructed the plan π_{i-1} for some $i \in \{1, \ldots, m\}$. Assume that there is at least one $n \in N$ such that (n, s_i) is reachable from an initial node of the execution graph of π_{i-1}. Then there is at least one such n so that there is a path from (n, s_i) to a node (n', s') or s' such that $s' \models G$ that does not visit any node (n'', s_i). Let n^* be any such n. If no node (n, s_i) is reachable from an initial node then let $n*$ be any node in N. Let $F_i(k) = k$ for all $k \in N_{i-1} \backslash (N \times \{s_i\})$ and $F_i(k) = s_i$ for all $k \in N \times \{s_i\}$.

- $N_i = (N \times \{s_{i+1}, \ldots, s_m\}) \cup \{s_1, \ldots, s_i\}$,
- $b_i = \{\langle U(s), (n, s)\rangle \in b_{i-1} | \langle \phi, n\rangle \in b, s \models \phi, s \notin N_i\} \cup \{\langle U(s), s\rangle | s \in N_i, s \models I\}$,
- $l_i(k) = \langle o, \{\langle \phi, F_i(k')\rangle | \langle \phi, k'\rangle \in B\}\rangle$ where $l_{i-1}(k) = \langle o, B\rangle$, for every $k \in N_i \backslash \{s_i\}$,
- $l_i(s_i) = \langle o, \{\langle \phi, F_i(k')\rangle | \langle \phi, k'\rangle \in B\}\rangle$ where $l_{i-1}((n^*, s)) = \langle o, B\rangle$.

kesken: jospa $n*$ solmua ei olekaan?
kesken: argumentti miksi graafi edelleen on RRG

□

So in the fully observable case plans can be equivalently represented as mappings from states to actions and the execution graph has a particularly simple form because no distinction between

```
1:  procedure prune(S,O,G);
2:  W_{-1} := S;
3:  W_0 := ∅;
4:  repeat
5:    W'_0 := W_0;
6:    W_0 := (W'_0 ∪ ⋃_{o∈O} preimg_o(W'_0 ∪ G)) ∩ S;
7:  until W_0 = W'_0;      (* States from which a goal state can be reached by ≥ 1 operators *)
8:  i := 0;
9:  repeat
10:   i := i + 1;
11:   k := 0;
12:   S_0 := ∅;
13:   repeat
14:     k := k + 1;          (* States from which a state in G is reachable with ≤ k steps. *)
15:     S_k := S_{k-1} ∪ ⋃_{o∈O}(S ∩ preimg_o(S_{k-1} ∪ G) ∩ spreimg_o(W_{i-1} ∪ G));
16:   until S_k = S_{k-1};          (* States that stay within W_{i-1} before reaching G. *)
17:   W_i := S_k;
18: until W_i = W_{i-1};          (* States in W_i stay within W_i before reaching G. *)
19: return W_i;
```

Figure 4.3: Algorithm for detecting a loop that eventually makes progress

the current state and the current plan node needs to be made. To simplify the presentation of the complexity proofs for the most general fully observable plannin problem we use this representation of plans.

We give a new algorithm for solving the planning problem with nondeterministic operators, full observability and repeated reachability goals. This problem generalizes the problem solved by an algorithm given by Cimatti et al. [2003] (the global strong cyclic algorithm) for our simplest objective of reachability goals. The algorithm runs in polynomial time in the size of the state space and hence yields an exponential time upper bound for the plan existence problem. The algorithm uses the subprocedure *prune* given in Figure 4.3.

We introduce some terminology. Let S be a set of states, O a set of operators, and $x : S \to O$ a mapping from states to operators. A sequence $s_0, \ldots, s_n$ of states is an *execution* if for all $i \in \{1, \ldots, n\}$ there is $o \in O$ such that $s_i \in img_o(s_{i-1})$, and it is an *execution of* x if $s_i \in img_{x(s_{i-1})}(s_{i-1})$ for all $i \in \{1, \ldots, n\}$.

Lemma 4.18 (Procedure prune) *Let S be a set of states, O a set of operators and $G \subseteq S$ a set of states. Then the procedure call prune(S,O,G) will terminate after a finite number of steps returning a set of states $W \subseteq S$ such that there is function $x : W \to O$ such that*

1. *for every $s \in W$ there is an execution $s_0, s_1, \ldots, s_n$ of x with $n \geq 1$ such that $s = s_0$ and $s_n \in G$,*
2. *$img_{x(s)}(\{s\}) \subseteq W \cup G$ for every $s \in W$, and*
3. *There is no function x satisfying the above properties for states not in W: for every $s \in S \backslash W$ and function $x' : S \to O$ there is an execution $s_0, \ldots, s_n$ of x' such that $s = s_0$ and there is no $m \geq n$ and execution $s_n, s_{n+1}, \ldots, s_m$ such that $s_m \in G$.*

Proof: The proof is by two nested induction proofs that respectively correspond to the repeat-until loops on lines 9 and 13 in the procedure *prune*. If there is no plan that is guaranteed to reach a goal state from a state s, then this is because for any plan after some number of executions steps i it is possible to reach a state from which no sequence actions can reach a goal state. A plan covering all other states exists with an execution reaching a goal state in some h steps. The outer loop and induction go through $i = 0, 1, 2, \ldots$ and the inner loop and induction through $h = 0, 1, 2, \ldots$.

Induction hypothesis: There is function $x : W_i \to O$ such that

1. for every $s \in W_i$ there is an execution $s_0, \ldots, s_n$ of x such that $n \geq 1$, $s = s_0$ and $s_n \in G$,
2. $\mathit{img}_{x(s)}(\{s\}) \subseteq W_{i-1} \cup G$ for every $s \in W_i$, and
3. for all functions $x' : S \to O$ and states $s \in S \backslash W_i$ there is $i' \in \{0, \ldots, i\}$ and an execution $s_0, \ldots, s_{i'}$ of x' such that $s_0 = s$ and there is no $h \geq i'$ and execution $s_{i'}, s_{i'+1}, \ldots, s_h$ such that $s_h \in G$.

Base case $i = 0$:

1. W_0 has been computed to fulfill exactly this property. We denote the value of the variables W_0 in the end of iteration i of the first repeat-until loop by $W_{0,i}$.

   Induction hypothesis:

   (a) There is a function $x : W_{0,j} \to O$ such that there is an execution of x for every $s \in W_{0,j}$ of length $j \geq 1$ reaching a state in G.

   (b) For states not in $W_{0,j}$ there is no x with this property.

   Base case $j = 1$: After the first iteration $W_{0,1} = \bigcup_{o \in O} \mathit{preimg}_o(G)$. Hence for every $s \in W_{0,1}$ assign $x(s) = o$ for any o such that $s \in \mathit{preimg}_o(G)$.

   (a) Now there is an execution of length 1 from any $s \in W_{0,1}$ to a state in G.

   (b) For states not in $W_{0,1}$ no operator alone may reach a state in G.

   Inductive case $j \geq 2$: By induction hypothesis there is a function x with execution of length $j - 1 \geq 1$ for reaching a state in G for every state for which such an execution exists. We extend this function to cover states $s \in W_{0,j} \backslash W_{0,j-1}$ as follows: $x(s) = o$ for any o such that $s \in \mathit{preimg}_o(W_{0,j-1} \cup G)$.

   (a) For any $s \in W_{0,j}$ there is an execution of x reaching a state in G because for states $s \in W_{0,j-1}$ this is by the induction hypothesis, and for states in $W_{0,j} \backslash W_{0,j-1}$ applying the operator $x(s)$ may reach a state in $W_{0,j-1}$ for which an execution reaching G exists by the induction hypothesis.

   (b) Let s be a state such that there is a function $x' : S \to O$ with an execution that reaches G from s with j steps. Hence there is a state s' for which an execution with x' reaches G from s' with $j - 1$ steps. Hence by the induction hypothesis $s \in W_{0,j-1}$ and consequently $s \in \mathit{preimg}_{x'(s)}(W_{0,j-1})$. Therefore for any state not in $W_{0,j}$ there is no such function x'.

2. As $W_{-1} = S$ trivially $\mathit{img}_{x(s)}(\{s\}) \subseteq W_{-1} \cup G$.

3. States $s \in W_0 \backslash W_{-1}$ are exactly those states from which no operator sequence leads to G by construction of W_0, as shown above.

Inductive case $i \geq 1$: For the inner *repeat-until* loop we prove inductively the following.
Induction hypothesis: There is function $x : S_k \rightarrow O$ such that

1. for every $s \in S_k$ there is an execution $s_0, s_1, \ldots, s_n$ of x such that $n \in \{1, \ldots, k\}$, $s = s_0$ and $s_n \in G$,
2. $\mathit{img}_{x(s)}(\{s\}) \subseteq W_{i-1} \cup G$ for every $s \in S_k$, and
3. for all functions $x' : S \rightarrow O$ and states $s \in S \backslash S_k$ either
   (a) there is $i' \in \{0, \ldots, i\}$ and an execution $s_0, \ldots, s_{i'}$ of x' such that $s_0 = s$ and there is no $h \geq i'$ and execution $s_{i'}, s_{i'+1}, \ldots, s_h$ such that $s_h \in G$, or
   (b) there is no $k' \in \{1, \ldots, k\}$ and an execution $s_0, \ldots, s_{k'}$ of x' such that $s_0 = s$ and $s_{k'} \in G$.

Base case $k = 0$: Since $S_0 = \emptyset$, cases (1) and (2) trivially hold for every $s \in S_0$. It remains to show the third component of the induction hypothesis.

3. For any $s \in S \backslash S_0 = S$ (3b) is satisfied because it requires executions to be longer than $k = 0$.

Inductive case $k \geq 1$: We extend the function $x : S_{k-1} \rightarrow O$ to cover states in $S_k \backslash S_{k-1}$. Let s be any state in S_k. If $s \in S_{k-1}$ then properties (1) and (2) are by the induction hypothesis. Otherwise $s \in S_k \backslash S_{k-1}$. Therefore by definition of S_k, $s \in \mathit{preimg}_o(S_{k-1} \cup G) \cap \mathit{spreimg}_o(W_{i-1} \cup G)$ for some $o \in O$.

1. As $s \in \mathit{preimg}_o(S_{k-1} \cup G)$ for some $o \in O$, by (4) of Lemma 2.2 either $s \in \mathit{preimg}_o(S_{k-1})$ or $s \in \mathit{preimg}_o(G)$.

   If $s \in \mathit{preimg}_o(G)$ then we set $x(s) = o$. The desired execution consists of s and a state $s' \in G$.

   If $s \in \mathit{preimg}_o(S_{k-1}) \backslash \mathit{preimg}_o(G)$ then there is a state $s' \in S_{k-1}$ such that $s' \in \mathit{img}_o(\{s\})$. By the induction hypothesis there is an execution of x starting from s' that ends in a goal state. For s such an execution is obtained by prefixing with o, so we define $x(s) = o$.

2. Since $s \in \mathit{spreimg}_o(W_{i-1} \cup G)$ by (2) and (3) of Lemma 2.2 $\mathit{img}_o(\{s\}) \subseteq W_{i-1} \cup G$.

3. Take any $s \in S \backslash S_k$. Now for every operator $o \in O$, either $s \notin \mathit{spreimg}_o(W_{i-1} \cup G)$ or $s \notin \mathit{preimg}_o(S_{k-1} \cup G)$. Consider any function $x' : S \rightarrow O$ such that $x'(s) = o$.

   In the first case by the outer induction hypothesis there is $i' \in \{0, \ldots, i-1\}$ and an execution $s_0, \ldots, s_{i'}$ of x' such that $s_0 \in \mathit{img}_o(s)$ and there is no $h \geq i'$ and execution $s_{i'}, s_{i'+1}, \ldots, s_h$ such that $s_h \in G$. Hence executing o first could similarly lead to the state $s_{i'}$ from which no goal could be reached, now requiring i steps.

   In the second case by the inner induction hypothesis for all $s' \in \mathit{img}_o(s)$ there is no execution of length $k-1$ ending in a goal state.

   Since this holds for any $o \in O$, every x' has one of these properties.

1: **procedure** planexistsFO(S,I,O,G)
2: G_{ne} := G;
3: **repeat**
4: W := prune(S,O,G_{ne});
5: G'_{ne} := G_{ne};
6: G_{ne} := $G_{ne} \cap W$;
7: **until** $G_{ne} = G'_{ne}$;
8: **if** $I \subseteq W$ **then return** true **else return** false;

Figure 4.4: Algorithm for nondeterministic planning with full observability

This completes the inner induction. To establish the induction step of the outer induction consider the following. The inner repeat-until loops ends when $S_k = S_{k-1}$. This means that $S_z = S_k$ for all $z \geq k$. Hence executions for reaching a goal state for (1) and (3) are allowed to have arbitrarily high length k. The outer induction hypothesis is obtained from the inner induction hypothesis by removing the upper bound and replacing S_k by W_i. By construction $W_i = S_k$.

This finishes the outer induction proof. The claim of the lemma is obtained from the outer induction hypothesis by noticing that the outer loop exits when $W_i = W_{i-1}$ (it will exit after a finite number of iterations because W_0 is finite and its size decreases on every iteration) and by replacing both W_i and W_{i-1} by W we obtain the claim of the lemma. □

The main procedure of the decision procedure is given in Figure 4.4. The procedure first sets $G_{ne} := G$, and then repeatedly eliminates from G_{ne} those goal states for which there is no plan that is guaranteed to reach a state in G_{ne}. The elimination of these states is by the subprocedure *prune* (Figure 4.3 and Lemma 4.18).

After the last iteration of the loop G_{ne} will be the maximal subset of G from which reaching a state in G_{ne} is guaranteed. The number of iterations is bounded by the number of states, and each iteration has a runtime polynomial in the number of states. Hence the total runtime of this stage is exponential in the size of the succinct transition system.

The last line of the procedure tests whether from every initial state a state in G_{ne} can be reached. If so, then is a plan x so that from any state that can be reached from an initial state by using this plan, there is a path to a goal state in G_{ne}, including the state in G_{ne}. Hence the RRG criterion is satisfied.

Theorem 4.19 *Testing plan existence for succinct transition systems with full observability under the repeated reachability criterion (PLANSAT-FO-ND-RRG) is in EXP.*

Proof: Given a succinct transition system Π, we can produce the corresponding transition system $F(\Pi) = (S, I, O, G, P)$ in exponential time. Then we call the procedure planexistsFO(S,I,O,G) which is given in Figure 4.4.

We prove an auxiliary result by induction on the number of iterations the loop makes. Let W_i and $G_{ne,i}$ be the values of the program variables W and G_{ne} respectively in iteration i and $G_{ne,0}$ is the initial value G of G_{ne}.

Induction hypothesis: There is $x : W_i \to O$ such that for all states s,

1. if $s \in W_i$ then there is an execution $s_1, \ldots, s_n$ of x such that $s_1 = s$ and $s_n \in G_{ne,i-1}$,
2. if $s \in W_i$ then $img_{x(s)}(s) \subseteq W_i$, and

3. if $s \notin W_i$ then for all $x' : S \to O$ there is an execution $s_1, \ldots, s_n$ of x' such that $s_1 = s$ and there is no execution $s_n, \ldots, s_m$ for any $m \geq n$ such that $s_m \in G$.

Base case $i = 1$: All three cases are directly by Lemma 4.18.
Inductive case $i \geq 2$:

1. By Lemma 4.18 there is an execution leading to $G_{ne,i-1}$ for every $s \in W_i$.

2. By Lemma 4.18 $img_{x(s)}(s) \subseteq W_i$ for every $s \in W_i$.

3. By Lemma 4.18 for every $s \notin W_i$ and every $x' : S \to O$ there is an execution $s_1, \ldots, s_n$ of x' such that $s_1 = s$ and there is no execution $s_n, \ldots, s_m$ for any $m \geq n$ such that $s_m \in G_{ne,i-1}$.

   Take any execution $s_n, \ldots, s_m$ for any $m \geq n$. If $s_m \notin G$ then this execution does not lead to G. If $s_m \in G$ then by the induction hypothesis and the fact that $s_m \notin G_{ne,i-1} \subseteq G$ there is an execution $s_m, \ldots, s_{m'}$ such that there is no execution $s_{m'}, \ldots, s_{m''}$ such that $s_{m''} \in G$.

   Hence for the execution $s_1, \ldots, s_n, \ldots, s_m, \ldots, s_{m'}$ there is no execution $s_{m'}, \ldots, s_{m''}$ such that $s_{m'} \in G$.

The termination condition of the loop guarantees that from any state in W and $G_{ne} \subseteq W$ a state in G_{ne} is eventually reached.

Hence there is a plan $x : S \to O$ such that for all $s \in I$ and for all executions $s_1, \ldots, s_n$ of x such that $s_1 = s$ there is an execution $s_n, \ldots, s_m$ of x such that $s_m \in G_{ne} \subseteq G$. This satisfies the RRG criterion.

Assume $I \not\subseteq W$ i.e. there is $s \in I$ such that $s \notin W$. Then by the above results for all $x : S \to O$ there is an execution $s_1, \ldots, s_n$ of x such that $s_1 = s$ and $s_m \notin G$ for all $m \geq n$ and executions $s_n, \ldots, s_m$. Hence if the algorithm returns *false* there is no plan that would satisfy the RRG criterion. □

4.3.2 Planning with unobservability

Next we investigate the repeated reachability criterion for planning without observability and for the most general case with partial observability. As pointed out in Example 4.4, the repeated reachability criterion does not require that at any point of time it is known that the current state is one of the goal states. This is why planning with the repeated reachability criterion does not trivially reduce to planning with goal reachability.

Representing and recognizing dependencies between a belief state and its predecessor belief states becomes necessary, and unlike for reachability goals, plans cannot be formalized as mappings from the current belief state to an action, that is, the belief states do not include all the necessary information needed for testing the satisfaction of the plan objective.

The EXPSPACE lower bound for the complexity of the most general unobservable problem PLANSAT-UO-ND-RRG is given by Theorem 4.12. We prove the EXPSPACE upper bound by devising a corresponding decision procedure. The structure of plans for unobservable problems is the same as for the PSPACE-complete fully observable deterministic problem with one initial state considered in Theorem 4.16: a sequence of actions followed by a loop. The complication in the unobservable case is that it is not necessarily ever known when a goal state is visited. The

algorithm we give is an extension of the algorithm given in Theorem 4.16. First a path to the starting belief state of a loop is found. Then a loop (a non-trivial path from the belief state to itself) is found. Simultaneously with the loop we find for each state in the belief state an execution that visits a goal state, showing that the loop satisfies the RRG criterion. All this can be done by using only an exponential amount of memory, and hence the EXPSPACE membership of the decision problem follows.

To show that this algorithm is complete, we first show that if a plan exists, then a plan with the given structure with an initial segment followed by a loop exists. Then we derive an upper bound on the length of such a loop.

In the unobservable case plans with an infinite execution reduce to a particularly simple form: a sequence σ of operators (the prefix) followed by another sequence σ' of operators (the loop) that is repeated to produce the infinite sequence of operators $\sigma\sigma'\sigma'\sigma'\ldots$. This is the structure of infinite paths in a graph with only nodes of degree one. This plan induces an infinite sequence of belief states, similarly consisting of a prefix σ_B followed by a loop σ'_B, but the lengths of σ_B and σ'_B do not necessarily equal the lengths of σ and σ'.

Any plan can be transformed so that the length of the prefix and the loop for both operators and the belief states coincide. Let B_1 be the belief state reached after executing σ. After also executing σ' we may reach some other belief state $B_2 \neq B_1$. Let $B_1, B_2, \ldots, B_n, \ldots$ be the sequence of belief states obtained this way, each B_i reached from the initial belief state by executing σ and then σ' $i-1$ times. Since the number of belief states is finite, for some $n \geq 1$ and $k \in \{1, \ldots, n-1\}$, $B_k = B_n$. Let n be the minimal such n. Now from B_n executing σ' $n-k$ times takes us to B_n. Now we can view $\sigma\sigma'^{k-1}$ as the prefix of the plan and σ'^{n-k} as the loop. This loop has the property that B_n is its first belief state on every iteration.

We now derive an upper bound on the length of such a loop. Let $o_1, \ldots, o_n$ be a loop with belief states $B_0, \ldots, B_n$ such that $B_0 = B_n$ and let $s_0, \ldots, s_n$ be an execution visiting a goal state for one state $s_0 \in B_0$, that is, $s_i \in G$ for some $i \in \{0, \ldots, n\}$. Assume for some $i, j, k \in \{0, \ldots, n\}$ such that $i \neq j \neq k \neq i$ both $B_i = B_j = B_k$ and $s_i = s_j = s_k$. If $s_g \in G$ for $g \in \{i, \ldots, j-1\}$ then we obtain a shorter loop visiting G for s_0 by eliminating operators $o_j, \ldots, o_{k-1}$, and if $g \in \{j, \ldots, k-1\}$ we eliminate operators $o_i, \ldots, o_{j-1}$. Hence there need to be at most two occurrences of any B_i, s_i, and the loop needs to have a length of at most $2^{|S|}2|S|$. Now we can concatenate the loops for all $s \in B_0$, obtaining a loop that can visit a goal state from any state in B_0 and having a length of at most $2^{|S|}2|S|^2$.

Theorem 4.20 *Testing plan existence for succinct transition systems without observability under the repeated reachability criterion (PLANSAT-UO-ND-RRG) is in EXPSPACE.*

Proof: For a given a succinct transition system Π we first produce the corresponding transition system $F(\Pi) = \langle S, I, O, G, (S)\rangle$ in exponential time. Then we find a plan by using the algorithm with the main procedure given in Figure 4.7. The outer loop iterates over all belief states B that may be the first in the loop that follows. This iteration takes only exponential space.

Let $n = |S|$ be the cardinality of the state space.

For each B first test with UOreach(S,O,I,B,n) that B is reachable from I. The procedure UOreach is the standard recursive binary search algorithm for testing the existence of paths of length 2^n between two belief states, similar to the procedure FOreach in Figure 4.2.

Then it is tested whether there is a loop from B to itself and for every $s \in B$ there is an execution of the loop that visits a goal state. For testing the existence of such executions we use mappings $r : B \rightarrow S$. These mappings are used by procedure *UOreachr* to keep track of the

1: **procedure** UOreach(S,O,B,B',m)
2: **if** $m \leq 0$ **then**
3: **if** $B = B'$ **or** ($B \subseteq \{s \in S | sos'$ for some $s' \in S\}$ **and** $B' = img_o(B)$ for some $o \in O$)
4: **then return** true
5: **else**
6: **for each** $B'' \subseteq S$ **do**
7: **if** UOreach(S,O,B,B'',$m-1$) **and** UOreach(S,O,B'',B',$m-1$) **then return** true;
8: **end**;
9: **return** false

Figure 4.5: Algorithm for finding a sequence of actions leading from B to B'

1: **procedure** UOreachr(S,O,B,r,B',r',m,Q)
2: **if** $m \leq 0$ **then**
3: **if** ($B = B'$ **and** $r = r'$) *(* Reachability by 0 actions *)*
4: **or** (there is $o \in O$ such that $B \subseteq \{s \in S | sos'$ for some $s' \in S\}$ **and** $B' = img_o(B)$
5: **and** ($r'(s) \in img_o(r(s))$ or ($r(s) = r'(s)$ and $r(s) \in G$)) for all $s \in Q$)
6: **then return** true
7: **else**
8: **for each** $B'' \subseteq S$ and function $r'' : Q \to S$ **do**
9: **if** UOreachr(S,O,B,r,B'',r'',$m-1$,Q) **and** UOreachr(S,O,B'',r'',B',r',$m-1$,Q)
10: **then return** true;
11: **end**;
12: **return** false

Figure 4.6: Algorithm for finding a sequence of action from B and B' that may visit G

1: **procedure** UOplanexistence(S,I,O,G)
2: **for each** $B \subseteq S$, operator $o \in O$ and functions $r : B \to S$ and $r' : B \to G$ **do**
3: **if** $r(s) \in img_o(s)$ for all $s \in B$
4: **and** UOreach(S,O,I,B,n)
5: **and** UOreachr(S,O,$img_o(B)$,r,B,r',$n+1+2\lceil \log_2 n \rceil$,$B$) **then return** true
6: **end do**;
7: **return** false

Figure 4.7: A decision procedure for testing plan existence without observability

visited states. That $r(s) = s'$ for some $s \in B$ means that at a given point of execution the state s' could have been reached when starting from $s \in B$. The value $r(s)$ is updated on each operator application when the loop is constructed until $r(s)$ is a goal state. After reaching a goal state $r(s)$ is not updated further. Hence if $r'(s)$ is a goal state when a loop from B back to itself has been found then there is an execution of the loop that visits a goal state when execution starts from s.

The main procedure iterates over possible starting belief states B of a loop and also over $r : B \to S$ and $r' : B \to G$, and then tests with *UOreachr* whether a loop that visits a goal state for every $s \in B$ exists. The procedure *UOreachr* can return an empty path, and therefore the main procedure iterates also over the possible first operators $o \in O$ of a loop. The initial values of $r(s)$ have to correspond to the chosen first operator of the loop (line 3). On line 5 the procedure call UOreachr(S,O,$img_o(B)$,r,B,r',$n + 1 + 2\lceil \log_2 n \rceil$) is made. If the procedure returns true, then there is an operator sequence from B to $img_o(B)$ to B, and since $r'(s) \in G$ for every $s \in B$ there is for every $s \in B$ an execution of this operator sequence that visits a goal state.

The maximum recursion depth is the logarithm of the maximum loop length:

$$\log_2(2^n 2n^2) = \log_2 2^n + 1 + \log_2 n^2 = n + 1 + 2\log_2 n.$$

□

4.3.3 Planning with partial observability

A 2-EXP upper bound for partially observable problems under the maintenance and reachability criteria can be directly obtained by using the algorithms for the corresponding fully observable problems (Theorem 4.19): view belief states (sets of states) as states and view belief states that consist of goal states only as goal states. For reachability and maintenance the plans that in Theorem 4.19 are mappings from states to operators can be directly translated to plans according to the general definition: each belief state corresponds to a plan node, the associated operator is applied, and the observations are used to choose the successor plan node (belief state.)

For the RRG criterion and partial observability the algorithm of Theorem 4.19 cannot be directly used because there are plans that are guaranteed to repeatedly reach a goal state but the belief state never consists of goal states only, as pointed out in Example 4.4.

Theorem 4.21 *Testing plan existence for succinct transition systems with partial observability under the reachability criterion (PLANSAT-PO-ND-RG) is in 2-EXP.*

Theorem 4.22 *Testing plan existence for succinct transition systems with partial observability under the maintenance criterion (PLANSAT-PO-ND-MG) is in 2-EXP.*

For deterministic operators the belief state size monotonically decreases as plan execution proceeds. This leads to a particularly simple structure for plans when executions are infinite: they consist of a branching non-looping part that ends in any of belief states $B_1, \ldots, B_n$. The sum of the cardinalities of these belief states is less than or equal to the cardinality of the initial belief state. Further, for each belief state $B \in \{B_1, \ldots, B_n\}$ there is a non-branching loop to itself so that for any starting state $s \in B$ there is an execution that visits a goal state, and all belief states in this loop have the same cardinality. Observations at this stage do not reduce the size of the belief states.

```
1:  procedure reachPOdet(S,O,B,L,(C_1,...,C_k),i)
2:  if i = 0 then
3:    begin
4:      for each j ∈ {1,...,k}
5:        if img_o(B) ∩ C_j ⊆ B' for no B' ∈ L and o ∈ O such that B ⊆ {s ∈ S|sos' for some s' ∈ S}
6:        and B ∩ C_j ⊆ B' for no B' ∈ L
7:        then return false
8:      end
9:      return true
10:   end
11: else
12:   for each L' ⊆ 2^S such that Σ_{B'∈L'} |B'| ≤ |B| and
13:                          for every B' ∈ L', B' ⊆ C_j for some j ∈ {1,...,k}
14:     if reachPOdet(S,O,B,L',(C_1,...,C_k),i − 1)
15:       and reachPOdet(S,O,B'',L,(C_1,...,C_k),i − 1) = true for all B'' ∈ L'
16:     then return true
17:   end for
18: return false
```

Figure 4.8: Algorithm that tests reachability of sets of belief states by deterministic operators in EXPSPACE

Theorem 4.23 *Assuming partial observability and deterministic operators, testing the existence of a plan that always ends in one of the belief states $B_1, \ldots, B_n$ can be done in exponential space.*

Proof: The idea is similar to the EXPSPACE membership proof of planning without observability: go through all possible intermediate stages of a plan by binary search (intermediate stage = a maximal set of plan nodes of which none is a successor of another.) Determinism yields an exponential upper bound on the sum of the cardinalities of the belief states that are possible after a given number of actions, and it also entails that no belief state has to be visited more than once (= acyclic plans). Hence plan executions have a doubly exponential length, and binary search only needs exponential recursion depth. As only an exponential amount of memory is needed at each call, the whole memory consumption is exponential.

Let $F(\Pi) = \langle S, I, O, G, (C_1, \ldots, C_k)\rangle$ be the transition system corresponding to a given succinct transition system Π. For belief state B and set L of belief states with $\sum_{B' \in L} |B'| \leq |S|$, test reachability of belief states in L from B with plans of depth 2^i by algorithm in Figure 4.8.

Testing plan existence is by calling reachPOdet(S,O,B,L,$(C_1, \ldots, C_k)$,$|S|$) for every $B = I \cap C_i, i \in \{1, \ldots, k\}$. Here $L = \{B_1, \ldots, B_n\}$. The algorithm always terminates, and a plan exists if and only if answer *true* is obtained in all cases.

The space consumption is (only) exponential because the recursion depth is exponential and the sets $L' \subseteq 2^S$ with $\sum_{B' \in L'} |B'| \leq |B|$ have size $\leq |S|$. This small sets L' suffice because all operators are deterministic, and after any number of actions, independently of how the plan has branched, the sum of the cardinalities of the possible belief states is not higher than the cardinality of the set of initial states. This is because the size of a successor belief state cannot be bigger, and the sum of sizes of belief states following a branch equals the size of the predecessor belief state. □

```
1:  procedure planexistsPOdet(S,I,O,G,(C_1,...,C_k))
2:  n := |S|;                                          (* Number of states *)
3:  for all sets L ⊆ 2^S such that Σ_{B∈L} |B| < |I| do
4:    if reachPOdet(S,O,I,L,(C_1,...,C_k),n) then
5:      if for every B ∈ L                             (* Starting belief state of a loop *)
6:        for some r : B → S and r' : B → G and o ∈ O
7:          such that r(s) ∈ img_o(s) for all s ∈ B
8:          UOreachr(S,O,img_o(B),r,B,r',n + 1 + ⌈log_2 n⌉,B)
9:      then return true;
10: end for
11: return false
```

Figure 4.9: Algorithm for testing plan existence for deterministic operators and partial observability

Theorem 4.24 *Testing plan existence for succinct transition systems with partial observability and deterministic operators under the repeated reachability criterion (PLANSAT-PO-D-RRG) is in EXPSPACE.*

Proof: The procedure for testing plan existence is given in Figure 4.9. The algorithm loops over the sets of starting belief states of loops (the sum of their cardinalities is at most the cardinality of the initial belief state).

For each potential set of loops' starting belief states, the existence of an acyclic plan for reaching these belief states is tested by the procedure *reachPOdet* that runs in exponential space by Theorem 4.23.

Finally, the existence of a goal-visiting non-branching loop for each of the designated starting belief states of a loop is tested by the procedure *UOreachr*, which runs in exponential space by Theorem 4.20. □

Planning is still simpler when operators are state-independent so that the changes to the state variables are always the same.

Theorem 4.25 *Testing plan existence for succinct transition systems with partial observability and deterministic state-independent operators under the repeated reachability criterion (PLANSAT-PO-ID-RRG) is in PSPACE.*

Proof: Since the set of observable state variables is always the same and no information about the unobservable state variables can be obtained indirectly (through operators with conditional effects), it suffices to initially observe everything, branch, and for every resulting belief state find a sequential plan, consisting of a prefix and a loop that visits a goal state.

The algorithm represents belief states as triples (I, L_o, L_c) where I is the initial state formula, L_o is the set of literals corresponding to the observations made in the initial state, and L_c is a consistent set of literals characterizing the changes to state variables made by the operators.

The notation $s \in B$ for a belief state $B = (I, L_o, L_c)$ means that there is a state s_I such that $s_I \models \{I\} \cup L_o$ and s is obtained from s_I by making literals in L_c true. This can be tested in

1: **procedure** 3app(o,B,s,B',s')
2: **let** $\langle I, L_o, L_c\rangle = B$;
3: **let** $\langle I', L'_o, L'_c\rangle = B'$;
4: **if** $s \not\models \text{prec}(o)$ for some $s \in B$
5: **or** $img_o(s) \neq \{s'\}$
6: **or** $I \neq I'$
7: **or** $L_o \neq L'_o$
8: **or** $L'_c \neq \text{eff}(o) \cup \{l \in L_c | \bar{l} \notin \text{eff}(o)\}$ **then return** false;
9: **return** true

Figure 4.10: Test for reachability of B', s' from B, s by operator o

nondeterministic polynomial time and hence in polynomial space.

This belief state representation is 3-valued in the sense that the value of each state variable is *true*, *false* or unknown. The dependencies between state variable values are taken into account in the membership tests $s \in B$ by looking at the initial state formula I and the initial observations.

For n state variables, a given initial state formula I and given initial observations L_o there are only $\mathcal{O}(3^n)$ belief states that can be reached from an initial belief state $B = \langle I, L_o, \emptyset\rangle$. Each such belief state $B' = \langle I, L_o, L_c\rangle$ is uniquely characterized by the set L_c of changes made to the state variables by state-independent operators. Since no sequential plan for reaching a belief state has to visit any belief state more than once, such plans have a length $\mathcal{O}(3^n)$.

The main procedure *planexistsPOsi* of the algorithm is given in Figure 4.13. The procedure considers all initial belief states B induced by the possible initial observations. Then the procedure iterates over belief states B_0 that could start a loop for B. First, B_0 must be reachable from B. This is tested with *3reach*. Second, for every state $s_0 \in B_0$ there must be a sequence of operators that visits G and returns to B_0. This is tested with *3reachG*. Since the operator sequence for every $s_0 \in B_0$ may be different the actual loop is obtained by juxtaposing the operator sequences for every $s_0 \in B_0$.

The procedures *3reach* and *3reachG* (Figures 4.11 and 4.12) are recursive similarly to the procedure *FOreach* in Figure 4.2. Since there are only $\mathcal{O}(2^n)$ belief states the recursion depth is only polynomial. Memory consumption at each call is only polynomial in the number of state variables. Hence the whole memory consumption is only polynomial in the size of the succinct transition system, and the plan existence problem is in PSPACE.

In both procedures the base case tests one-step reachability with the algorithm *3app* (Figure 4.10) that tests applicability of operators in a 3-valued belief state. In the procedure *3app* the set of literals in the effect e of an operator $\langle c, e\rangle$ is denoted by eff(o) and the precondition c by prec(o). □

4.4 Summary of the results

We summarize the results of this work and results established in earlier work.

```
1:  procedure 3reach(A,O,B,s,B',s',m)
2:  let ⟨I, L_o, L_c⟩ = B;
3:  if m ≤ 0 then
4:    if (B = B' and s = s') or 3app(o,B,s,B',s') for some o ∈ O then return true
5:    else return false
6:  else
7:    for each B'' ∈ {I} × {L_o} × 2^{A∪{¬a|a∈A}} and s'' ∈ B'' do
8:      if 3reach(A,O,B,s,B'',s'',m − 1) and 3reach(A,O,B'',s'',B',s',m − 1)
9:      then return true
10:   end for;
11: return false
```

Figure 4.11: Algorithm for finding a path from B, s to B', s'

```
1:  procedure 3reachG(A,O,G,B,s,B',s',m)
2:  let ⟨I, L_o, L_c⟩ = B;
3:  if m ≤ 0 then
4:    if ((B = B' and s = s') or 3app(o,B,s,B',s') for some o ∈ O) and {s, s'} ∩ G ≠ ∅
5:    then return true else return false
6:  else
7:    for each B'' ∈ {I} × {L_o} × 2^{A∪{¬a|a∈A}} and state s'' ∈ B'' do
8:      if 3reachG(A,O,G,B,s,B'',s'',m − 1) and 3reach(A,O,B'',s'',B',s',m − 1)
9:      then return true;
10:     if 3reach(A,O,B,s,B'',s'',m − 1) and 3reachG(A,O,G,B'',s'',B',s',m − 1)
11:     then return true
12:   end for
13:   return false
```

Figure 4.12: Algorithm for finding a path from B to B' that visits G

```
1:  procedure planexistsPOsi(A,I,O,G,V)
2:  if for every B = ⟨I, L_o, ∅⟩ ∈ {I} × 2^{A∪{¬a|a∈A}} × {∅} and s : A → {0, 1} such that
3:            L_o ∪ {I} is consistent
4:            {a ∈ A|a ∈ L_o or ¬a ∈ L_o} = V and          (* All observations *)
5:            s ∈ B                                       (* Every matching initial state *)
6:    for some B_0 ∈ {I} × {L_o} × 2^{A∪{¬a|a∈A}} and s_0 ∈ B_0 (* Loop's starting belief state *)
7:      3reach(A,O,B,s,B_0,s_0,|A|) and                   (* Reachable from the initial belief state *)
8:      for every s' ∈ B_0
9:        for some s'' ∈ B_0
10:         3reachG(A,O,G,B_0,s',B_0,s'',|A|)             (* Loop from s' with visit to G *)
11: then return true
```

Figure 4.13: Algorithm for testing plan existence for deterministic state-independent operators and partial observability

	transition type		
observability	state-independent deterministic (ID)	state-dependent deterministic (D)	state-dependent non-deterministic (ND)
full (FO)	PSPACE (T4.10)	PSPACE (T4.10)	EXP (T4.14)
no (UO)	PSPACE (T4.10)	EXPSPACE (T4.12)	EXPSPACE (T4.12)
partial (PO)	PSPACE (T4.10)	EXPSPACE (T4.12)	2-EXP (T4.15)

Table 4.1: Complexity lower bound (hardness) theorems for reachability and several initial states (PLANSAT-??-??-RG)

	transition type		
observability	state-independent deterministic (ID)	state-dependent deterministic (D)	state-dependent non-deterministic (ND)
full (FO)	PSPACE (T4.10)	PSPACE (T4.10)	EXP (T4.14)
no (UO)	PSPACE (T4.10)	PSPACE (T4.10)	EXPSPACE (T4.12)
partial (PO)	PSPACE (T4.10)	PSPACE (T4.10)	2-EXP (T4.15)

Table 4.2: Complexity lower bound (hardness) theorems for reachability and one initial state (PLANSAT-??-??-RG-1)

4.4.1 Lower bounds for reachability

On goal reachability we have hardness proofs for certain complexity classes for three different degrees of observability (unobservable, full, partial) and transitions with three types of restrictions (nondeterministic, deterministic state-dependent, deterministic state-independent). References to theorems are given in Tables 4.1 and 4.2, the first for succinct transition systems with several initial states, and the second with one initial state.

4.4.2 Upper bounds for reachability

For reachability we have constructive proofs of membership in the same complexity classes we already proved hardness for reachability. These results show that the plan existence problems are complete for the complexity classes in questions. References to theorems are given in Tables 4.3 and 4.4. With the exception of Theorem 4.21 the results were established with repeated reachability and therefore they also hold for maintenance. 2-EXP upper bound for maintenance is stated in Theorem 4.22.

	transition type		
observability	state-independent deterministic (ID)	state-dependent deterministic (D)	state-dependent non-deterministic (ND)
full (FO)	PSPACE (T4.16)	PSPACE (T4.16)	EXP (T4.19)
no (UO)	PSPACE (T4.25)	EXPSPACE (T4.20)	EXPSPACE (T4.20)
partial (PO)	PSPACE (T4.25)	EXPSPACE (T4.24)	2-EXP (T4.21)

Table 4.3: Complexity upper bound theorems for reachability and several initial states (PLANSAT-??-??-RG)

	transition type		
observability	state-independent deterministic (ID)	state-dependent deterministic (D)	state-dependent non-deterministic (ND)
full (FO)	PSPACE (T4.16)	PSPACE (T4.16)	EXP (T4.19)
no (UO)	PSPACE (T4.16)	PSPACE (T4.16)	EXPSPACE (T4.20)
partial (PO)	PSPACE (T4.16)	PSPACE (T4.16)	2-EXP (T4.21)

Table 4.4: Complexity upper bound theorems for reachability and one initial state (PLANSAT-??-??-RG-1)

Chapter 5

Algorithms for nondeterministic planning

In Chapter 4 we proved complexity upper bounds for a number of nondeterministic planning problems by giving resource-bounded algorithms that solve these problems. For the partially observable problems these algorithms are in most cases not practical because of the very high number of belief states these algorithms generate. In this chapter we consider more practical algorithms for the some of these problems. Unlike the algorithms in Chapter 4, these algorithms typically generate only a very small fraction of the belief states explicitly, and therefore they are practical for much bigger problems.

Section 5.1.1 generalizes the regression operation for deterministic operators from Section 3.1.2 to nondeterministic operators. This regression operation can be used as a part of algorithms for nondeterministic planning. In Section 5.1.2 we give a translation of nondeterministic operators into the classical propositional logic.

Section 5.2 shows how the image and preimage operations used in some of the algorithms in Chapter 4 can be implemented by formula manipulation and how the nondeterministic regression operation is a special case of the general method of computing preimages. These techniques have earlier been used in connection with symbolic model-checking [Burch *et al.*, 1994; Clarke *et al.*, 1994; Bryant, 1992], and they can be used as an implementation technique of some of the algorithms in Sections 4.3.1 and 5.4.

In the rest of the chapter we present some more practical approaches for synthesizing plans under observability restrictions. Section 5.3.1 presents a more advanced translation of sets of nondeterministic operators into the propositional logic and shows how it can be used as a part of a quantified Boolean formula that represents the planning problem without observability. In Section 5.3.2 a powerful family of heuristics for search in the belief space is defined. Section 5.4 presents a framework for plan search in the presence of partial observability.

5.1 Nondeterministic operators

In this section we will present a basic translation of nondeterministic operators into the propositional logic and a regression operation for nondeterministic operators. In the next sections we will discuss a general framework for computing with nondeterministic operators and their transition relations which are represented as propositional formulae. This framework provides techniques

for computing both regression and progression for sets of states that are represented as formulae.

5.1.1 Regression for nondeterministic operators

Regression for deterministic operators is given in Definition 3.5. It can be easily generalized to a subclass of nondeterministic operators.

Definition 5.1 (Regression for nondeterministic operators) *Let ϕ be a propositional formula and $o = \langle c, e_1 | \cdots | e_n \rangle$ an operator where $e_1, \ldots, e_n$ are deterministic. Define*

$$\mathit{regr}_o^{nd}(\phi) = \mathit{regr}_{\langle c,e_1\rangle}(\phi) \wedge \cdots \wedge \mathit{regr}_{\langle c,e_n\rangle}(\phi).$$

Theorem 5.2 *Let ϕ be a formula over A, o an operator over A, and S the set of all states over A. Then $\{s \in S | s \models \mathit{regr}_o^{nd}(\phi)\} = \mathit{spreimg}_o(\{s \in S | s \models \phi\})$.*

Proof: Let $o = \langle c, (e_1 | \cdots | e_n) \rangle$.

$\{s \in S | s \models \mathit{regr}_o^{nd}(\phi)\}$
$= \{s \in S | s \models \mathit{regr}_{\langle c,e_1\rangle}(\phi) \wedge \cdots \wedge \mathit{regr}_{\langle c,e_n\rangle}(\phi)\}$
$= \{s \in S | s \models \mathit{regr}_{\langle c,e_1\rangle}(\phi), \ldots, s \models \mathit{regr}_{\langle c,e_n\rangle}(\phi)\}$
$= \{s \in S | \mathit{app}_{\langle c,e_1\rangle}(s) \models \phi, \ldots, \mathit{app}_{\langle c,e_n\rangle}(s) \models \phi\}$ T3.7
$= \{s \in S | s' \models \phi$ for all $s' \in \mathit{img}_o(s)$ and there is $s' \models \phi$ with $sos'\}$
$= \mathit{spreimg}_o(\{s \in S | s \models \phi\})$

The second last equality is because $\mathit{img}_o(s) = \{\mathit{app}_{\langle c,e_1\rangle}(s), \ldots, \mathit{app}_{\langle c,e_n\rangle}(s)\}$. □

Example 5.3 Let $o = \langle d, (b | \neg c) \rangle$. Then

$$\begin{aligned} \mathit{regr}_o^{nd}(b \leftrightarrow c) &= \mathit{regr}_{\langle d,b\rangle}(b \leftrightarrow c) \wedge \mathit{regr}_{\langle d,\neg c\rangle}(b \leftrightarrow c) \\ &= (d \wedge (\top \leftrightarrow c)) \wedge (d \wedge (b \leftrightarrow \bot)) \\ &\equiv d \wedge c \wedge \neg b. \end{aligned}$$

■

5.1.2 Translation of nondeterministic operators into propositional logic

In Section 3.6.2 we gave a translation of deterministic operators into the propositional logic. In this section we extend this translation to nondeterministic operators.

We define for effects e the sets $\mathit{changes}(e)$ of state variables that are possibly changed by e, or in other words, the set of state variables occurring in an atomic effect in e.

$$\begin{aligned} \mathit{changes}(a) &= \{a\} \\ \mathit{changes}(\neg a) &= \{a\} \\ \mathit{changes}(c \rhd e) &= \mathit{changes}(e) \\ \mathit{changes}(e_1 \wedge \cdots \wedge e_n) &= \mathit{changes}(e_1) \cup \cdots \cup \mathit{changes}(e_n) \\ \mathit{changes}(e_1 | \cdots | e_n) &= \mathit{changes}(e_1) \cup \cdots \cup \mathit{changes}(e_n) \end{aligned}$$

We make the following assumption to simplify the translation.

Assumption 5.4 *Let $a \in A$ be a state variable. Let $e_1 \wedge \cdots \wedge e_n$ occur in the effect of an operator. If $e_1, \ldots, e_n$ are not all deterministic, then a or $\neg a$ may occur as an atomic effect in at most one of $e_1, \ldots, e_n$.*

This assumption rules out effects like $(a|b) \wedge (\neg a|c)$ that may make a simultaneously true and false. It also rules out effects like $((d \rhd a)|b) \wedge ((\neg d \rhd \neg a)|c)$ that are well-defined and could be translated into the propositional logic. However, the additional complexity outweighs the benefit of allowing them. Effects can often easily be transformed by the equivalences in Table 2.2 to satisfy Assumption 5.4: $((d \rhd a)|b) \wedge ((\neg d \rhd \neg a)|c)$ is equivalent to $((d \rhd a) \wedge (\neg d \rhd \neg a))|((d \rhd a) \wedge c)|(b \wedge (\neg d \rhd \neg a))|(b \wedge c)$.

The problem in the translation that does not show up with deterministic operators is that for nondeterministic choices $e_1|\cdots|e_n$ the formula for each e_i has to express the changes for exactly the same set of state variables. This set B is given as a parameter to the translation function. The set B has to include all state variables possibly changed by the effect.

$$\begin{array}{rcl}
\tau_B^{nd}(e) & = & \tau_B(e) \text{ when } e \text{ is deterministic} \\
\tau_B^{nd}(e_1|\cdots|e_n) & = & \tau_B^{nd}(e_1) \vee \cdots \vee \tau_B^{nd}(e_n) \\
\tau_B^{nd}(e_1 \wedge \cdots \wedge e_n) & = & \tau_{B\setminus(B_2\cup\cdots\cup B_n)}^{nd}(e_1) \wedge \tau_{B_2}^{nd}(e_2) \wedge \cdots \wedge \tau_{B_n}^{nd}(e_n) \\
 & & \text{where } B_i = \mathit{changes}(e_i) \text{ for all } i \in \{2, \ldots, n\}
\end{array}$$

The first part of the translation $\tau_B^{nd}(e)$ for deterministic e is the translation of deterministic effects we presented in Section 3.6.2 restricted to state variables in B. The other two parts cover all nondeterministic effects in normal form. In the translation of $e_1 \wedge \cdots \wedge e_n$ all state variables that are not changed are handled in the translation of e_1. Assumption 5.4 guarantees that for each $\tau_B^{nd}(e)$ all state variables changed by e are in B.

Example 5.5 We translate the effect

$$e = (a|(d \rhd a)) \wedge (c|d)$$

into a propositional formula. The set of state variables is $A = \{a, b, c, d\}$.

$$\begin{array}{rcl}
\tau_{\{a,b,c,d\}}^{nd}(e) & = & \tau_{\{a,b\}}^{nd}(a|(d \rhd a)) \wedge \tau_{\{c,d\}}^{nd}(c|d) \\
 & = & (\tau_{\{a,b\}}^{nd}(a) \vee \tau_{\{a,b\}}^{nd}(d \rhd a)) \wedge (\tau_{\{c,d\}}^{nd}(c) \vee \tau_{\{c,d\}}^{nd}(d)) \\
 & = & ((a' \wedge (b \leftrightarrow b')) \vee (((a \vee d) \leftrightarrow a') \wedge (b \leftrightarrow b'))) \wedge \\
 & & ((c' \wedge (d \leftrightarrow d')) \vee ((c \leftrightarrow c') \wedge d'))
\end{array}$$

■

For expressing a state in terms of A' instead of A, or vice versa, we need to map a valuation of A to a corresponding valuation of A', or vice versa. for this purpose we define $s[A'/A] = \{\langle a', s(a)\rangle | a \in A\}$.

Definition 5.6 *Let A be a set of state variables. Let $o = \langle c, e\rangle$ be an operator over A in normal form. Define $\tau_A^{nd}(o) = c \wedge \tau_A^{nd}(e)$.*

Lemma 5.7 *Let o be an operator over a set A of state variables. Then*

$$\{v | v \textit{ is a valuation of } A \cup A', v \models \tau_A^{nd}(o)\} = \{s \cup s'[A'/A] | s, s' \in S, s' \in \mathit{img}_o(s)\}.$$

Proof: We show that there is a one-to-one match between valuations satisfying $\tau_A^{nd}(o)$ and pairs of states and their successor states.

For the proof from right to left assume that s and s' are states such that $s' \in \mathit{img}_o(s)$. Hence there is $E \in [e]_s$ such that s' is obtained from s by making literals in E true. Let $v = s \cup s'[A'/A]$. We show that $v \models \tau_A^{nd}(o)$. Let $o = \langle c, e\rangle$. Since $\mathit{img}_o(s)$ is non-empty, $s \models c$. It remains to show that $v \models \tau_A^{nd}(e)$.

Induction hypothesis: Let e be any effect over a set B of state variables, and s and s' states such for some $E \in [e]_s$ $s' \models E$ and $s(a) = s'(a)$ for every $a \in B$ such that $\{a, \neg a\} \cap E = \emptyset$. Then $s \cup s'[A'/A] \models \tau_B^{nd}(e)$.

Base case: e is a deterministic effect. There is only one $E \in [e]_s$. A proof similar to that of Lemma 3.42 shows that $s \cup s'[A'/A] \models \tau_B^{nd}(e)$.

Inductive case 1, $e = e_1 \wedge \cdots \wedge e_n$: By definition $\tau_B^{nd}(e_1 \wedge \cdots \wedge e_n) = \tau_{B\setminus(B_2\cup\cdots\cup B_n)}^{nd}(e_1) \wedge \tau_{B_2}^{nd}(e_2) \wedge \cdots \wedge \tau_{B_n}^{nd}(e_n)$ for $B_i = \mathit{changes}(e_i), i \in \{2, \ldots, n\}$. Let E be any member of $[e]_s$ and s' a state such that $s' \models E$ and $s(a) = s'(a)$ for every $a \in B$ such that $\{a, \neg a\} \cap E = \emptyset$. By definition of $[e]_s$ we have $E = E_1 \cup \cdots \cup E_n$ for some $E_i \in [e_i]_s$ for every $i \in \{1, \ldots, n\}$. The assumptions of the induction hypothesis hold for every e_i and $B_i, i \in \{2, \ldots, n\}$:

1. $s' \models E_i$ because $E_i \subseteq E$.
2. By Assumption 5.4 $s(a) = s'(a)$ for every $a \in B_i$ such that $\{a, \neg a\} \cap E_i = \emptyset$.

Similarly for e_1 and $B\setminus(B_2 \cup \cdots \cup B_n)$. Hence $s \cup s'[A'/A] \models \tau_{B_i}^{nd}(e_i)$ for all $i \in \{2, \ldots, n\}$ and $s \cup s'[A'/A] \models \tau_{B\setminus(B_2\cup\cdots\cup B_n)}^{nd}(e_i)$, and therefore $s \cup s'[A'/A] \models \tau_B^{nd}(e)$.

Inductive case 2, $e = e_1|\cdots|e_n$: By definition $\tau_B^{nd}(e_1|\cdots|e_n) = \tau_B^{nd}(e_1) \vee \cdots \vee \tau_B^{nd}(e_n)$. By definition $[e_1|\cdots|e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$. Hence $E \in [e_i]_s$ for some $i \in \{1, \ldots, n\}$. Hence the assumptions of the induction hypothesis hold for at least one $e_i, i \in \{1, \ldots, n\}$ and we get $s \cup s'[A'/A] \models \tau_B^{nd}(e_i)$. As $\tau_B^{nd}(e_i)$ is one of the disjuncts of $\tau_B^{nd}(e)$ finally $s \cup s'[A'/A] \models \tau_B^{nd}(e)$.

For the proof from left to right assume that $v \models \tau_B^{nd}(e)$ for $v = s \cup s'[A'/A]$. We prove by structural induction that the changes from s to s' correspond to $[e]_s$.

Induction hypothesis: Let e be any effect, B a set of state variables that includes those occurring in e, and s and s' states such that $v \models \tau_B^{nd}(e)$ where $v = s \cup s'[A'/A]$. Then there is $E \in [e]_s$ such that $s \models E$ and $s(a) = s'(a)$ for all $a \in B$ such that $\{a, \neg a\} \cap E = \emptyset$.

Base case: e is a deterministic effect. There is only one $E \in [e]_s$. A proof similar to that of Lemma 3.42 shows that the changes between s and s' for $a \in B$ correspond to E.

Inductive case 1, $e = e_1 \wedge \cdots \wedge e_n$: By definition $[e]_s = \{E_1 \cup \cdots \cup E_n | E_1 \in [e_1]_s, \ldots, E_n \in [e_n]_s\}$, and by Assumption 5.4 sets of the state variables occurring in $e_1, \ldots, e_n$ are disjoint. By definition $\tau_B^{nd}(e_1 \wedge \cdots \wedge e_n) = \tau_{B\setminus(B_2\cup\cdots\cup B_n)}^{nd}(e_1) \wedge \tau_{B_2}^{nd}(e_2) \wedge \cdots \wedge \tau_{B_n}^{nd}(e_n)$ for $B_i = \mathit{changes}(e_i), i \in \{2, \ldots, n\}$. The induction hypothesis for e and all $a \in B$ is directly by the induction hypothesis for all $a \in B = (B\setminus(B_2 \cup \cdots \cup B_n)) \cup B_2 \cup \cdots \cup B_n$ because $v \models \tau_{B\setminus(B_2\cup\cdots\cup B_n)}^{nd}(e_1) \wedge \tau_{B_2}^{nd}(e_2) \wedge \cdots \wedge \tau_{B_n}^{nd}(e_n)$.

Inductive case 2, $e = e_1|\cdots|e_n$: By definition $[e_1|\cdots|e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$. By definition $\tau_B^{nd}(e_1|\cdots|e_n) = \tau_B^{nd}(e_1) \vee \cdots \vee \tau_B^{nd}(e_n)$. Because $v \models \tau_B^{nd}(e_1|\cdots|e_n)$, $v \models \tau_B^{nd}(e_i)$ for some $i \in \{1, \ldots, n\}$. By the induction hypothesis there is $E \in [e_i]_s$ with the given property. We get the induction hypothesis for e because $[e_i]_s \subseteq [e]_s$ and hence also $E \in [e]_s$.

Therefore s' is obtained from s by making some literals in $E \in [e]_s$ true and retaining the values of state variables not mentioned in E, and $s' \in \mathit{img}_o(s)$. □

5.2 Computing with transition relations as formulae

As discussed in Section 2.3, formulae are a representation of sets of states. In this section we show how operations on transition relations have a counterpart as operations on formulae that represent transition relations. These techniques have first been used in connection of symbolic model-checking [Burch *et al.*, 1994; Clarke *et al.*, 1994].

Most implementations of the techniques in this section are based on binary decision diagrams (BDDs) [Bryant, 1992], a representation (essentially a normal form) of propositional formulae with useful computational properties, but the techniques are applicable to other representations of propositional formulae as well.

5.2.1 Existential and universal abstraction

The most important operations performed on transition relations represented as propositional formulae are based on *existential abstraction* and *universal abstraction*.

Definition 5.8 Existential abstraction *of a formula ϕ with respect to an atomic proposition a is the formula*

$$\exists a.\phi = \phi[\top/a] \vee \phi[\bot/a].$$

Universal abstraction is defined analogously by using conjunction instead of disjunction.

Definition 5.9 Universal abstraction *of a formula ϕ with respect to an atomic proposition a is the formula*

$$\forall a.\phi = \phi[\top/a] \wedge \phi[\bot/a].$$

Existential and universal abstraction of ϕ with respect to a *set of atomic propositions* is defined in the obvious way: for $B = \{b_1, \ldots, b_n\}$ such that B is a subset of the propositional variables occurring in ϕ define

$$\begin{aligned} \exists B.\phi &= \exists b_1.(\exists b_2.(\ldots \exists b_n.\phi \ldots)) \\ \forall B.\phi &= \forall b_1.(\forall b_2.(\ldots \forall b_n.\phi \ldots)). \end{aligned}$$

In the resulting formulae there are no occurrences of variables in B.

Let ϕ be a formula over A. Then $\exists A.\phi$ is a formula that consists of the constants $\top$ and $\bot$ and the logical connectives only. The truth-value of this formula is independent of the valuation of A, that is, its value is the same for all valuations.

The following lemma expresses the important properties of existential and universal abstraction. When we write $v \cup v'$ for a pair of valuations we view valuations v as binary relations, that is, sets of pairs such that $\{(a, b), (a, c)\} \not\subseteq v$ for any a, b and c such that $b \neq c$.

Lemma 5.10 *Let ϕ be a formula over $A \cup A'$ and v' a valuation of A'. Then*

1. *$v' \models \exists A.\phi$ if and only if $(v \cup v') \models \phi$ for at least one valuation v of A, and*
2. *$v' \models \forall A.\phi$ if and only if $(v \cup v') \models \phi$ for all valuations v of A.*

Proof: We prove the statements by induction on the cardinality of A. We only give the proof for $\exists$. The proof for $\forall$ is analogous to that for $\exists$.

Base case $|A| = 0$: There is only one valuation $v = \emptyset$ of the empty set $A = \emptyset$. When there is nothing to abstract we have $\exists\emptyset.\phi = \phi$. Hence trivially $v' \models \exists\emptyset.\phi$ if and only if $(v \cup \emptyset) \models \phi$.

matrices	formulas	sets of states
vector $V_{1\times n}$	formula over A	set of states
matrix $M_{n\times n}$	formula over $A \cup A'$	transition relation
$V_{1\times n} + V'_{1\times n}$	$\phi_1 \vee \phi_2$	set union
	$\phi_1 \wedge \phi_2$	set intersection
$M_{n\times n} \times N_{n\times n}$	$\exists A'.(\tau_A^{nd}(o) \wedge \tau_A^{nd}(o')[A''/A', A'/A])[A'/A'']$	sequential composition $o \circ o'$
$V_{1\times n} \times M_{n\times n}$	$(\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A']$	$\mathit{img}_o(T)$
$M_{n\times n} \times V_{n\times 1}$	$\exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A])$	$\mathit{preimg}_o(T)$
	$\forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o)$	$\mathit{spreimg}_o(T)$

Table 5.1: Correspondence between matrix operations, Boolean operations and set-theoretic/relational operations. Above $T = \{s \in S | s \models \phi\}$, M is the matrix corresponding to $\tau_A^{nd}(o)$ and N is the matrix corresponding to o'.

Inductive case $|A| \geq 1$: Take any $a \in A$. $v' \models \exists A.\phi$ if and only if $v' \models \exists A\backslash\{a\}.(\phi[\top/a] \vee \phi[\bot/a])$ by the definition of $\exists a.\phi$. By the induction hypothesis $v' \models \exists A\backslash\{a\}.(\phi[\top/a] \vee \phi[\bot/a])$ if and only if $(v_0 \cup v') \models \phi[\top/a] \vee \phi[\bot/a]$ for at least one valuation v_0 of $A\backslash\{a\}$. Since the formula $\phi[\top/a] \vee \phi[\bot/a]$ represents both possible valuations of a in ϕ, the last statement is equivalent to $(v \cup v') \models \phi$ for at least one valuation v of A. □

5.2.2 Images and preimages as formula manipulation

Let $A = \{a_1, \ldots, a_n\}$, $A' = \{a'_1, \ldots, a'_n\}$ and $A'' = \{a''_1, \ldots, a''_n\}$. Let ϕ_1 be a formula over $A \cup A'$ and ϕ_2 be a formula over $A' \cup A''$. The formulae can be viewed as representations of $2^n \times 2^n$ matrices or as transition relations over a state space of size 2^n.

The product matrix of ϕ_1 and ϕ_2 is represented by a the following formula over $A \cup A''$.

$$\exists A'.\phi_1 \wedge \phi_2$$

Example 5.11 Let $\phi_1 = a \leftrightarrow \neg a'$ and $\phi_2 = a' \leftrightarrow a''$ represent two actions, reversing the truth-value of a and doing nothing. The sequential composition of these actions is

$$\begin{aligned} \exists a'.\phi_1 \wedge \phi_2 &= ((a \leftrightarrow \neg\top) \wedge (\top \leftrightarrow a'')) \vee ((a \leftrightarrow \neg\bot) \wedge (\bot \leftrightarrow a'')) \\ &\equiv ((a \leftrightarrow \bot) \wedge (\top \leftrightarrow a'')) \vee ((a \leftrightarrow \top) \wedge (\bot \leftrightarrow a'')) \\ &\equiv a \leftrightarrow \neg a''. \end{aligned}$$

■

This idea can be used for computing the images, preimages and strong preimages of operators and sets of states in terms of formula manipulation by existential and universal abstraction. Table 5.1 outlines a number of connections between operations on vectors and matrices, on propositional formulae, and on sets and relations. For transition relations we use valuations of $A \cup A'$ for representing pairs for states and for states we use valuations of A.

Lemma 5.12 *Let ϕ be a formula over A and v a valuation of A. Then $v \models \phi$ if and only if $v[A'/A] \models \phi[A'/A]$, and $(\phi[A'/A])[A/A'] = \phi$.*

Definition 5.13 *Let o be an operator and ϕ a formula. Define*

$$\begin{aligned} \mathit{img}_o(\phi) &= (\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A'] \\ \mathit{preimg}_o(\phi) &= \exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A]) \\ \mathit{spreimg}_o(\phi) &= \forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o). \end{aligned}$$

Theorem 5.14 *Let $T = \{s \in S | s \models \phi\}$. Then $\{s \in S | s \models \mathit{img}_o(\phi)\} = \{s \in S | s \models (\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A']\} = \mathit{img}_o(T)$.*

Proof: $s' \models (\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A']$ □
iff $s'[A'/A] \models \exists A.(\phi \wedge \tau_A^{nd}(o))$ L5.12
iff there is valuation s of A such that $(s \cup s'[A'/A]) \models \phi \wedge \tau_A^{nd}(o)$ L5.10
iff there is valuation s of A such that $s \models \phi$ and $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$
iff there is $s \in T$ such that $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$
iff there is $s \in T$ such that $s' \in \mathit{img}_o(s)$ L5.7
iff $s' \in \mathit{img}_o(T)$.

Theorem 5.15 *Let $T = \{s \in S | s \models \phi\}$. Then $\{s \in S | s \models \mathit{preimg}_o(\phi)\} = \{s \in S | s \models \exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A])\} = \mathit{preimg}_o(T)$.*

Proof: $s \models \exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A])$
iff there is $s'_0 : A' \rightarrow \{0,1\}$ such that $(s \cup s'_0) \models \tau_A^{nd}(o) \wedge \phi[A'/A]$
iff there is $s'_0 : A' \rightarrow \{0,1\}$ such that $s'_0 \models \phi[A'/A]$ and $(s \cup s'_0) \models \tau_A^{nd}(o)$ L5.10
iff there is $s' : A \rightarrow \{0,1\}$ such that $s' \models \phi$ and $(s \cup s'_0) \models \tau_A^{nd}(o)$ L5.12
iff there is $s' \in T$ such that $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$
iff there is $s' \in T$ such that $s' \in \mathit{img}_o(s)$ L5.7
iff there is $s' \in T$ such that $s \in \mathit{preimg}_o(s')$ (5) of L2.2
iff $s \in \mathit{preimg}_o(T)$.
Above we define $s' = s'_0[A/A']$ (and hence $s'_0 = s'[A'/A]$.) □

Theorem 5.16 *Let $T = \{s \in S | s \models \phi\}$. Then $\{s \in S | s \models \mathit{spreimg}_o(\phi)\} = \{s \in S | s \models \forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o)\} = \mathit{spreimg}_o(T)$.*

Proof:
$s \models \forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o)$
iff $s \models \forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A])$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $(s \cup s'_0) \models \tau_A^{nd}(o) \rightarrow \phi[A'/A]$ for all $s'_0 : A' \rightarrow \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L5.10
iff $(s \cup s'_0) \not\models \tau_A^{nd}(o)$ or $s'_0 \models \phi[A'/A]$ for all $s'_0 : A' \rightarrow \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $(s \cup s'[A'/A]) \not\models \tau_A^{nd}(o)$ or $s' \models \phi$ for all $s' : A \rightarrow \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L5.12
iff $s' \notin \mathit{img}_o(s)$ or $s' \models \phi$ for all $s' : A \rightarrow \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L5.7
iff $s' \in \mathit{img}_o(s)$ implies $s' \models \phi$ for all $s' : A \rightarrow \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $\mathit{img}_o(s) \subseteq T$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' : A \rightarrow \{0,1\}$ with $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$ L5.10
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' : A \rightarrow \{0,1\}$ with $s' \in \mathit{img}_o(s)$ L5.7
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' \in T$ with $s' \in \mathit{img}_o(s)$
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' \in T$ with sos'
iff $s \in \mathit{spreimg}_o(T)$.

Above we define $s' = s'_0[A/A']$ (and hence $s'_0 = s'[A'/A]$.) □

Corollary 5.17 *Let $o = \langle c, (e_1| \cdots |e_n)\rangle$ be an operator such that all e_i are deterministic. The formula $spreimg_o(\phi)$ is logically equivalent to $regr_o^{nd}(\phi)$ as given in Definition 5.1.*

Proof: By Theorems 5.2 and 5.16 $\{s \in S | s \models regr_o(\phi)\} = spreimg_o(\{s \in S | s \models \phi\}) = \{s \in S | s \models spreimg_o(\phi)\}$. □

Example 5.18 Let $o = \langle c, a \wedge (a \rhd b)\rangle$. Then

$$regr_o^{nd}(a \wedge b) = c \wedge (\top \wedge (b \vee a)) \equiv c \wedge (b \vee a).$$

The transition relation of o is represented by

$$\tau_A^{nd}(o) = c \wedge a' \wedge ((b \vee a) \leftrightarrow b') \wedge (c \leftrightarrow c').$$

The preimage of $a \wedge b$ with respect to o is represented by

$$\begin{array}{rcl}
\exists a'b'c'.((a' \wedge b') \wedge \tau_A^{nd}(o)) & \equiv & \exists a'b'c'.((a' \wedge b') \wedge c \wedge a' \wedge ((b \vee a) \leftrightarrow b') \wedge (c \leftrightarrow c')) \\
& \equiv & \exists a'b'c'.(a' \wedge b' \wedge c \wedge (b \vee a) \wedge c') \\
& \equiv & \exists b'c'.(b' \wedge c \wedge (b \vee a) \wedge c') \\
& \equiv & \exists c'.(c \wedge (b \vee a) \wedge c') \\
& \equiv & c \wedge (b \vee a)
\end{array}$$

■

Hence regression for nondeterministic operators (Definition 5.1) can be viewed as a specialized method for computing preimages of sets of states represented as formulae.

Many algorithms include the computation of the union of images or preimages with respect to all operators, for example $\bigcup_{o \in O} img_o(T)$, or in terms of formulae, $\bigvee_{o \in O} img_o(\phi)$ where $T = \{s \in S | s \models \phi\}$. A technique used by many implementations of such algorithms is the following. Instead of computing the images or preimages one operator at a time, construct a combined transition relation for all operators. For an illustration of the technique, consider $img_{o_1}(\phi) \vee img_{o_2}(\phi)$ that represents the union of state sets represented by $img_{o_1}(\phi)$ and $img_{o_2}(\phi)$. By definition

$$img_{o_1}(\phi) \vee img_{o_2}(\phi) = (\exists A.(\phi \wedge \tau_A^{nd}(o_1)))[A/A'] \vee (\exists A.(\phi \wedge \tau_A^{nd}(o_2)))[A/A'].$$

Since substitution commutes with disjunction we have

$$img_{o_1}(\phi) \vee img_{o_2}(\phi) = (\exists A.(\phi \wedge \tau_A^{nd}(o_1))) \vee (\exists A.(\phi \wedge \tau_A^{nd}(o_2)))[A/A'].$$

Since existential abstraction commutes with disjunction we have

$$img_{o_1}(\phi) \vee img_{o_2}(\phi) = (\exists A.((\phi \wedge \tau_A^{nd}(o_1)) \vee (\phi \wedge \tau_A^{nd}(o_2))))[A/A'].$$

By logical equivalence finally

$$img_{o_1}(\phi) \vee img_{o_2}(\phi) = (\exists A.(\phi \wedge (\tau_A^{nd}(o_1) \vee \tau_A^{nd}(o_2))))[A/A'].$$

Hence an alternative way of computing the union of images $\bigvee_{o \in O} \mathit{img}_o(\phi)$ is to first form the disjunction $\bigvee_{o \in O} \tau_A^{nd}(o)$ and then conjoin the formula with ϕ and only once existentially abstract the propositional variables in A. This may reduce the amount of computation because existential abstraction is in general expensive and it may be possible to simplify the formulae $\bigvee_{o \in O} \tau_A^{nd}(o)$ before existential abstraction.

The definitions of $\mathit{preimg}_o(\phi)$ and $\mathit{spreimg}_o(\phi)$ allow using $\bigvee_{o \in O} \tau_A^{nd}(o)$ in the same way.

Note that defining progression for arbitrary formulae (sets of states) seems to require the explicit use of existential abstraction with potential exponential increase in formula size. A simple syntactic definition of progression similar to that of regression does not seem to be possible because the value of a state variable in a given state cannot be stated in terms of the values of the state variables in the successor state. This is because of the asymmetry of deterministic actions: the current state and an operator determine the successor state uniquely but the successor state and the operator do not determine the current state uniquely. In other words, the changes that take place are a function of the current state, but not a function of the successor state. Taking an action erases the information that determines which changes take place between two states. This information is visible in the predecessor state but not in the successor state.

5.3 Planning without observability

In this section we present algorithmic techniques for planning with nondeterministic operators and no observability.

The first approach, presented in Section 5.3.1, is a generalization of the *planning by satisfiability* approach by Kautz and Selman [1992; 1996] to nondeterministic problems. Even under the restriction to polynomially long plans the nondeterministic planning problem does not belong to the complexity class NP. Therefore it cannot in general be practically translated into the satisfiability problem of the classical propositional logic. Instead, we reduce the problem to the evaluation of quantified Boolean formulae QBF.

The second approach applies heuristic search to search in the space of belief states. Our contribution in Section 5.3.2 is the introduction of a family of distance heuristics. This work follows the approach presented by Bonet and Geffner [2001] for the deterministic planning problem.

5.3.1 Planning by evaluation of QBF

The techniques presented in Sections 3.6 and 3.8 can be extended to nondeterministic operators. The notion of parallel application of operators and partially ordered plans can be generalized to nondeterministic operators.

Let T be a set of operators and s a state such that $s \models c$ for every $\langle c, e \rangle \in T$ and $E_1 \cup \cdots \cup E_n$ is consistent for for any $E_i \in [e_i]_s, i \in \{1, \ldots, n\}$ and $T = \{\langle c_1, e_1 \rangle, \ldots \langle c_n, e_n \rangle\}$. Then define $\mathit{img}_T(s)$ as the set of states s' that are obtained from s by making $E_1 \cup \cdots \cup E_n$ true in s where $E_i \in [e_i]_s$ for every $i \in \{1, \ldots, n\}$. We also use the notation sTs' for $s' \in \mathit{img}_T(s)$ and $\mathit{img}_T(S) = \bigcup_{s \in S} \mathit{img}_T(s)$.

In Section 5.1.2 we showed how nondeterministic operators can be translated into formulae in the propositional logic. This translation is not sufficient for reasoning about actions and plans in a setting with more than one agent. This is because the formulae $\tau_A^{nd}(o_1) \vee \cdots \vee \tau_A^{nd}(o_n)$ do not distinguish between the choice of operator in $\{o_1, \ldots, o_n\}$ and the nondeterministic effects (the opponent) of each operator, even though the former is controllable and the latter is not.

In nondeterministic planning in general we have to treat the controllable and uncontrollable choices differently. We cannot do this practically in the propositional logic but by using quantified Boolean formulae (QBF) we can. For the QBF representation of nondeterministic operators we universally quantify over all uncontrollable eventualities (nondeterminism) and existentially quantify over controllable eventualities (the choice of operators).

We need to universally quantify over all the nondeterministic choices because for every choice the remaining operators in the plan must lead to a goal state. This is achieved by associating with every atomic effect a formula that is true if and only if that effect is executed, similarly to functions $EPC_l(e)$ in Definition 3.1, so that for l to become true the universally quantified auxiliary variables that represent nondeterminism have to have values corresponding to an effect that makes l true.

The operators are assumed to be in normal form. For simplicity of presentation we further transform nondeterministic choices $e_1|\cdots|e_n$ so that only binary choices exist. For example $a|b|c|d$ is replaced by $(a|b)|(c|d)$. Each binary choice can be encoded in terms of one auxiliary variable.

The condition for the atomic effect l to be executed when e is executed is $EPC_l^{nd}(e,\sigma)$. The sequence σ of integers is used for deriving unique names for auxiliary variables in $EPC_l^{nd}(e,\sigma)$. The sequences correspond to paths in the tree formed by nested nondeterministic choices and conjunctions.

$$\begin{array}{lcl} EPC_l^{nd}(e,\sigma) & = & EPC_l(e) \text{ if } e \text{ is deterministic} \\ EPC_l^{nd}(e_1|e_2,\sigma) & = & (x_\sigma \wedge EPC_l^{nd}(e_1,\sigma 1)) \vee (\neg x_\sigma \wedge EPC_l^{nd}(e_2,\sigma 1)) \\ EPC_l^{nd}(e_1 \wedge \cdots \wedge e_n,\sigma) & = & EPC_l^{nd}(e_1,\sigma 1) \vee \cdots \vee EPC_l^{nd}(e_n,\sigma n) \end{array}$$

The translation of nondeterministic operators into the propositional logic is similar to the translation for deterministic operators given in Section 3.8. Nondeterminism is encoded by making the effects conditional on the values of the auxiliary variables x_σ. Different valuations of these auxiliary variables correspond to different nondeterministic effects.

The following frame axioms express the conditions under which state variables $a \in A$ may change from true to false and from false to true. Let $e_1,\ldots,e_n$ be the effects of $o_1,\ldots,o_n$ respectively. Each operator $o \in O$ has a unique integer index $\Omega(o)$.

$$\begin{array}{l} (a \wedge \neg a') \rightarrow ((o_1 \wedge EPC_{\neg a}^{nd}(e_1,\Omega(o_1))) \vee \cdots \vee (o_n \wedge EPC_{\neg a}^{nd}(e_n,\Omega(o_n)))) \\ (\neg a \wedge a') \rightarrow ((o_1 \wedge EPC_a^{nd}(e_1,\Omega(o_1))) \vee \cdots \vee (o_n \wedge EPC_a^{nd}(e_n,\Omega(o_n)))) \end{array}$$

For $o = \langle c, e\rangle \in O$ there is a formula for describing values of state variables in the predecessor and successor states when the operator is applied.

$$\begin{array}{l} (o \rightarrow c) \wedge \\ \bigwedge_{a\in A}(o \wedge EPC_a^{nd}(e,\Omega(o)) \rightarrow a') \wedge \\ \bigwedge_{a\in A}(o \wedge EPC_{\neg a}^{nd}(e,\Omega(o)) \rightarrow \neg a') \end{array}$$

Example 5.19 Consider $o_1 = \langle \neg a, (b|(c \rhd d)) \wedge (a|c)\rangle$ and $o_2 = \langle \neg b, (((d \rhd b)|c)|a)\rangle$. The

application of these operators is described by the following formulae.

$$
\begin{array}{ll}
\neg(a \wedge \neg a') & (\neg a \wedge a') \rightarrow ((o_1 \wedge x_{12}) \vee (o_2 \wedge \neg x_2)) \\
\neg(b \wedge \neg b') & (\neg b \wedge b') \rightarrow ((o_1 \wedge x_{11}) \vee (o_2 \wedge x_2 \wedge x_{21} \wedge d)) \\
\neg(c \wedge \neg c') & (\neg c \wedge c') \rightarrow ((o_1 \wedge \neg x_{12}) \vee (o_2 \wedge x_2 \wedge \neg x_{21})) \\
\neg(d \wedge \neg d') & (\neg d \wedge d') \rightarrow (o_1 \wedge \neg x_{11} \wedge c) \\
o_1 \rightarrow \neg a & \\
(o_1 \wedge x_{12}) \rightarrow a' & (o_1 \wedge x_{11}) \rightarrow b' \\
(o_1 \wedge \neg x_{12}) \rightarrow c' & (o_1 \wedge \neg x_{11} \wedge c) \rightarrow d' \\
o_2 \rightarrow \neg b & \\
(o_2 \wedge \neg x_2) \rightarrow a' & (o_2 \wedge x_2 \wedge x_{21} \wedge d) \rightarrow b' \\
(o_2 \wedge x_2 \wedge \neg x_{21}) \rightarrow c' &
\end{array}
$$

■

Two operators o and o' may be applied in parallel only if they do not interfere. Hence we use formulae

$$\neg(o \wedge o')$$

for all operators o and o' that interfere and $o \neq o'$.

Let X be the set of auxiliary variables x_σ in all the above formulae. The conjunction of all the above formulae is denoted by

$$\mathcal{R}_3(A, A', O, X).$$

We use two lemmata for proving properties about these formulae and the translation of nondeterministic operators into the propositional logic.

Let $\Xi_\sigma(e)$ be the set of propositional variables $x_{\sigma'}$ in the translation of the effect e with a given σ. This is equal to the set of variables $x_{\sigma'}$ in formulae $EPC_a^{nd}(e, \sigma)$ and $EPC_{\neg a}^{nd}(e, \sigma))$ for all $a \in A$.

Definition 5.20 *Define the set of literals $[e]_s^{\sigma,v}$ which are the active effects of e when e is executed in state s and nondeterministic choices are determined by the valuation v of propositional variables in $\Xi_\sigma(e)$ as follows.*

$$
\begin{array}{rcl}
[e]_s^{\sigma,v} & = & [e]_s^{det} \text{ if } e \text{ is deterministic} \\
[e_1|e_2]_s^{\sigma,v} & = & \begin{cases} [e_1]_s^{\sigma 1,v} \text{ if } v(x_\sigma) = 1 \\ [e_2]_s^{\sigma 1,v} \text{ if } v(x_\sigma) = 0 \end{cases} \\
[e_1 \wedge \cdots \wedge e_n]_s^{\sigma,v} & = & [e_1]_s^{\sigma 1,v} \cup \cdots \cup [e_n]_s^{\sigma n,v}
\end{array}
$$

Lemma 5.21 *Let s be a state and $\{v_1, \ldots, v_n\}$ all valuations of $\Xi_\sigma(e)$. Then $\bigcup_{1 \leq i \leq n} [e]_s^{\sigma,v_i} = [e]_s$.*

Lemma 5.22 *Let O and $T \subseteq O$ be sets of operators, s and s' states, v_x a valuation of $X = \bigcup_{\langle c,e \rangle \in O} \Xi_{\Omega(\langle c,e \rangle)}(e)$, and v_o a valuation of O such that $v_o(o) = 1$ iff $o \in T$.*

Then $s \cup s'[A'/A] \cup v_o \cup v_x \models \mathcal{R}_3(A, A', O, X)$ if and only if

1. *$s \models a$ iff $s' \models a$ for all $a \in A$ such that $\{a, \neg a\} \cap \bigcup_{\langle c,e \rangle \in T} [e]_s^{\Omega(\langle c,e \rangle), v_x} = \emptyset$,*

2. $s' \models \bigcup_{\langle c,e\rangle \in T} [e]_s^{\Omega(\langle c,e\rangle),v_x}$, *and*

3. $s \models c$ *for all* $\langle c, e\rangle \in T$.

The number of auxiliary variables x_σ can be reduced when two operators o and o' interfere. Since they cannot be applied simultaneously the same auxiliary variables can control the nondeterminism in both operators. To share the variables rename the ones occurring in the formulae for one of the operators so that the variables needed for o is a subset of those for o' or vice versa. Having as small a number of auxiliary variables as possible may be important for the efficiency for algorithms evaluating QBF and testing propositional satisfiability.

The formulae $\mathcal{R}_3(A, A', O, X)$ can be used for plan search with algorithms that evaluate QBF as well as for testing by a satisfiability algorithm whether a conditional plan (with full, partial or no observability) that allows several operators simultaneously indeed is a valid plan.

In deterministic planning in propositional logic (Section 3.6) the problem is to find a sequence of operators so that a goal state is reached when the operators are applied starting in the initial state. When there are several initial states, the operators are nondeterministic and it is not possible to use observations during plan execution for selecting operators, the problem is to find an operator sequence so that a goal state is reached in all possible executions of the operator sequence. There may be several executions because there may be several initial states and the operators may be nondeterministic. Expressing the quantification over all possible executions cannot be concisely expressed in the propositional logic. This is the reason why quantified Boolean formulae are used instead.

The existence of an n-step partially-ordered plan that reaches a state satisfying G from any state satisfying the formula I can be tested by evaluating the QBF Φ_n^{qpar} defined as

$$\exists V_{plan} \forall V_{nd} \exists V_{exec}$$
$$I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \mathcal{R}_3(A^1, A^2, O^1, X^1) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1}) \wedge G^n).$$

Here $V_{plan} = O^0 \cup \cdots \cup O^{n-1}$, $V_{nd} = A^0 \cup X^0 \cup \cdots \cup X^{n-1}$ and $V_{exec} = A^1 \cup \cdots \cup A^n$. Define $\Phi_n^{qparM} = I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \mathcal{R}_3(A^1, A^2, O^1, X^1) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1}) \wedge G^n)$. The valuation of V_{plan} corresponds to a sequence of sets of operators. For a given valuation of V_{plan} any valuation of V_{nd} determines an execution of these operators. The valuation of V_{exec} is uniquely determined by the valuation of $V_{plan} \cup V_{nd}$.

The algorithms for evaluating QBF that extend the Davis-Putnam procedure traverse an and-or tree in which the and-nodes correspond to universally quantified variables and or-nodes correspond to existentially quantified variables. If the QBF is *true* then these algorithms return a valuation of the outermost existential variables. For a true Φ_n^{qpar} this valuation of V_{plan} corresponds to a plan that can be constructed like the plans in the deterministic case in Section 3.8.1.

Theorem 5.23 *The QBF Φ_n^{qpar} has value* true *if and only if there is a sequence $T_0, \ldots, T_{n-1}$ of sets of operators such that for every $i \in \{0, \ldots, n\}$ and every state sequence $s_0, \ldots, s_i$ such that*

1. $s_0 \models I$ *and*
2. $s_0 T_0 s_1 T_1 s_2 \cdots s_{i-1} T_{i-1} s_i$

T_i is applicable in s_i if $i < n$ and $s_i \models G$ if $i = n$.

Proof: We first prove the implication from left to right. Since Φ_n^{qpar} is true there is a valuation v_{plan} of $V_{plan} = O^0 \cup \cdots \cup O^{n-1}$ such that for all valuations v_{nd} of $V_{nd} = A^0 \cup X^0 \cup \cdots \cup X^{n-1}$ there is a valuation v_{exec} of $V_{exec} = A^1 \cup \cdots \cup A^n$ such that $v_{plan} \cup v_{nd} \cup v_{exec} \models I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1}) \wedge G^n)$.

Let $T_0, \ldots, T_{n-1}$ be the sequence of sets of operators such that for all $o \in O$ and $i \in \{0, \ldots, n-1\}$, $o \in T_i$ if and only if $v_{plan}(o^i) = 1$. We prove the right hand side of the theorem by induction on n.

Induction hypothesis: For every $s_0, \ldots, s_i$ such that $s_0 \models I$ and $s_0 T_0 s_1 T_1 s_2 \cdots s_{i-1} T_{i-1} s_i$:

1. T_i is applicable in s_i if $i < n$.
2. $s_i \models G$ if $i = n$.

Base case $i = 0$: Let s_0 be any state sequence such that $s_0 \models I$.

1. If $0 < n$ then we have to show that T_0 is applicable in s_0.

   Let $E = E_1 \cup \cdots \cup E_m$ for all $j \in \{1, \ldots, m\}$ and any $E_j \in [e_j]_{s_0}$, where $e_1, \ldots, e_m$ are respectively the effects of the operators $o_1, \ldots, o_m$ in T_0. Such sets E are the possible active effects of T_0.

   We have to show that E is consistent and the preconditions of operators in T_0 are true in s_0.

   By Lemma 5.21 there is a valuation v of X such that $E = \bigcup_{\langle c,e\rangle \in T_0} [e]_{s_0}^{\Omega(\langle c,e\rangle),v}$.

   Let v_{nd} be any valuation of V_{nd} such that $s_0[A^0/A] \subseteq v_{nd}$ and $v[X^0/X] \subseteq v_{nd}$. Since Φ_n^{qpar} is true there is a valuation of v_{exec} such that $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$.

   Since $v_{nd} \models I^0$ also $v_{plan} \cup v_{nd} \cup v_{exec} \models \mathcal{R}_3(A^0, A^1, O^0, X^0)$. Hence by Lemma 5.22 the preconditions of operators in T_0 are true in s_0 and $s_1 \models E$ where s_1 is the state such that $s_1(a) = v_{exec}(a^1)$ for all $a \in A$. Since E was chosen arbitrarily from the sets of possible sets of active effects of T_0 and it is consistent, T_0 is applicable in s_0.

2. If $n = 0$ then $V_{plan} = V_{exec} = \emptyset$ and $\forall V_{nd}(I^0 \rightarrow G^0)$ is true, and $v_{nd} \models G^0$ for every valuation v_{nd} of V_{nd} such that $v_{nd} \models I^0$.

Inductive case $i \geq 1$: Let $s_0, \ldots, s_i$ be any sequence such that $s_0 \models I$ and $s_0 T_0 s_1 \ldots s_{i-1} T_{i-1} s_i$.

1. If $i < n$ then we have to show that T_i is applicable in s_i.

   Let $E = E_1 \cup \cdots \cup E_m$ for all $j \in \{1, \ldots, m\}$ and any $E_j \in [e_j]_{s_i}$, where $e_1, \ldots, e_m$ are respectively the effects of the operators $o_1, \ldots, o_m$ in T_i. Such sets E are the possible active effects of T_i.

   We have to show that E is consistent and the preconditions of operators in T_i are true in s_i.

By Lemma 5.21 there is a valuation v of X such that $E = \bigcup_{\langle c,e\rangle \in T_i} [e]_{s_i}^{\Omega(\langle c,e\rangle),v}$.

Since by the induction hypothesis $s_j T_j s_{j+1}$ for all $j \in \{0,\ldots,i-1\}$, by Lemma 5.21 for every $j \in \{0,\ldots,i-1\}$ there is a valuation v_j^x of X such that $s_j[A/A^j] \cup s_{j+1}[A'/A^{j+1}] \cup v_o \cup v_j^x \models \mathcal{R}_3(A, A', O, X)$ where v_o assigns every $o \in O$ value 1 iff $o \in T_j$.

Let v_{nd} be any valuation of V_{nd} such that $s_0[A^0/A] \subseteq v_{nd}$ and $v[X^i/X] \subseteq v_{nd}$ and $v_j^x[X^j/X] \subseteq v_{nd}$ for all $j \in \{0,\ldots,i-1\}$.

Since Φ_n^{qpar} is true there is a valuation of v_{exec} such that $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$.

Since $v_{nd} \models I^0$ also $v_{plan} \cup v_{nd} \cup v_{exec} \models \mathcal{R}_3(A^i, A^{i+1}, O^i, X^i)$. Hence by Lemma 5.22 the preconditions of operators in T_i are true in s_i and $s_{i+1} \models E$ where s_{i+1} is a state such that $s_{i+1}(a) = v_{exec}(a^{i+1})$ for all $a \in A$. Since any E is consistent, T_i is applicable in s_i.

2. If $i = n$ we have to show that $s_n \models G$. Like in the proof for the previous case we construct valuations v_{nd} and v_{exec} matching the execution $s_0,\ldots,s_n$, and since $v_{plan} \cup v_{nd} \cup v_{exec} \models I^0 \rightarrow G^n$ we have $s_n \models G$.

Then we prove the implication from right to left. So there is sequence $T_0,\ldots,T_{n-1}$ for which all executions are defined and reach G.

We show that Φ_n^{qpar} is true: there is valuation v_{plan} of $V_{plan} = O^0 \cup \cdots \cup O^{n-1}$ such that for every valuation v_{nd} of $V_{nd} = A^0 \cup X^0 \cup \cdots \cup X^{n-1}$ there is a valuation v_{exec} of $V_{exec} = A^1 \cup \cdots \cup A^n$ such that $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$.

We define the valuation v_{plan} of V_{plan} by $o \in T_i$ iff $v_{plan}(o^i) = 1$ for every $o \in O$ and $i \in \{0,\ldots,n-1\}$.

Take any valuation v_{nd} of V_{nd}. Define the state s_0 by $s_0(a) = 1$ iff $v_{nd}(a^0) = 1$ for every $a \in A$.

If $s_0 \not\models I$ then $v_{nd} \not\models I^0$ and $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$ for any valuation v_{exec} of V_{exec}.

It remains to consider the case $s_0 \models I$.

Define for every $i \in \{1,\ldots,n\}$ sets E_i and states s_i as follows.

1. Let v_x^i be a valuation of X such that $v_x^i(x) = v_{nd}(x^{i-1})$ for every $x \in X$.

2. Let $E_i = \bigcup_{\langle c,e\rangle \in T_{i-1}} [e]_{s_{i-1}}^{\Omega(\langle c,e\rangle),v_x^i}$.

   We show below that this is the set of literals made true by T_{i-1} in s_{i-1}.

3. Define $s_i(a) = 1$ iff $a \in E_i$ or $s_{i-1}(a) = 1$ and $\neg a \notin E_i$, for every $a \in A$.

Let $v_{exec} = s_1[A^1/A] \cup \cdots \cup s_n[A^n/A]$.

Induction hypothesis: $v_{plan} \cup v_{nd} \cup s_1[A^1/A] \cup \cdots \cup s_i[A^i/A] \models I^0 \wedge \mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{i-1}, A^i, O^{i-1}, X^{i-1})$ and $s_j T_j s_{j+1}$ for all $j \in \{0,\ldots,i-1\}$.

Base case $i = 0$: Trivial because $v_{nd} \models I^0$.

Inductive case $i \geq 1$: Let $v_x \subseteq v_{nd}$ be the valuation of X^{i-1} determined by v_{nd} and let v_o be the valuation of O^{i-1} such that $v_o(o) = v_{plan}(o^{i-1})$ for every $o \in O$. By Lemma 5.22 $v_{plan} \cup v_{nd} \cup s_{i-1}[A^{i-1}/A] \cup s_i[A^i/A] \models \mathcal{R}_3(A^{i-1}, A^i, O^{i-1}, X^{i-1})$. This together with the claim of the induction hypothesis for $i-1$ establishes the first part of the claim of the hypothesis for i. By Lemma 5.21 the set E_i is one of the possible sets of active effects of T_{i-1} in s_{i-1}. Hence $s_{i-1} T_{i-1} s_i$. This finishes the induction proof.

Hence $v_{plan} \cup v_{nd} \cup v_{exec} \models I^0 \wedge \mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1})$, and $v_{exec} \models G^n$ because $s_n \models G$ by assumption and $s_n[A^n/A] \subseteq v_{exec}$. □

5.3.2 Distance heuristics for the belief space

We present a general framework for studying heuristics for planning in the belief space. Earlier work has focused on giving implementations of heuristics that work well on benchmarks, without studying them at a more analytical level. Existing heuristics have evaluated belief states in terms of their cardinality or have used distance heuristics directly based on the distances in the underlying state space. Neither of these types of heuristics is very widely applicable: often goal belief state is not approached through a sequence of belief states with a decreasing cardinality, and distances in the state space ignore the main implications of partial observability.

To remedy these problems we present a family of admissible, increasingly accurate distance heuristics for planning in the belief space, parameterized by an integer n. We show that the family of heuristics is theoretically robust: it includes the simplest heuristic based on the state space as a special case and as a limit the exact distances in the belief space.

Following the lead in classical planning [Bonet and Geffner, 2001], also restricted types of planning in the belief space, most notably the planning problem without any observability at all (sometimes known as conformant planning), has also been represented as a heuristic search problem [Bonet and Geffner, 2000]. However, the implementation of heuristic search in the belief space is more complex than in classical planning because of the difficulty of deriving good heuristics. First works on the topic have used distances in the state space [Bonet and Geffner, 2000] and cardinalities of the belief states [Bertoli *et al.*, 2001a]. On some types of problems these heuristics work well, but not on all, and the two proposed approaches have orthogonal strengths.

Many problems cannot be solved by blindly taking actions that reduce the cardinality of the current belief state: the cardinality of the belief state may stay the same or increase during plan execution, and hence the decrease in cardinality is not characteristic to belief space planning in general.

Similarly, distances in the state space completely ignore the most distinctive aspect of planning with partial observability: the same action must be used in two states if the states are not observationally distinguishable. A given (optimal) plan for an unobservable problem may increase the actual current state-space distance to the goal states (on a given execution) when the distance in the belief-space monotonically decreases, and vice versa. Hence, the state space distances may yield wildly misleading estimates of the distances in the corresponding belief space.

To achieve more efficient planning it is necessary to develop belief space heuristics that combine the strengths of existing heuristics based on cardinalities and distances in the state space. In this work we present such a family of heuristics, parameterized by a natural number n. The accuracy of the distance estimates improves as n grows. As the special case $n = 1$ we have a heuristic based on distances in the state space, similar to ones used in earlier work. When the cardinality of the state space equals n, the distance estimates equal the actual distances in the belief space.

Belief states in the belief space are sets of states, that is, the belief space is the powerset of the state space. An operator o maps a belief state B to the belief state

$$\{s' \in S | s \in B, sos'\}$$

if o is applicable in every $s \in B$, equivalently, for every $s \in B$ there is $s' \in S$ such that sos',

where S is the set of all states.

A labeled graph $\langle 2^S, R_1^b, R_2^b, \ldots, R_n^b \rangle$ can be constructed to represent the transition system associated with the belief space. The directed edges R_i^b are defined so that belief state B is related to B' by R_i^b (that is BR_i^bB') if B is mapped to B' by o_i. Like in classical planning, these relations R_i^b are partial functions, and similarly to classical planning, planning without observability is finding a path from the initial belief state I to a belief state G' such that $G' \subseteq G$.

Planning in the belief space can be solved like a state-space search problem: start from the initial belief state and repeatedly follow the directed edges in the belief space to reach new belief states, until a belief state that is a subset of the goal states is reached.

When a heuristic search algorithm is used for avoiding the enumeration of all the belief states, then it is important to use some informative heuristic for guiding the search.

The most obvious heuristic would be an estimate for the distance from the current belief state B to the set of goal states G. First we discuss some admissible heuristics that are derived *from distances in the state space* that underlies the belief space. These are the most obvious heuristics one could use. Then we focus on more informative heuristics that are not directly derived from the distances in the state space and take the observability restrictions into account.

Research on conformant planning so far has concentrated on distance heuristics derived from the distances of individual states in the state space. The distance of a belief state is for example the maximum length of a shortest path from a constituent state to a goal state.

Bonet and Geffner [2000] have used a related distance measure, the optimal expected number of steps of reaching the goal in the corresponding probabilistic problem. Bryce and Kambhampati [2004] have considered efficient approximations of state space distances.

The most obvious distance heuristics are based on the weak and strong distances in the state space. [1]. The weak distances of states are based on the following inductive definition. Sets D_i consist of those states from which a goal state is reachable in i steps or less.

$$\begin{aligned} D_0 &= G \\ D_{i+1} &= D_i \cup \{s \in S | o \in O, s' \in D_i, sos'\} \end{aligned}$$

A state s has *weak distance* $d \geq 1$ if $s \in D_d \backslash D_{d-1}$ and distance 0 if $s \in G$. This means that it is possible to reach one of the goal states starting from s by a sequence of d operators if the nondeterministic alternatives play out favorably. Of course, nondeterministic actions may come out unfavorably and a higher number of actions may be needed, or the goal may even become unreachable, but if it is possible that the goals are reached in d steps then the weak distance is d.

Strong distances are based on a slightly different inductive definition. Now D_i consists of those states for which there is a guarantee of reaching a goal state in i steps or less.

$$\begin{aligned} D_0 &= G \\ D_{i+1} &= D_i \cup \{s \in S | o \in O, s' \in D_i, sos', sos'' \text{ implies } s'' \in D_i \text{ for all } s''\} \end{aligned}$$

A state s has *strong distance* $d \geq 1$ if $s \in D_d \backslash D_{d-1}$ and strong distance 0 if $s \in G$.

Next we derive distance heuristics for the belief space based on state space distances. Both weak and strong distances yield an admissible distance heuristic for belief states, but strong distances are (not always properly) higher than weak distances and therefore a more accurate estimate for plan length.

[1] This terminology is inspired by Cimatti et al. [2003], where it was used for referring to the two ways of computing sets of predecessor states.

Definition 5.24 (State space distance) *The* state space distance *of a belief state B is $d \geq 1$ when $B \subseteq D_d$ and $B \not\subseteq D_{d-1}$, and it is 0 when $B \subseteq D_0 = G$.*

Even though computing the exact distances for a typical succinct representation of transition systems, like STRIPS operators, is PSPACE-hard, the much higher complexity of planning problems with partial observability still often justifies it: this computation would in many cases be an inexpensive preprocessing step, preceding the much more expensive solution of the partially observable planning problem. Otherwise less expensive approximations can be used.

The essence of planning without observability is that the same sequence of actions has to lead to a goal state for every state in the belief state. The distance heuristics from the strong distances in the state space may assign distance 1 to both state s_1 and state s_2, but the distance from $\{s_1, s_2\}$ may be anywhere between 1 and infinite. The inaccuracy in estimating the distance of $\{s_1, s_2\}$ based on the distances of s_1 and s_2 is that the distances of s_1 and s_2 in the state space may be along paths that have nothing to do with the path for $\{s_1, s_2\}$ in the belief space. That is, the actions on these three paths may be completely different.

This leads to a powerful idea. Instead of estimating the distance of B in terms of distances of states $s \in B$ in the state space S, let us estimate it in terms of distances of n-tuples of states in the product state space S^n, in which there is a transition from $\langle s_1, \ldots, s_n \rangle$ to $\langle s'_1, \ldots, s'_n \rangle$ if and only if there is an operator o that allows a transition from s_i to s'_i for every $i \in \{1, \ldots, n\}$. Here the important point is that the transition between the tuples is by using the same operator for every component state. This corresponds to the necessity of using the same operator for every state, because observations cannot distinguish between them.

This leads to a generalization of strong distances. We define the distances of n-tuples of states as follows.

$$\begin{aligned} D_0 &= G^n \\ D_{i+1} &= D_i \cup \{\sigma \in S^n | o \in O, \sigma' \in D_i, \sigma R_o^n \sigma', \sigma R_o^n \sigma'' \text{ implies } \sigma'' \in D_i \text{ for all } \sigma''\} \end{aligned}$$

Here R_o^n is defined by

$$\langle s_1, \ldots, s_n \rangle R_o^n \langle s'_1, \ldots, s'_n \rangle \text{ if } s_i o s'_i \text{ for all } i \in \{1, \ldots, n\}.$$

Now we can define the n-distance of a belief state as follows.

Definition 5.25 (n-distance) *Belief state B has n-distance $d = 0$ if for all $\{s_1, \ldots, s_n\} \subseteq B$, $\langle s_1, \ldots, s_n \rangle \in D_0$ (this is equivalent to $B \subseteq G$.) Belief state B has n-distance $d \geq 1$ if for all $\{s_1, \ldots, s_n\} \subseteq B$, $\langle s_1, \ldots, s_n \rangle \in D_d$ and for some $\{s_1, \ldots, s_n\} \subseteq B$, $\langle s_1, \ldots, s_n \rangle \notin D_{d-1}$. If the distance d is not any natural number, then $d = \infty$.*

So, we look at all the n-element subsets of B, see what their distance to goals is in the product state space S^n, and take the maximum of those distances. Note that when we pick the elements $s_1, \ldots, s_n$ from B, we do not and cannot assume that the elements $s_1, \ldots, s_n$ are distinct, for example because B may have less than n states. Of course, the definition assumes that B has at least one state. The 1-distance of a belief state coincides with the state space distance.

The motivation behind n-distances is that computing the actual distance of belief states is very expensive (as complex as the planning problem itself) but we can use an informative notion of distances for "small" belief states of size n.

Next we investigate the properties of n-distances. The first result shows that n-distances are at least as good an estimate as m-distances when $n > m$. This result is based on a technical lemma

that shows that m-tuples from the definition of m-distances are included in the n-tuples of the definition of n-distances.

Lemma 5.26 (Embedding) *Let $n > m$ and let $D_0, D_1, \ldots$ be the sets in the definition of m-distances, and $D'_0, D'_1, \ldots$ the sets in the definition of n-distances.*

Then for all $i \geq 0$, all belief states B and all $\{s_1, \ldots, s_m\} \subseteq B$, if $\langle s_1, \ldots, s_m, s'_{m+1}, \ldots, s'_n \rangle \in D'_i$ where $s'_k = s_m$ for all $k \in \{m+1, \ldots, n\}$, then $\langle s_1, \ldots, s_m \rangle \in D_i$.

Proof: By induction on i. Base $i = 0$: If $\{s_1, \ldots, s_m\} \subseteq B$ and $\langle s_1, \ldots, s_m, s_m, \ldots, s_m \rangle \in D'_0$ (state s_m is repeated so that the number of components in the tuple is n), then $\langle s_1, \ldots, s_m, s_m, \ldots, s_m \rangle \in G^n$. Consequently, $\langle s_1, \ldots, s_m \rangle \in D_0 = G^m$.

Inductive case $i \geq 1$: We show that if $\langle s_1, \ldots, s_m, s_m, \ldots, s_m \rangle \in D'_i$, then there is an operator $o \in O$ such that for any states $s'_1, \ldots, s'_m$ such that $s_i o s'_i$ for all $i \in \{1, \ldots, m\}$, $\langle s'_1, \ldots, s'_m \rangle \in D_{i-1}$, and hence $\langle s_1, \ldots, s_m \rangle \in D_i$.

So assume $\langle s_1, \ldots, s_m, s_m, \ldots, s_m \rangle \in D'_i$. Hence there is an operator $o \in O$ such that for all $s'_1, \ldots, s'_n$ such that $s_i o s'_i$ for all $i \in \{1, \ldots, m\}$ and $s_m o s'_j$ for all $j \in \{m+1, \ldots, k\}$, the n-tuple $\langle s'_1, \ldots, s'_n \rangle$ is in D'_{i-1}. Now $\langle s'_1, \ldots, s'_m, s'_m, \ldots, s'_m \rangle \in D'_{i-1}$ because this tuple is one of those reachable from $\langle s_1, \ldots, s_m, s_m \ldots, s_m \rangle$, and hence by the induction hypothesis $\langle s'_1, \ldots, s'_m \rangle \in D_{i-1}$. Because this holds for all s'_i reachable by o from the corresponding s_i, we have $\langle s_1, \ldots, s_m \rangle \in D_i$. □

The embedding of m-distances in n-distances as implies that n-distances are more accurate than m-distances when $n > m$.

Theorem 5.27 *Let d_n be the n-distance for a belief state B and d_m the m-distance for B. If $n > m$, then $d_n \geq d_m$.*

Proof: So assume that the m-distance of B is d_m. This implies that there is $\{s_1, \ldots, s_m\} \subseteq B$ such that $\langle s_1, \ldots, s_m \rangle \notin D_{d_m - 1}$. Let $\sigma = \langle s_1, \ldots, s_m, s_m, \ldots, s_m \rangle$ where s_m is repeated $n - m + 1$ times. By Lemma 5.26 $\sigma \notin D'_{d_m - 1}$. Hence there is $\{s_1, \ldots, s_m, s_m, \ldots, s_m\rangle \subseteq B$ such that $\langle s_1, \ldots, s_m, s_m, \ldots, s_m \rangle \notin D'_{d_m}$. Hence the n-distance d_n of B is greater than or equal to d_m. □

So, 2-distances are a better estimate for belief states than the estimate given by state space distances, and 3-distances are a better estimate than 2-distances, and so on.

For the last result we need another lemma, which we give without a proof. The lemma states that the components of the tuples in the sets D_i may be reordered and replaced by existing components.

Lemma 5.28 *Let $D_0, D_1, \ldots,$ be the sets in the definition of n-distances. Then if $\langle s_1, \ldots, s, s', \ldots, s_k \rangle$ is in D_i, then so is $\langle s_1, \ldots, s', s, \ldots, s_k \rangle$, and if $\langle s_1, s_2, s_3, \ldots, s_k \rangle$ is in D_i, then so is $\langle s_1, s_1, s_3, \ldots, s_k \rangle$.*

Theorem 5.29 *Let the belief space be 2^S, where the cardinality of the state space is $n = |S|$. Then the n-distance of B equals the distance of B in the belief space.*

Proof: The proof is by course-of-values induction on the distance i. Let $D_0, D_1, D_2, \ldots$ be the sets in the definition of n-distances.

Induction hypothesis: Belief state $B = \{s_1, \ldots, s_k\}$ has distance $i \geq 1$ if and only if $\sigma = \langle s_1, \ldots, s_k, s_k, \ldots, s_k \rangle \in D_i \backslash D_{i-1}$ (where s_k is repeated so that σ is an n-tuple), and distance 0 if $B \subseteq G$.

Base case $i = 0$: Belief state B has distance 0 iff $B \subseteq G$ iff $\sigma \in G^n = D_0$, which trivially holds.

Inductive case $i \geq 1$: Belief state B has distance i
iff there is operator $o \in O$ such that $B' = \{s'_1, \ldots, s'_{k'}\} = \{s' \in S | s \in B, sos'\}$ has distance $i - 1$ and there is no $o' \in O$ such that $B'' = \{s''_1, \ldots, s''_{k''}\} = \{s'' \in S | s \in B, so's''\}$ has distance less than $i - 1$
iff there is $o \in O$ such that for $\{s'_1, \ldots, s'_{k'}\} = \{s' \in S | s \in B, sos'\}$ we have $\langle s'_1, \ldots, s'_{k'}, s'_{k'}, \ldots, s'_{k'} \rangle \in D_{i-1} \backslash D_{i-2}$ (to nicely handle the case $i = 1$ we define $D_{i-2} = D_{1-2} = D_{-1}$ as $D_{-1} = \emptyset$) and there is no $o' \in O$ such that for $\{s''_1, \ldots, s''_{k''}\} = \{s'' \in S | s \in B, so's''\}$ and any $e \geq 2$ we would have $\langle s''_1, \ldots, s''_{k''}, s''_{k''}, \ldots, s''_{k''} \rangle \in D_{i-e}$ iff $\langle s_1, \ldots, s_k, s_k, \ldots, s_k \rangle \in D_i \backslash D_{i-1}$.

The second "iff" is by two applications of the induction hypothesis, once for the first part with operator o and $i - 1$ and once for the second part with operator o'' and $i - e$, and the third/last iff is established in the rest of the proof.

Because $\langle s'_1, \ldots, s'_{k'}, s'_{k'}, \ldots, s'_{k'} \rangle \in D_{i-1}$, by Lemma 5.28 all the n-tuples having the elements of $B' = \{s'_1, \ldots, s'_{k'}\} = \{s' \in S | s \in B, sos'\}$ or a subset of them as its components are in D_{i-1}. These are all the successor tuples of $\langle s_1, \ldots, s_k, s_k, \ldots, s_k \rangle$ under R^n_o. Hence by definition of D_i $\langle s_1, \ldots, s_k, s_k, \ldots, s_k \rangle$ is in D_i.

Because there are no $o' \in O$ taking one of the tuples consisting of elements of $B = \{s_1, \ldots, s_k\}$ to a tuple in D_{i-e} for $e \geq 2$, we have $\langle s_1, \ldots, s_k, s_k, \ldots, s_k \rangle \notin D_{i-1}$, which completes the equivalence. □

In summary, the accuracy of the n-distances grows as n grows, and asymptotically when n equals the number of states it is perfectly accurate.

In addition to including state space distances as a special case, the family of n-distances also takes into account the cardinalities of belief states, although only in a restricted manner as determined by the magnitude of n. Consider the belief state $B = \{s_1, s_2\}$. Its 2-distance is determined by the membership of the tuples $\sigma_1 = \langle s_1, s_2 \rangle$ (and symmetrically $\langle s_2, s_1 \rangle$), $\sigma_2 = \langle s_1, s_1 \rangle$, and $\sigma_3 = \langle s_2, s_2 \rangle$ in the sets D_i. The distance of σ_1 is at least as high as that of σ_2 and σ_3, because any sequence of actions leading to goals that is applicable for $\{s_1, s_2\}$ is also applicable for s_1 alone and for s_2 alone, and there might be shorter action sequences applicable for s_1 and s_2 but not for $\{s_1, s_2\}$. Therefore, any reduction in the size of a belief state, like from $\{s_1, s_2\}$ to $\{s_1\}$ would appropriately improve the n-distance estimate.

Accuracy of the heuristics

Preceding results show that the accuracy of n-distances increases as n grows, reaching perfect accuracy when n equals the cardinality of the state space. To investigate the impact of more accurate heuristics we have implemented a planner that does heuristic search in the belief space. The planner implements three heuristics for guiding the search algorithms: the 1-distances, the 2-distances, and the cardinality of belief states. The first two heuristics are admissible, and can be used in connection with optimal heuristic search algorithms like A$*$. The third heuristic, size of the belief states, does not directly yield an admissible heuristic, and we use it only in connection with a search algorithm that does not rely on admissibility.

The planner is implemented in C and represents belief states as BDDs with the CUDD system from Colorado University. CUDD provides functions for computing the cardinality of a belief state. Our current implementation does not support n-distances for n other than 1 and 2.

The search algorithms implemented in our planner include the optimal heuristic search algorithm A$*$, the suboptimal family of algorithms WA$*$[2], and suboptimal best-first search which first expands those nodes that have the lowest estimated remaining distance to the goal states.

The main topic to be investigated is the relative accuracy of n-distances, and as a secondary topic we briefly evaluate the effectiveness of different types of search algorithms and heuristics. We use the following benchmarks. Regrettably there are few meaningful benchmarks; all the interesting ones are for the more general problem of partially observable planning.

- Bonet and Geffner [2000] proposed one of the most interesting benchmarks for conformant planning so far, sorting networks [Knuth, 1998]. A sorting network consists of an ordered (or a partially ordered) set of gates acting on a number of input lines. Each gate combines a comparator and a swapper: if first input is greater than the second, then swap the values. The goal is to sort the input sequence. The sorting network always has to perform the same operations irrespective of the input, and hence exactly corresponds to planning without observability.

  Our size parameter is the number of inputs.

- In the empty room benchmark a robot without any sensors moves in a room to north, south, east and west and its goal is to get to the middle of the room. This is possible by going to north or south and then west or east until the robot knows that it is in one of the corners. Then it is easy to go to the goal position. The robot does not know its initial location.

  Size n characterizes room size $2^n \times 2^n$.

- Our blocks world benchmark is the standard blocks world, but with several initial states and modified so that from every initial state every goal state is reachable without observability even when the initial state is not known. For this it is sufficient that the operator for moving a block onto the table is always applicable but has not effect if there is another block on the block. The initial belief state consists of all the possible configurations of the n blocks, and the goal is to build a stack consisting of all the blocks in a fixed order.

- The ring of rooms benchmark involves a round building with a cycle of n rooms with a window in each that can be closed and locked. Initially the state of the windows and the location of the robot is unknown. The robot can move to the next room either in clockwise or counterclockwise direction, and then close and lock the windows. Locking is possible only if the window is closed. Locking an already locked window and closing an already closed window does not have any effect.

  The size parameter is the number of rooms.

There are other benchmarks considered in the literature, but their flavor is close to some of the above, and many can be easily reformulated as planning with full observability.

Table 5.2 makes a comparison on the accuracy of 1-distances and 2-distances on a number of

[2]We parameterize WA$*$ with $W = 5$, giving a 5 times higher value to the estimated remaining distance than to the distance so far, yielding solutions with cost at most 5 times the optimal.

	1-distance		2-distance		exact
	len	%	len	%	len
sort02	1	1.00	1	1.00	1
sort03	1	0.33	2	0.67	3
sort04	2	0.40	3	0.60	5
sort05	2	0.22	3	0.33	9
sort06	3	0.25	4	0.33	12
sort07	3	0.19	5	0.31	16
sort08	4	0.16	6	0.32	19
ring03	8	1.00	8	1.00	8
ring04	11	1.00	11	1.00	11
ring05	14	1.00	14	1.00	14
ring06	17	1.00	17	1.00	17
ring07	20	1.00	20	1.00	20
BW02	2	0.67	3	1.00	3
BW03	4	0.57	5	0.71	7
BW04	6	0.46	8	0.62	13
BW05	7	0.41	9	0.53	17
emptyroom01	2	1.00	2	1.00	2
emptyroom02	4	0.50	8	1.00	8
emptyroom03	8	0.40	20	1.00	20
emptyroom04	16	0.36	44	1.00	44
emptyroom05	32	0.35	92	1.00	92

Table 5.2: Accuracy of 1-distances and 2-distances on a number of problem instances

problem instances. For each heuristic we first give the distance estimate for the initial belief state, followed by the percentage of the actual distance. The actual distance (= length of the shortest plan) was determined by A$*$ and is given in the last column.

As expected, on most of the problems 2-distances are strictly better estimates than 1-distances, and surprisingly, on one of the problems, the empty room navigation problem, the 2-distances equal the lengths of the shortest plans.

For the ring of room problems 1-distances and 2-distances are the same, and coincide with the actual shortest plan length. This is because of the simple structure of the problem and its belief space. It seems that 2-distances would also not provide an advantage over 1-distances on many other problems in which there are no dependencies between state variables with unknown values.

The sorting network problem is the most difficult of the benchmarks in terms of the relation between difficulty and number of state variables. Every initial state (combination of input values) in this benchmark can be solved by a sorting network with a small number of gates (more precisely $\lfloor \frac{n}{2} \rfloor$), which makes the 1-distances small. Increasing n monotonically increases n-distances, but the increase is slow because for a small number of input combinations the smallest network sorting them all is still rather small, and as the number of input value combinations is exponential in the number of inputs, only a tiny fraction of all combinations is covered.

Table 5.3 gives runtimes on all combinations of search algorithm and heuristic. We only report the time spent in the search algorithm, ignoring a preprocessing phase during which BDDs representing 1-distances and 2-distances are computed. Computing 2-distances is more expensive than computing 1-distances because there are twice as many variables in the BDDs and the efficiency

instance	A∗				WA∗				best first					
	1-distance		2-distance		1-distance		2-distance		1-distance		2-distance		cardinality	
	time	len	time	len	time	len	time	len	time	len	time	len	time	len
sort02	0.00	1	0.00	1	0.00	1	0.00	1	0.00	1	0.00	1	0.00	1
sort03	0.00	3	0.00	3	0.00	3	0.00	3	0.00	3	0.00	3	0.00	3
sort04	0.00	5	0.01	5	0.00	5	0.00	5	0.00	6	0.00	6	0.00	5
sort05	0.12	9	0.15	9	0.13	9	0.07	9	0.00	10	0.01	10	0.00	9
sort06	139.41	12	154.64	12	251.87	12	25.81	12	0.01	15	0.01	15	0.00	12
sort07	$> 2h$		$> 2h$		$> 2h$		$> 2h$		0.01	21	0.01	20	0.01	16
sort08	$> 2h$		$> 2h$		$> 2h$		$> 2h$		0.02	28	0.05	28	0.02	19
ring03	0.01	8	0.01	8	0.00	8	0.00	8	0.00	8	0.00	8	0.01	8
ring04	0.00	11	0.01	11	0.00	11	0.00	11	0.00	11	0.01	11	0.00	11
ring05	0.01	14	0.03	14	0.01	14	0.04	14	0.01	14	0.03	14	0.01	14
ring06	0.03	17	0.12	17	0.03	17	0.14	17	0.03	17	0.14	17	0.03	17
BW02	0.00	3	0.00	3	0.00	3	0.00	3	0.00	3	0.00	3	0.00	3
BW03	0.01	7	0.00	7	0.00	7	0.01	7	0.00	7	0.01	7	0.00	7
BW04	0.71	13	0.93	13	0.04	13	0.06	14	0.02	14	0.04	14	0.03	14
BW05	180.47	17	307.62	17	1.26	17	2.87	17	0.40	22	1.41	21	0.36	34
emptyroom01	0.00	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00	2	0.00	4
emptyroom02	0.00	8	0.00	8	0.00	8	0.00	8	0.00	12	0.00	8	0.00	12
emptyroom03	0.16	20	0.01	20	0.03	24	0.00	20	0.00	50	0.00	20	0.00	36
emptyroom04	37.28	44	0.07	44	10.59	52	0.06	44	0.09	222	0.06	44	0.01	106
emptyroom05	$> 2h$		0.92	92	$> 2h$		0.89	92	2.53	950	0.89	92	0.03	342

Table 5.3: Runtimes and plan sizes of a number of problem instances

of BDDs decreases as BDDs grow. The higher accuracy of 2-distances is often reflected in the runtimes.

On the empty room problems, performance of A∗ and 1-distances quickly deteriorates as room size grows, while with 2-distances A∗ immediately constructs optimal plans even for bigger rooms. On sorting networks and WA∗ 2-distances lead to a better performance because of its advantage over 1-distances in accuracy, but finding bigger optimal networks is still very much out of reach.

On all of the problems, best-first search is the fastest to find a plan, but plans were much longer on the empty room and blocks world problems, and slightly worse on sorting networks. For bigger sorting networks the cardinality heuristic combined with best-first is the best combination, as runtimes with the other heuristics and with A∗ and WA∗ grow much faster. We believe that our collection of benchmarks is too small to say conclusively anything general about the relative merits of the heuristics.

Interestingly, our planner with best-first search and the cardinality heuristic produces optimal sorting networks up to size 8 (Bonet and Geffner [2000] report producing networks until size 6 with an optimal algorithm), and for bigger networks the difference to known best networks is first relatively small, but later grows; see Table 5.4.

Related work

Bonet and Geffner [2000] were one of the first to apply heuristic state-space search to planning in the belief space. They used a variant of the state space distance heuristic considered by us, with the difference that they were addressing probabilistic problems and considered expected distances

inputs	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
gates (best known)	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60
gates (our planner)	1	3	5	9	12	16	19	26	31	39	46	56	64	74	81

Table 5.4: Sizes of sorting networks found by best-first search and cardinality heuristic. Networks up to 8 inputs are optimal. From 9 on optimal network sizes are not known. Total runtime for 16 inputs is 5.81 seconds on a 800 MHz Pentium.

of states under the optimal probabilistic plan, instead of the non-probabilistic weak or strong distances.

Bryce and Kambhampati [2004] compute distances with Graphplan's [Blum and Furst, 1997] planning graphs, and recognize that the smallest of the weak distances of states in a belief state – as is trivially obtained from planning graphs – is not very informative, and propose improvements based on multiple planning graphs: for a formula $\chi_1 \vee \chi_2 \vee \cdots \vee \chi_n$ describing a belief state, compute the estimate for each χ_i separately. Then an admissible estimate for the whole belief state is bounded from above by $\max_{i=1}^{n} \min_{s \in \sigma(\chi_i)} \delta(s)$ where $\sigma(\chi_i)$ is the set of states described by χ_i, and $\delta(s)$ is the distance from state s to a goal state. This is below the state space distances (1-distances) because minimization is used, not maximization. It may be difficult to do distance maximization with planning graphs as they do not represent most dependencies between state variables.

Smith and Weld's [1998] multiple planning graphs and especially their induced mutexes are related to our n-distances. They compute a kind of approximation of our n-distances, but as this computation is based on distances of state variable values as in the work by Bryce and Kambhampati, the approximation is not very good. With the multiple planning graphs there do not appear to be useful ways of controlling the accuracy parameter n, and Smith and Weld then essentially consider n that equals the number of initial states for deterministic problems.

Haslum and Geffner [2000] have defined a family of increasingly accurate heuristics for classical deterministic planning. Their accuracy parameter n is the number of state variables analogously to our parameter n of states. However, they give approximations of distances in the state space (our 1-distances), not in the belief space, and many of the phenomena important for conditional planning, like nondeterminism, do not show up in their framework.

5.4 Planning with partial observability

Planning with partial observability is much more complicated than its two special cases with full and no observability. Like planning without observability, the notion of belief states becomes very important. Like planning with full observability, formalization of plans as sequences of operators is insufficient. However, plans also cannot be formalized as mappings from states to operators because partial observability implies that the current state is not necessarily unambiguously known. Hence we will need the general definition of plans introduced in Section 4.1.2.

When executing operator o in belief state B the set of possible successor states is $img_o(B)$, and based on the observation that are made, this set is restricted to $B' = img_o(B) \cap C$ where C is the equivalence class of observationally indistinguishable states corresponding to the observation.

In planning with unobservability, a backward search algorithm starts from the goal belief state

and uses regression or strong preimages for finding predecessor belief states until a belief state covering the initial belief state is found.

With partial observability, plans do not just contain operators but may also branch. With branching the sequence of operators may depend on the observations, and this makes it possible to reach goals also when no fixed sequence of operators reaches the goals. Like strong preimages in backward search correspond to images, the question arises what does branching correspond to in backward search?

Example 5.30 Consider the blocks world with three blocks with the goal state in which all the blocks are on the table. There are three operators, each of which picks up one block (if there is nothing on top of it) and places it on the table. We can only observe which blocks are not below another block. This splits the state space to seven observational classes, corresponding to the valuations of the state variables clear-A, clear-B and clear-C in which at least one block is clear.

The plan construction steps are given in Figure 5.1. Starting from the top left, the first diagram depicts the goal belief state. The second diagram depicts the belief states obtained by computing the strong preimage of the goal belief state with respect to the move-A-onto-table action and splitting the set of states to belief states corresponding to the observational classes. The next two diagrams are similarly for strong preimages of move-B-onto-table and move-C-onto-table.

The fifth diagram depicts the computation of the strong preimage from the union of two existing belief states in which the block A is on the table and C is on B or B is on C. In the resulting belief state A is the topmost block in a stack containing all three blocks. The next two diagrams similarly construct belief states in which respectively B and C are the topmost blocks.

The last three diagrams depict the most interesting cases, constructing belief states that subsume two existing belief states in one observational class. The first diagram depicts the construction of the belief state consisting of both states in which A and B are clear and C is under either A or B. This belief state is obtained as the strong preimage of the union of two existing belief states, the one in which all blocks are on the table and the one in which A is on the table and B is on top of C. The action that moves A onto the table yields the belief state because if A is on C all blocks will be on the table and if A is already on the table nothing will happen. Construction of the belief states in which B and C are clear and A and C are clear is analogous and depicted in the last two diagrams.

The resulting plan reaches the goal state from any state in the blocks world. The plan in the program form is given in Figure 5.2 (order of construction is from the end to the beginning.)

■

We restrict to acyclic plans. Construction of cyclic plans requires looking at more global properties of transition graphs than what is needed for acyclic plans. Taking these local properties into account is difficult because we want to avoid explicit enumeration of the belief states.

5.4.1 Problem representation

Now we introduce the representation for sets of state sets for which a plan for reaching goal states exists.

In the following example states are viewed as valuations of state variables, and the observational classes correspond to valuations of those state variables that are observable.

Example 5.31 Consider the blocks world with the state variables *clear(X)* observable, allowing

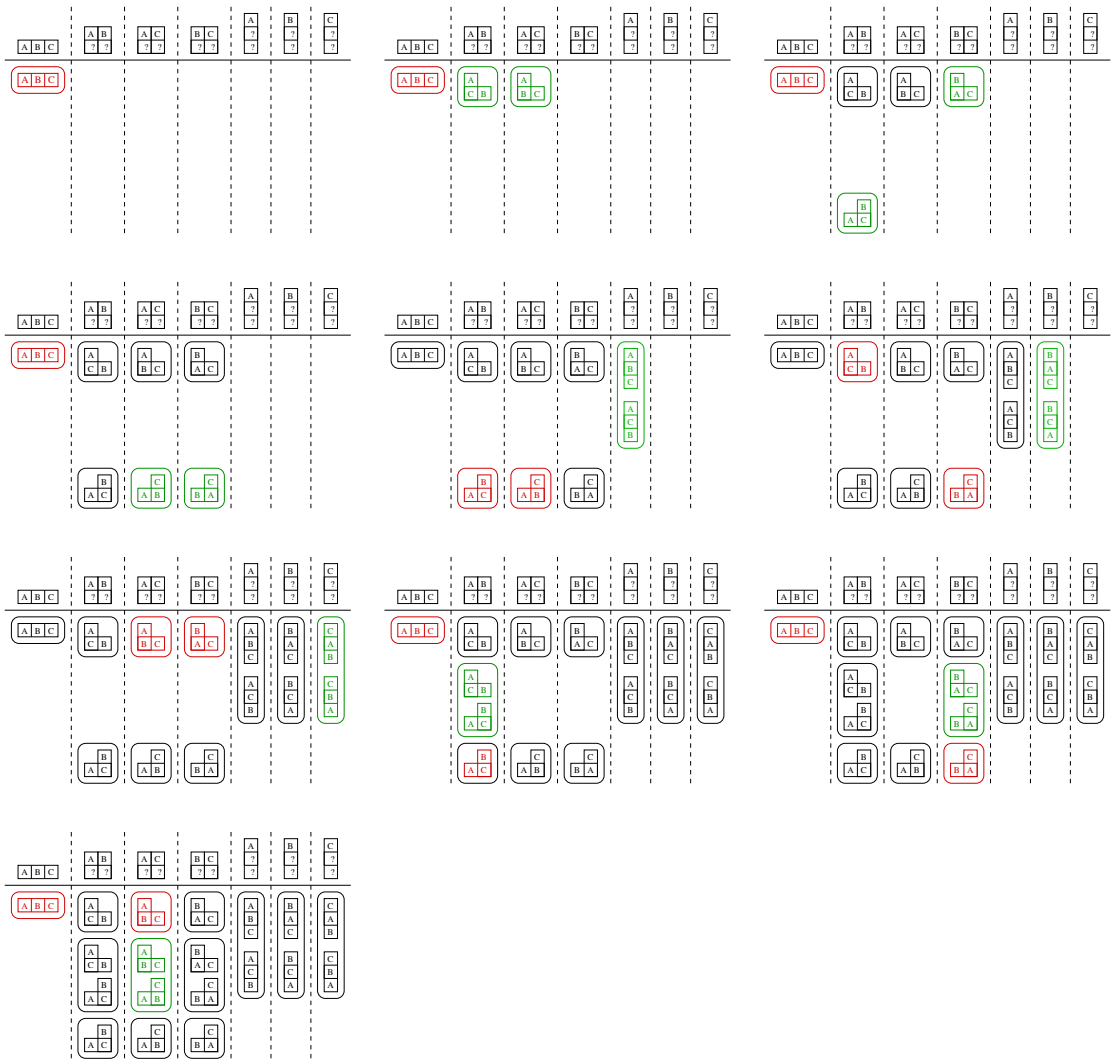
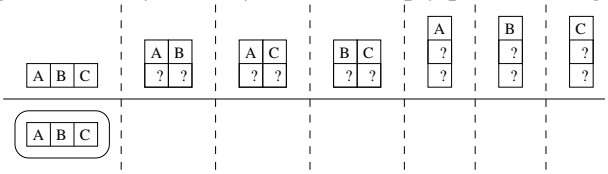


Figure 5.1: Solution of a simple blocks world problem

to observe the topmost block of each stack. With three blocks there are 7 observational classes because there are 7 valuations of $\{\text{clear(A)}, \text{clear(B)}, \text{clear(C)}\}$ with at least one block clear.

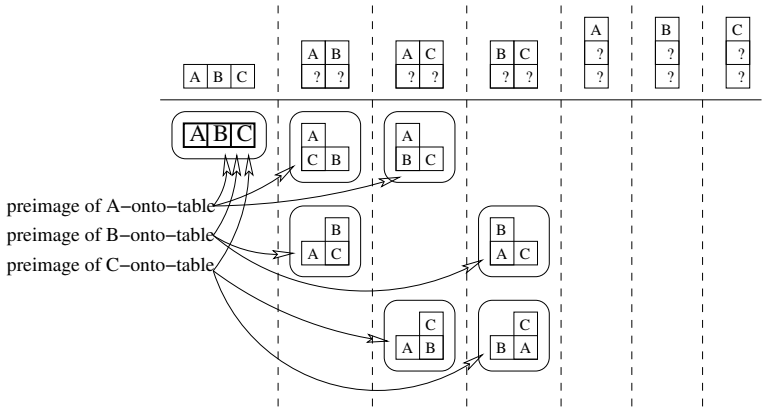
Consider the problem of trying to reach the state in which all blocks are on the table. For each block there is an action for moving it onto the table from wherever it was before. If a block cannot be moved nothing happens. Initially we only have the empty plan for the goal states.



Then we compute the preimages of this set with actions that respectively put the blocks A, B and C onto the table, and split the resulting sets to the different observational classes.

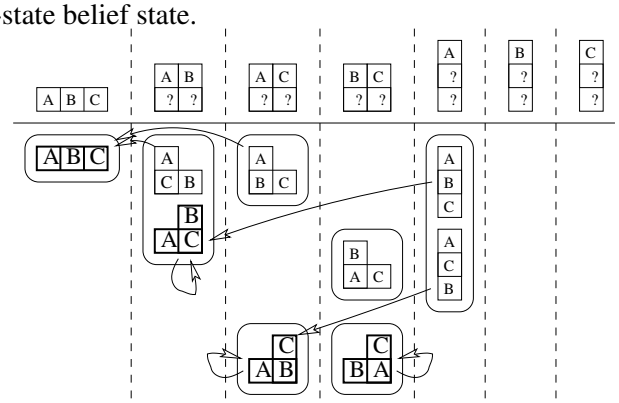
```
16:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-C THEN GOTO 15
  IF clear-B AND clear-C THEN GOTO 13
  IF clear-A AND clear-B THEN GOTO 11
  IF clear-A THEN GOTO 5
  IF clear-B THEN GOTO 7
  IF clear-C THEN GOTO 9
15:
  move-C-onto-table
14:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-C THEN GOTO 1
13:
  move-B-onto-table
12:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-B AND clear-C THEN GOTO 3
11:
  move-A-onto-table
10:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-B THEN GOTO 2
9:
  move-C-onto-table
8:
  IF clear-A AND clear-C THEN GOTO 1
  IF clear-B AND clear-C THEN GOTO 2
7:
  move-B-onto-table
6:
  IF clear-A AND clear-B THEN GOTO 1
  IF clear-B AND clear-C THEN GOTO 3
5:
  move-A-onto-table
4:
  IF clear-A AND clear-B THEN GOTO 2
  IF clear-A AND clear-C THEN GOTO 3
3:
  move-C-onto-table
  GOTO end
2:
  move-B-onto-table
  GOTO end
1:
  move-A-onto-table
end:
```

Figure 5.2: A plan for a partially observable blocks world problem



Now for these 7 belief states we have a plan consisting of one or zero actions. But we also have plans for sets of states that are only represented implicitly. These involve branching. For example, we have a plan for the state set consisting of the four states in which respectively all blocks are on the table, A is on C, A is on B, and B is on A. This plan first makes observations and branches, and then executes the plan associated with the belief state obtained in each case. Because 3 observational classes each have 2 belief states, there are 2^3 maximal state sets with a branching plan. From each class only one belief state can be chosen because observations cannot distinguish between belief states in the same class.

We can find more belief states that have plans by computing preimages of existing belief states. Let us choose the belief states in which respectively all blocks are on the table, B is on C, C is on B, and C is on A, and compute their union's preimage with A-onto-table. The preimage intersected with the observational classes yields new belief states: for the class with A and B clear there is a new 2-state belief state covering both previous belief states in the class, and for the class with A clear there is a new 2-state belief state.



Computation of further preimages yields for each observational class a belief state covering all the states in that class, and hence a plan for every belief state. ■

Next we formalize the framework in detail.

Definition 5.32 (Belief space) *Let $P = (C_1, \ldots, C_n)$ be a partition of the set of all states. Then* a belief space *is an n-tuple $\langle G_1, \ldots, G_n \rangle$ where $G_i \subseteq 2^{C_i}$ for all $i \in \{1, \ldots, n\}$ and $B \not\subset B'$ for all $i \in \{1, \ldots, n\}$ and $\{B, B'\} \subseteq G_i$.*

Note that in each component of a belief space we only have set-inclusion maximal belief states. The simplest belief spaces are obtained from sets B of states as $\mathcal{F}(B) = \langle \{C_1 \cap B\}, \ldots, \{C_n \cap$

$B\}\rangle$. A belief space is extended as follows.

Definition 5.33 (Extension) *Let $P = (C_1, \ldots, C_n)$ be the partition of all states, $G = \langle G_1, \ldots, G_n\rangle$ a belief space, and T a set of states. Define $G \oplus T$ as $\langle G_1 \Cup (T \cap C_1), \ldots, G_n \Cup (T \cap C_n)\rangle$ where the operation $\Cup$ adds the latter set of states to the former set of sets of states and eliminates sets that are not set-inclusion maximal, defined as $U \Cup V = \{R \in U \cup \{V\} | R \not\subset K \text{ for all } K \in U \cup \{V\}\}$.*

A belief space $G = \langle G_1, \ldots, G_n\rangle$ represents the set of sets of states flat$(G) = \{B_1 \cup \cdots \cup B_n | B_i \in G_i$ for all $i \in \{1, \ldots, n\}\}$ and its cardinality is $|G_1| \cdot |G_2| \cdot \ldots \cdot |G_n|$.

5.4.2 Complexity of basic operations

The basic operations on belief spaces needed in planning algorithms are testing the membership of a set of states in a belief space, and finding a set of states whose preimage with respect to an action is not contained in the belief space. Next we analyze the complexity of these operations.

Theorem 5.34 *For belief spaces G and state sets B, testing whether there is $B' \in$ flat(G) such that $B \subseteq B'$, and computing $G \oplus B$ takes polynomial time.*

Proof: Idea: A linear number of set-inclusion tests suffices. □

Our algorithm for extending belief spaces by computing the preimage of a set of states (Lemma 5.36) uses exhaustive search and runs in worst-case exponential time. This asymptotic worst-case complexity is very likely the best possible because the problem is NP-hard. Our proof for this fact is a reduction from SAT: represent each clause as the set of literals that are not in it, and then a satisfying assignment is a set of literals that is not included in any of the sets, corresponding to the same question about belief spaces.

Theorem 5.35 *Testing if for belief space G there is $R \in$ flat(G) such that preimg$_o(R) \not\subseteq R'$ for all $R' \in$ flat(G) is NP-complete. This holds even for deterministic actions o.*

Proof: Membership is easy: For $G = \langle G_1, \ldots, G_n\rangle$ choose nondeterministically $R_i \in G_i$ for every $i \in \{1, \ldots, n\}$, compute $R =$ preimg$_o(R_1 \cup \cdots \cup R_n)$, and verify that $R \cap C_i \not\subseteq B$ for some $i \in \{1, \ldots, n\}$ and all $B \in G_i$. Each of these steps takes only polynomial time.

Let $T = \{c_1, \ldots, c_m\}$ be a set of clauses over propositions $A = \{a_1, \ldots, a_k\}$. We define a belief space based on states $\{a_1, \ldots, a_k, \hat{a}_1, \ldots, \hat{a}_k, z_1, \ldots, z_k, \hat{z}_1, \ldots, \hat{z}_k\}$. The states $\hat{a}$ represent negative literals. Define

$$\begin{array}{rcl} c'_i & = & (A \backslash c_i) \cup \{\hat{a} | a \in A, \neg a \notin c_i\} \text{ for } i \in \{1, \ldots, m\}, \\ G & = & \langle \{c'_1, \ldots, c'_m\}, \{\{z_1\}, \{\hat{z}_1\}\}, \ldots, \{\{z_k\}, \{\hat{z}_k\}\}\rangle, \\ o & = & \{\langle a_i, z_i\rangle | 1 \leq i \leq k\} \cup \{\langle \hat{a}_i, \hat{z}_i\rangle | 1 \leq i \leq k\}. \end{array}$$

We claim that T is satisfiable if and only if there is $B \in$ flat(G) such that preimg$_o(B) \not\subseteq B'$ for all $B' \in$ flat(G).

Assume T is satisfiable, that is, there is M such that $M \models T$. Define $M' = \{z_i | a_i \in A, M \models a_i\} \cup \{\hat{z}_i | a_i \in A, M \not\models a_i\}$. Now $M' \subseteq B$ for some $B \in$ flat(G) because from each class only one of $\{z_i\}$ or $\{\hat{z}_i\}$ is taken. Let $M'' =$ preimg$_o(M') = \{a_i \in A | M \models a_i\} \cup \{\hat{a}_i | a_i \in A, M \not\models a_i\}$. We show that $M'' \not\subseteq B$ for all $B \in$ flat(G). Take any $i \in \{1, \ldots, m\}$. Because $M \models c_i$, there is

```
1:  procedure findnew(o,A,F,H);
2:  if F = ⟨⟩ and preimg_o(A) ⊈ B for all B ∈ flat(H)
3:  then return A;
4:  if F = ⟨⟩ then return ∅;
5:  F is ⟨{f_1, ..., f_m}, F_2, ..., F_k⟩ for some k ≥ 1;
6:  for i := 1 to m do
7:    B := findnew(o,A ∪ f_i,⟨F_2, ..., F_k⟩,H);
8:    if B ≠ ∅ then return B;
9:  end;
10: return ∅;
```

Figure 5.3: Algorithm for finding new belief states

$a_j \in c_i \cap A$ such that $M \models a_j$ (or $\neg a_j \in c_i$, for which the proof goes similarly.) Now $z_j \in M'$, and therefore $a_j \in M''$. Also, $a_j \notin c'_j$. As there is such an a_j (or $\neg a_j$) for every $i \in \{1, \ldots, m\}$, M'' is not a subset of any c'_i, and hence $M'' \not\subseteq B$ for all $B \in \text{flat}(G)$.

Assume there is $B \in \text{flat}(G)$ such that $D = \mathit{preimg}_o(B) \not\subseteq B'$ for all $B' \in \text{flat}(G)$. Now D is a subset of $A \cup \{\hat{a} | a \in A\}$ with at most one of a_i and $\hat{a}_i$ for any $i \in \{1, \ldots, k\}$. Define a model M such that for all $a \in A$, $M \models a$ if and only if $a \in D$. We show that $M \models T$. Take any $i \in \{1, \ldots, m\}$ (corresponding to a clause.) As $D \not\subseteq B$ for all $B \in \text{flat}(G)$, $D \not\subseteq c'_i$. Hence there is a_j or $\hat{a}_j$ in $D \backslash c'_i$. Consider the case with a_j ($\hat{a}_j$ goes similarly.) As $a_j \notin c'_i$, $a_j \in c_i$. By definition of M, $M \models a_j$ and hence $M \models c_i$. As this holds for all $i \in \{1, \ldots, m\}$, $M \models T$. □

5.4.3 Algorithms

Based on the problem representation in the preceding section, we devise a planning algorithm that repeatedly identifies new belief states (and associated plans) until a plan covering the initial states is found. The algorithm in Figure 5.4 tests for plan existence; further book-keeping is needed for outputting a plan. The size of the plan is proportional to the number of iterations the algorithm performs, and outputting the plan takes polynomial time in the size of the plan. The algorithm uses the subprocedure *findnew* (Figure 5.3) for extending the belief space (this is the NP-hard subproblem from Theorem 5.35). Our implementation of the subprocedure orders sets $f_1, \ldots, f_m$ by cardinality in a decreasing order: bigger belief states are tried first. We also use a simple pruning technique for deterministic actions o: If $\mathit{preimg}_o(f_i) \subseteq \mathit{preimg}_o(f_j)$ for some i and j such that $i > j$, then we may ignore f_i.

Lemma 5.36 *Let H be a belief space and o an action. The procedure call findnew(o,∅,F, H) returns a set B' of states such that $B' = \mathit{preimg}_o(B)$ for some $B \in \mathit{flat}(F)$ and $B' \not\subseteq B''$ for all $B'' \in \mathit{flat}(H)$, and if no such belief state exists it returns ∅.*

Proof: Sketch: The procedure goes through the elements $\langle B_1, \ldots, B_n \rangle$ of $F_1 \times \cdots \times F_n$ and tests whether $\mathit{preimg}_o(B_1 \cup \cdots \cup B_n)$ is in H. The sets $B_1 \cup \cdots \cup B_n$ are the elements of $\text{flat}(F)$. The traversal through $F_1 \times \cdots \times F_n$ is by generating a search tree with elements of F_1 as children of the root node, elements of F_2 as children of every child of the root node, and so on, and testing whether the preimage is in H. The second parameter of the procedure represents the state set constructed so far from the belief space, the third parameter is the remaining belief space, and the

```
 1:  procedure plan(I,O,G);
 2:  H := F(G);
 3:  progress := true;
 4:  while progress and I ⊈ I' for all I' ∈ flat(H) do
 5:    progress := false;
 6:    for each o ∈ O do
 7:      B := findnew(o,∅,H,H);
 8:      if B ≠ ∅ then
 9:        begin
10:          H := H ⊕ preimg_o(B);
11:          progress := true;
12:        end;
13:    end;
14:  end;
15:  if I ⊆ I' for some I' ∈ flat(H) then return true
16:  else return false;
```

Figure 5.4: Algorithm for planning with partial observability

last parameter is the belief space that is to be extended, that is, the new belief state may not belong to it. □

The correctness proof of the procedure *plan* consists of the following lemma and theorems. The first lemma simply says that extending a belief space H is monotonic in the sense that the members of flat(H) can only become bigger.

Lemma 5.37 *Assume T is any set of states and $B \in$ flat(H). Then there is $B' \in$ flat$(H \oplus T)$ so that $B \subseteq B'$.*

The second lemma says that if we have belief states in different observational classes such that each is included in a belief state of a belief space H, then there is a set in flat(H) that includes all these belief states.

Lemma 5.38 *Let $B_1, \ldots, B_n$ be sets of states so that for every $i \in \{1, \ldots, n\}$ there is $B'_i \in$ flat(H) such that $B_i \subseteq B'_i$, and there is no observational class C such that for some $\{i, j\} \subseteq \{1, \ldots, n\}$ both $i \neq j$ and $B_i \cap C \neq \emptyset$ and $B_j \cap C \neq \emptyset$. Then there is $B' \in$ flat(H) such that $B_1 \cup \cdots \cup B_n \subseteq B'$.*

The proof of the next theorem shows how the algorithm is capable of finding any plan by constructing it bottom up starting from the leaf nodes. The construction is based on first assigning a belief state to each node in the plan, and then showing that the algorithm reaches that belief state from the goal states by repeated computation of preimages.

Theorem 5.39 *Whenever there exists a finite acyclic plan for a problem instance, the algorithm in Figure 5.4 returns* true.

Proof: Assume that there is a plan $\langle N, b, l\rangle$ for a problem instance $\langle S, I, O, G, P\rangle$. We assume that states in S are valuations of a set of state variables. Label all nodes of the plan as follows. Each

initial node n_i for $i \in \{1, \ldots, m\}$ with $\{\langle \phi_1, n_1 \rangle, \ldots, \langle \phi_m, n_m \rangle\}$ we assign the label $Z(n_i) = \{s \in I | s \models \phi_i\}$.

When all parent nodes $n_1, \ldots, n_m\rangle$ of a node n have a label, we assign a label to n. Let $l(n_i) = \langle o_i, \{\langle \phi_i, n \rangle, \ldots\}\rangle$ for all $i \in \{1, \ldots, m\}$. Then $Z(n) = \bigcup_{i \in \{1,\ldots,n\}} \{s \in \mathit{img}_{o_i}(Z(n_i)) | s \models \phi_i\}$. If the above labeling does not assign anything to a node n, then assign $Z(n) = \emptyset$. Each node is labeled with exactly those states that are possible in that node on some execution.

We show that if plans for $Z(n_1), \ldots, Z(n_k)$ exist, where $n_1, \ldots, n_k$ are children of a node n, then the algorithm determines that a plan for $Z(n)$ exists as well.

Induction hypothesis: for every plan node n such that all paths from it to a terminal node have length i or less, $B = Z(n)$ is a subset of some $B' \in \text{flat}(H)$ where H is the value of the program variable H after the *while* loop exits and H could not be extended further.

Base case $i = 0$: Terminal nodes of the plan are labeled with subsets of G. By Lemma 5.37 there is G' such that $G \subseteq G'$ and $G' \in \text{flat}(H)$ because initially $H = \mathcal{F}(G)$ and thereafter it was repeatedly extended.

Inductive case $i \geq 1$: Let n be a plan node with $l(n) = (o, \{\langle \phi_1, n_1 \rangle, \ldots, \langle \phi_k, n_k \rangle\}$.

We show that $Z(n) \subseteq B$ for some $B \in \text{flat}(H)$.

By the induction hypothesis $Z(n_i) \subseteq B$ for some $B \in \text{flat}(H)$ for all $i \in \{1, \ldots, k\}$.

For all $i \in \{1, \ldots, k\}$ $\{s \in \mathit{img}_o(Z(n)) | s \models \phi_i\} \subseteq Z(n_i)$.

Hence by Lemma 5.38 $B = \bigcup_{i \in \{1,\ldots,k\}} \{s \in \mathit{img}_o(Z(n)) | s \models \phi_i\} \subseteq B'$ for some $B' \in \text{flat}(H)$. Assume that there is no such B''. But now by Lemma 5.36 findnew(o,$\emptyset$,H,H) would return B''' such that $\mathit{preimg}_o(B''') \not\subseteq B$ for all $B \in \text{flat}(H)$, and the *while* loop could not have exited with H, contrary to our assumption about H. □

Theorem 5.40 *Let $\Pi = \langle S, I, O, G, P\rangle$ be a problem instance. If plan(I,O,G) returns* true*, then Π has a solution plan.*

Proof: Let $H_0, H_1, \ldots$ be the sequence of belief spaces H produced by the algorithm.

Induction hypothesis: For every $B \in H_{i,j}$ for some $j \in \{1, \ldots, n\}$ and $H_i = \langle H_{i,1}, \ldots, H_{i,n}\rangle$ a plan reaching G exists.

Base case $i = 0$: Every component of H_0 consists of a subset of G. The empty plan reaches G.

Inductive case $i \geq 1$: H_{i+1} is obtained as $H_i \oplus \mathit{preimg}_o(B)$ where $B =$ findnew(o,$\emptyset$,H_i,H_i) and o is an operator.

By Lemma 5.36 $B \in \text{flat}(H_i)$. By the induction hypothesis there are plans π_i for every $B \cap C_i, i \in \{1, \ldots, n\}$. The plan that executes o followed by π_i on observation C_i reaches G from $\mathit{preimg}_o(B)$.

Let $B' \in H_{i+1,j}$ for $H_{i+1} = \langle H_{i+1,1}, \ldots, H_{i+1,n}\rangle$ and some $j \in \{1, \ldots, n\}$. We show that for B' there is a plan for reaching G.

If $B' \in H_{i,j}$ then by the induction hypothesis a plan exists.

Otherwise $B' \subseteq \mathit{preimg}_o(B)$ and we can use the plan for $\mathit{preimg}_o(B)$ that first applies o and then continues with a plan associated with one of the belief states in H_i. □

It would be easy to define an algorithm that systematically generates all belief states (plans) breadth-first and therefore plans with optimal execution lengths, but this algorithm would in practice be much slower and plans would be bigger.

Above we have used only one partition of the state space to observational classes. However, it is straightforward to generalize the above definitions and algorithms to the case in which several partitions are used, each for a different set of actions. This means that the possible observations depend on the action that has last been taken.

5.5 Literature

There is a difficult trade-off between the two extreme approaches, producing a conditional plan covering all situations that might be encountered, and planning only one action ahead. Schoppers [1987] proposed *universal plans* as a solution to the high complexity of planning. Ginsberg [1989] attacked Schopper's idea. Schopper's proposal was to have memoryless plans that map any given observations to an action. He argued that plans have to be memoryless in order to be able to react to all the unforeseeable situations that might be encountered during plan execution. Ginsberg argued that plans that are able to react to all possible situations are necessarily much too big to be practical. It seems to us that Schopper's insistence on using plans without a memory is not realistic nor necessary, and that most of Ginsberg's argumentation on impracticality of universal plans relies on the lack of any memory in the plan execution mechanism. Of course, we agree that a conditional plan that can be executed efficiently can be much bigger than a plan or a planner that has no restrictions on the amount of time consumed in deciding about the action to be taken. Plans without such restrictions could have as high expressivity as Turing machines, for example, and then a conditional plan does not have to be less succinct than the description of a general purpose planning algorithm.

There is some early work on conditional planning that mostly restricts to the fully observable case and is based on partial-order planning [Etzioni *et al.*, 1992; Peot and Smith, 1992; Pryor and Collins, 1996]. We have not discussed these algorithms because they have only been shown to solve very small problem instances.

A variant of the algorithm for constructing plans for nondeterministic planning with full observability in Section 4.3.1 was first presented by Cimatti et al. [2003]. The algorithms by Cimatti et al. construct mappings of states to actions whereas our presentation in Section 4.3.1 focuses on the computation of distances of states, and plans are synthesized afterwards on the basis of the distances. We believe that our algorithms are conceptually simpler. Cimatti et al. also presented an algorithm for finding *weak plans* that may reach the goals but are not guaranteed to. However, finding weak plans is polynomially equivalent to the deterministic planning problem of Chapter 3.

The nondeterministic planning problem with unobservability is not very interesting because all robots and intelligent beings can sense their environment to at least some extent. However, there are problems (outside AI) that are equivalent to the unobservable planning problem. Finding homing/reset/synchronization sequences of circuits/automata is an example of such a problem [Pixley *et al.*, 1992].

Bertoli et al. have presented a forward search algorithm for finding conditional plans in the general partially observable case [Bertoli *et al.*, 2001b].

Bibliography

[Alur *et al.*, 1997] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 340–351. Springer-Verlag, 1997.

[Anderson *et al.*, 1998] C. Anderson, D. Smith, and D. Weld. Conditional effects in Graphplan. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 44–53. AAAI Press, 1998.

[Bäckström and Nebel, 1995] C. Bäckström and B. Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[Balcázar *et al.*, 1988] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Springer-Verlag, Berlin, 1988.

[Balcázar *et al.*, 1990] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*. Springer-Verlag, Berlin, 1990.

[Baral *et al.*, 2000] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1):241–267, 2000.

[Bensalem *et al.*, 1996] S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In R. Alur and T. A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 323–335, New Brunswick, New Jersey, USA, July 1996. Springer-Verlag.

[Bertoli *et al.*, 2001a] P. Bertoli, A. Cimatti, and M. Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 467–472, 2001.

[Bertoli *et al.*, 2001b] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 473–478. Morgan Kaufmann Publishers, 2001.

[Best and Devillers, 1987] E. Best and R. Devillers. Sequential and concurrent behavior in Petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.

[Blum and Furst, 1997] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[Bonet and Geffner, 2000] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61. AAAI Press, 2000.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Bryant, 1992] R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[Bryce and Kambhampati, 2004] D. Bryce and S. Kambhampati. Heuristic guidance measure for conformant planning. In *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 365–374. AAAI Press, 2004.

[Burch *et al.*, 1994] J. R. Burch, E. M. Clarke, D. E. Long, K. L. MacMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, 1994.

[Bylander, 1994] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[Bylander, 1996] T. Bylander. A probabilistic analysis of propositional STRIPS planning. *Artificial Intelligence*, 81(1-2):241–271, 1996.

[Calvanese *et al.*, 2002] D. Calvanese, G. De Giacomo, and M. Y. Vardi. Reasoning about actions and planning in LTL action theories. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eighth International Conference (KR 2002)*, pages 593–602, 2002.

[Castellini *et al.*, 2003] C. Castellini, E. Giunchiglia, and A. Tacchella. SAT-based planning in complex domains: concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147(1–2):85–117, 2003.

[Chandra *et al.*, 1981] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Cimatti *et al.*, 2003] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1–2):35–84, 2003.

[Cimatti, 2003] A. Cimatti, 2003. personal communication.

[Clarke *et al.*, 1994] E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. Technical Report CS-94-204, Carnegie Mellon University, School of Computer Science, October 1994.

[de Bakker and de Roever, 1972] J. W. de Bakker and W. P. de Roever. A calculus of recursive program schemes. In *Proceedings of the First International Colloquium on Automata, Languages and Programming*, pages 167–196. North-Holland, 1972.

[Diekert and Métivier, 1997] V. Diekert and Y. Métivier. Partial commutation and traces. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 457–534. Springer-Verlag, 1997.

[Dijkstra, 1976] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, New Jersey, 1976.

[Dimopoulos *et al.*, 1997] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Fourth European Conference on Planning (ECP'97)*, number 1348 in Lecture Notes in Computer Science, pages 169–181. Springer-Verlag, 1997.

[Do and Kambhampati, 2001] M. B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.

[Efron and Tibshirani, 1986] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1:54–75, 1986.

[Efron and Tibshirani, 1993] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, New York, 1993.

[Emerson and Sistla, 1996] E. A. Emerson and A. P. Sistla. Symmetry and model-checking. *Formal Methods in System Design: An International Journal*, 9(1/2):105–131, 1996.

[Ernst *et al.*, 1997] M. Ernst, T. Millstein, and D. S. Weld. Automatic SAT-compilation of planning problems. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1169–1176. Morgan Kaufmann Publishers, 1997.

[Erol *et al.*, 1995] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1–2):75–88, 1995.

[Etzioni *et al.*, 1992] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, pages 115–125. Morgan Kaufmann Publishers, October 1992.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 956–961. Morgan Kaufmann Publishers, 1999.

[Gerevini and Schubert, 1998] A. Gerevini and L. Schubert. Inferring state constraints for domain-independent planning. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 905–912. AAAI Press, 1998.

[Ghallab *et al.*, 1998] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the Planning Domain Definition Language, version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University, October 1998.

[Giacomo and Vardi, 2000] G. D. Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In S. Biundo and M. Fox, editors, *Recent Advances in AI Planning. Fifth European Conference on Planning (ECP'99)*, number 1809 in Lecture Notes in Artificial Intelligence, pages 226–238. Springer-Verlag, 2000.

[Ginsberg, 1989] M. L. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4):40–44, 1989.

[Godefroid, 1991] P. Godefroid. Using partial orders to improve automatic verification methods. In E. M. Clarke, editor, *Proceedings of the 2nd International Conference on Computer-Aided Verification (CAV '90), Rutgers, New Jersey, 1990*, number 531 in Lecture Notes in Computer Science, pages 176–185. Springer-Verlag, 1991.

[Hart *et al.*, 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum-cost paths. *IEEE Transactions on System Sciences and Cybernetics*, SSC-4(2):100–107, 1968.

[Haslum and Geffner, 2000] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 140–149. AAAI Press, 2000.

[Haslum and Jonsson, 2000] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In S. Biundo and M. Fox, editors, *Recent Advances in AI Planning. Fifth European Conference on Planning (ECP'99)*, number 1809 in Lecture Notes in Artificial Intelligence, pages 308–318. Springer-Verlag, 2000.

[Heljanko, 2001] K. Heljanko. 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*, pages 218–232. Springer-Verlag, 2001.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Howard, 1960] R. A. Howard. *Dynamic programming and Markov decision processes*. The MIT Press, 1960.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press, August 1996.

[Kautz and Selman, 1999] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 318–325. Morgan Kaufmann Publishers, 1999.

[Kautz and Walser, 1999] H. Kautz and J. Walser. State-space planning by integer optimization. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 526–533. AAAI Press, 1999.

[Khomenko *et al.*, 2005] A. Khomenko, Victor an Kondratyev, M. Koutny, and W. Vogler. Merged processes - a new condensed representation of Petri net behaviour. Technical report CS-TR 884, School of Computing Science, University of Newcastle upon Tyne, January 2005.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[Knuth, 1998] D. E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Publishing Company, 1998.

[Korf, 1985] R. E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Kupferman and Vardi, 1997] O. Kupferman and M. Vardi. Synthesis with incomplete information, 1997.

[Littman *et al.*, 1998] M. L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[Littman, 1997] M. L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 748–754, Menlo Park, July 1997. AAAI Press.

[Lozano and Balcázar, 1990] A. Lozano and J. L. Balcázar. The complexity of graph problems for succinctly represented graphs. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG'89*, number 411 in Lecture Notes in Computer Science, pages 277–286. Springer-Verlag, 1990.

[Madani *et al.*, 2003] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1–2):5–34, 2003.

[McAllester and Rosenblitt, 1991] D. A. McAllester and D. Rosenblitt. Systematic nonlinear planning. In T. L. Dean and K. McKeown, editors, *Proceedings of the 9th National Conference on Artificial Intelligence*, volume 2, pages 634–639. AAAI Press / The MIT Press, 1991.

[McDermott, 1999] D. V. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1–2):111–159, 1999.

[McMillan, 2003] K. L. McMillan. Interpolation and SAT-based model checking. In W. A. Hunt Jr. and F. Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, number 2725 in Lecture Notes in Computer Science, pages 1–13, 2003.

[Meyer and Stockmeyer, 1972] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society, 1972.

[Mneimneh and Sakallah, 2003] M. Mneimneh and K. Sakallah. Computing vertex eccentricity in exponentially large graphs: QBF formulation and solution. In E. Giunchiglia and A. Tacchella, editors, *SAT 2003 - Theory and Applications of Satisfiability Testing*, number 2919 in Lecture Notes in Computer Science, pages 411–425, 2003.

[Mundhenk *et al.*, 2000] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720, 2000.

[Nguyen *et al.*, 2002] X. Nguyen, S. Kambhampati, and R. S. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135:73–123, 2002.

[Papadimitriou and Yannakakis, 1986] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71:181–185, 1986.

[Papadimitriou, 1994] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[Pearl, 1984] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[Peot and Smith, 1992] M. A. Peot and D. E. Smith. Conditional nonlinear planning. In J. Hendler, editor, *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 189–197. Morgan Kaufmann Publishers, 1992.

[Pixley *et al.*, 1992] C. Pixley, S.-W. Jeong, and G. D. Hachtel. Exact calculation of synchronization sequences based on binary decision diagrams. In *Proceedings of the 29th Design Automation Conference*, pages 620–623, 1992.

[Pnueli and Rosner, 1989] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer-Verlag, July 1989.

[Pryor and Collins, 1996] L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[Puterman, 1994] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[Rintanen *et al.*, 2005] J. Rintanen, K. Heljanko, and I. Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. Report 216, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2005.

[Rintanen, 1998] J. Rintanen. A planning algorithm not based on directional search. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 617–624. Morgan Kaufmann Publishers, June 1998.

[Rintanen, 1999] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[Rintanen, 2004a] J. Rintanen. Complexity of planning with partial observability. In S. Zilberstein, J. Koehler, and S. Koenig, editors, *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 345–354. AAAI Press, 2004.

[Rintanen, 2004b] J. Rintanen. Distance estimates for planning in the discrete belief space. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004) and the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI-2004)*, pages 525–530. AAAI Press, 2004.

[Rintanen, 2004c] J. Rintanen. Evaluation strategies for planning as satisfiability. In R. López de Mántaras and L. Saitta, editors, *ECAI 2004: Proceedings of the 16th European Conference on Artificial Intelligence*, volume 110 of *Frontiers in Artificial Intelligence and Applications*, pages 682–687. IOS Press, 2004.

[Rintanen, 2004d] J. Rintanen. Phase transitions in classical planning: an experimental study. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004)*, pages 710–719. AAAI Press, 2004.

[Rintanen, 2005] J. Rintanen. Conditional planning in the discrete belief space. In L. P. Kaelbling, editor, *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1260–1265. Morgan Kaufmann Publishers, 2005.

[Rosenschein, 1981] S. J. Rosenschein. Plan synthesis: A logical perspective. In P. J. Hayes, editor, *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 331–337. William Kaufmann, August 1981.

[Sacerdoti, 1975] E. D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.

[Schoppers, 1987] M. J. Schoppers. Universal plans for real-time robots in unpredictable environments. In J. P. McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1039–1046. Morgan Kaufmann Publishers, 1987.

[Selman *et al.*, 1996] B. Selman, D. G. Mitchell, and H. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):459–465, 1996.

[Sheeran *et al.*, 2000] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In W. A. Hunt and S. D. Johnson, editors, *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2000.

[Smallwood and Sondik, 1973] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Smith and Weld, 1998] D. E. Smith and D. S. Weld. Conformant Graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 889–896. AAAI Press, 1998.

[Sondik, 1971] E. J. Sondik. *The optimal control of partially observable Markov processes*. PhD thesis, Stanford University, 1971.

[Sondik, 1978] E. J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: discounted costs. *Operations Research*, 26(2):282–304, 1978.

[Starke, 1991] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Journal of Mathematical Modelling and Simulation in Systems Analysis*, 8(4/5):293–303, 1991.

[Stockmeyer and Chandra, 1979] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.

[Turner, 2002] H. Turner. Polynomial-length planning spans the polynomial hierarchy. In *Logics in Artificial Intelligence, European Conference, JELIA 2002*, number 2424 in Lecture Notes in Computer Science, pages 111–124. Springer-Verlag, 2002.

[Valmari, 1991] A. Valmari. Stubborn sets for reduced state space generation. In G. Rozenberg, editor, *Advances in Petri Nets 1990. 10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany*, number 483 in Lecture Notes in Computer Science, pages 491–515. Springer-Verlag, 1991.

[van Beek and Chen, 1999] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 585–590. AAAI Press, 1999.

[Vardi and Stockmeyer, 1985] M. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 240–251. Association for Computing Machinery, 1985.

[Vardi, 1995] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In P. Wolper, editor, *Computer Aided Verification, Proceedings of the 7th International Conference*, volume 939 of *Lecture Notes in Computer Science*, pages 267–278. Springer-Verlag, 1995.

[Wolfman and Weld, 1999] S. A. Wolfman and D. S. Weld. The LPSAT engine & its application to resource planning. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume I, pages 310–315. Morgan Kaufmann Publishers, 1999.

Index