Technical Report 220

State-Space Traversal Techniques for Planning

Jussi Rintanen
Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Georges-Köhler-Allee, 79110 Freiburg im Breisgau
Germany

July 13, 2005

Foreword

These notes are based on the AI planning lectures at the Albert-Ludwigs-Universität Freiburg. I would like to thank all the students who have participated in the planning course and given comments, pointed out errors, and suggested other improvements, including Slawomir Grzonka, Bernd Gutmann, Raimund Renner, Richard Schmidt, and Martin Wehrle. Also my teaching and research assistants Marco Ragni and Dr. Markus Büttner have provided valuable feedback.

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.

To reason about actions, action sequences and plans it is necessary to model the dynamics of the world. Depending on the form of planning different kinds of world models are used. For more physical planning tasks, like path and motion planning, representing the quantitative geometric and physical properties of the world is necessary.

For planning at a more abstract level, with the physical properties of the world abstracted away, the world can be represented in terms of the individual facts that hold and are relevant for the planning task at hand. In this lecture the smallest components of the world are modeled as *state variables*, and individual world states are modeled as *valuations* of state variables. An individual state variable could for example indicate the location of an object, and for applications for which only the locations of certain objects are relevant, a state would be unambiguously characterized by the locations of all the objects.

As actions change the state of the world and the world is modeled in terms of state variables, actions are most naturally modeled as objects that change the values of the state variables.

1.1 Early research on AI planning

Research that has lead to current AI planning started in the 1960's in the form of programs that tried to simulate problem solving abilities of human beings. One of the first programs of this kind was the General Problem Solver (GPS) by Newell and Simon [Ernst *et al.*, 1969]. GPS performed

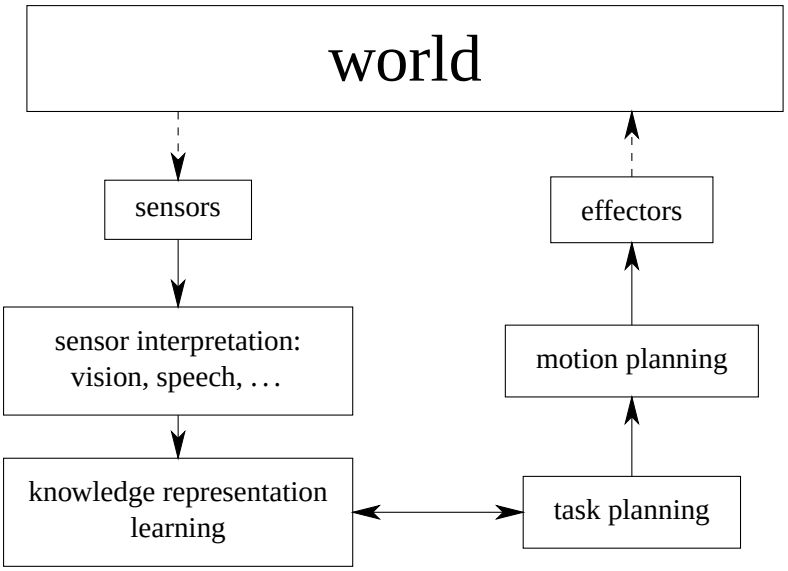


Figure 1.1: Software architecture of an intelligent robot

state space search guided by estimated differences between the current state and the goal states.

At the end of 1960's Green proposed the use of theorem-provers for constructing plans [Green, 1969]. However, because of the immaturity of theorem-proving techniques at that time, this approach was soon mostly abandoned in favor of specialized planning algorithms. There was theoretically oriented work on deductive planning which used different kinds of modal and dynamic logics [Rosenschein, 1981] but these works had little impact on the development of efficient planning algorithms. Deductive and logic-based approaches to planning gained popularity again only at the end of the 1990's as a consequence of the development of more sophisticated programs for the satisfiability problem of the classical propositional logic [Kautz and Selman, 1996].

One of the most well known early planning systems is the STRIPS planner from the beginning of the 1970's [Fikes and Nilsson, 1971]. The states in STRIPS are sets of formulae, and the operators change these state descriptions by adding and deleting formulae in the sets. Heuristics similar to the ones used in the GPS system were used in guiding the search. The definition of operators, with a *precondition* as well as *add* and *delete* lists, corresponding to the facts that respectively become true and false, and the associated terminology, is still in common use, although restricted to atomic facts, that is, the add list is simply the set of state variables that the action makes true, and the delete list similarly consists of the state variables that become false.

Starting in the mid 1970's the dominating approach to domain-independent planning was the so-called partial-order, or causal link, or nonlinear planning [Sacerdoti, 1975; McAllester and Rosenblitt, 1991], which remained popular until the mid-1990's and the introduction of the Graphplan planner [Blum and Furst, 1997] which started the shift away from partial-order planning to types of algorithms that had earlier been considered infeasible, even the then-notorious total-order planners. The basic idea of partial-order planning is that a plan is incrementally constructed starting from the initial state and the goals, by either adding an action to the plan so that one of the open goals or operator preconditions is fulfilled, or adding an ordering constraint on operators already

in the plan in order to resolve a potential conflict between them. In contrast to the forward or backward search strategies in Chapter 3 partial-order planners tried to avoid unnecessarily imposing an ordering on operators. The main advantages of both partial-order planners and Graphplan are present in the SAT/CSP approach to planning which is discussed in Section 3.6.

In parallel to partial-order planning, the notion of hierarchical planning emerged [Sacerdoti, 1974], and it has been deployed in many real-world applications. The idea in hierarchical planning is that the problem description imposes a structure on solutions and restricts the number of choices the planning algorithm has to make. A hierarchical plan consists of a main task which is decomposed to smaller tasks which are recursively solved. For each task there is a choice between solution methods. The less choice there is, the more efficiently the problem is solved. Furthermore, many hierarchical planners allow the embedding of problem-specific heuristics and problem-solvers to further speed up planning.

A collection of articles on AI planning starting from the late 1960's has been edited by Allen et al. [1990]. Many of the papers are mainly of historical interest, and some of them outline ideas that are still in use.

1.2 Overview

This lecture gives an overview of some of the main techniques currently used in AI planning for finding action sequences. In the most basic planning problem with one initial state and deterministic actions, often called *classical planning*, an algorithm that finds a sequence of actions from the unique initial state to a goal state solves the planning problem. For more general forms of planning, for example with nondeterministic actions, the required algorithms are far more complicated, but many of the techniques introduced for classical planning can be generalized and used as an implementation technique for subprocedures in the planning algorithms.

Chapter 2 presents a framework for formalizing different kinds of planning problems. Section 2.1 introduces the basic transition system model and its restriction to classical/deterministic problems that are the topic of Chapter 3. Section 2.3 introduces a representation of transition systems that is based on state variables and operators. This representation is used in Chapter 3 and it is closely related to the representations used in most planning research. With state variables and operators very big transition systems can be represented compactly.

In Chapter 3 the emphasis is on two main strands of research that have emerged during the last ten years of research on classical planning.

In 1995 the Graphplan planner of Blum and Furst [1997] demonstrated the competitiveness of a constraint-based approach to classical planning, earlier proposed by Kautz and Selman [1992] in a slightly different framework. The idea is to pose planning as a form of a constraint satisfaction problem: find a plan and a corresponding state sequence consisting of n steps so that the first state is the initial state, the last state is a goal state, and the changes in state variable values between two consecutive time points correspond to the execution of some actions. The constraint-satisfaction viewpoint together with powerful techniques for pruning search trees were a big improvement over earlier planning algorithms.

Blum and Furst's planner was based on ad hoc search algorithm with backward chaining. Soon after the introduction of Graphplan Kautz and Selman [1996] showed that an equally efficient planner could be obtained by a translation of the planning problem into the propositional logic and using a general purpose satisfiability algorithm. Main benefits of this approach are the repre-

sentation of the planning problem as propositional formulae, which allows the use of many kinds of declarative information during the planning process, and the very fast development of more efficient satisfiability algorithms and their implementation in the last ten years, improving the problem-solving capability of this approach tremendously.

An important development outside the artificial intelligence is the emergence of verification techniques that generalize Kautz and Selman's [1992; 1996] idea to temporal logic model-checking [Biere *et al.*, 1999], making satisfiability testing a leading logic-based technique in model-checking and increasingly replacing techniques based on binary decision diagrams BDDs [Bryant, 1992; Burch *et al.*, 1994; Clarke *et al.*, 1994].

A completely different approach to planning gained popularity after the 1998 planning competition and the GPT planner by Bonet and Geffner [2001]. It is based on the use of general-purpose heuristic search algorithms combined with heuristics automatically derived from the descriptions of planning problems. Planners based on heuristic state-space search have shown very good performance in finding non-optimal solutions to the kind of problems considered in the planning competitions. The main benefit of the approach is the simplicity of the basic approach which allows a lot of flexibility in incorporating problem-specific heuristics and pruning techniques in the heuristic search algorithm.

In Chapter 4 we present generalizations of some of the logic-based techniques to nondeterministic planning, and give an overview of a general framework of reasoning about states, state sets and actions in the propositional logic, as first used in connection of model-checking in computer-aided verification since the early 1990's [Burch *et al.*, 1994; Clarke *et al.*, 1994] and recently as an implementation technique of algorithms for nondeterministic planning. These representations have been called *symbolic* in the verification community, in contrast to the more concrete enumerative representations used by model-checking algorithms that explicitly enumerate states in the state space. Most often these techniques have been used in connection with binary decision diagrams BDDs [Bryant, 1992] but the techniques are applicable to other classes of propositional formulae just as well.

Chapter 2

Background

In this chapter we define the formal machinery needed in the rest of the lecture for describing different planning problems and algorithms. We give the basic definitions related to the classical propositional logic, and the definition of the transition system model that is the basis of most work on planning and that are closely related to finite automata and transition systems in other areas of computer science.

Section 2.1 defines transition systems and Section 2.1.1 their restriction to deterministic actions and one initial state, which is used in classical/deterministic planning. The general transition system definition is given a representation in terms of state variables and operators in Section 2.3, and in Section 2.3.1 it is restricted to the classical/deterministic special case. This last definition of transition systems is extensively used in Chapter 3.

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

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. $spreimg_o(T) \subseteq preimg_o(T)$
2. $img_o(spreimg_o(T)) \subseteq T$
3. *If* $T \subseteq T'$ *then* $img_o(T) \subseteq img_o(T')$.
4. $preimg_o(T \cup T') = preimg_o(T) \cup preimg_o(T')$.
5. $s' \in img_o(s)$ *if and only if* $s \in preimg_o(s)$.

Proof:

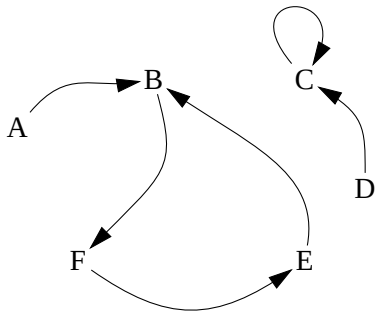
1. $spreimg_o(T) = \{s \in S | s' \in T, sos', img_o(s) \subseteq T\} \subseteq \{s \in S | s' \in T, sos'\} = \bigcup_{s' \in T} \{s \in S | sos'\} = \bigcup_{s' \in T} preimg_o(s') = preimg_o(T)$.
2. Take any $s' \in img_o(spreimg_o(T))$. Hence there is $s \in spreimg_o(T)$ so that sos'. As $s \in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

□

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*



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

Figure 2.1: The transition graph and the incidence matrix of a deterministic action

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 $app_{o_n}(\cdots app_{o_1}(s)\cdots)$.

2.1.2 Incidence matrices

Actions and other binary relations can be represented in terms of incidence matrices M (adjacency matrices) in which the element in row i and column j indicates whether a transition from state i to j is possible.

Figure 2.1 depicts the transition graph of an action and the corresponding incidence matrix. The action can be seen to be deterministic because for every state there is at most one arrow going out of it, and each row of the matrix contains at most one non-zero element.

For matrices $M_1, \ldots, M_n$ which represent the transition relations of actions $a_1, \ldots, a_n$ the combined transition relation is $M = M_1 + M_2 + \cdots + M_n$. The matrix M now tells whether a state can be reached from another state by at least one of the actions.

Here $+$ is the usual matrix addition that uses the Boolean addition for integers 0 and 1, which is defined as $0 + 0 = 0$, and $b + b' = 1$ if $b = 1$ or $b' = 1$. Boolean addition is used because in the presence of nondeterminism we could have 1 for both of two transitions from A to B and from A to C. For probabilistic planning problems normal addition is used and matrix elements are interpreted as probabilities of nondeterministic transitions.

The incidence matrix corresponding to first taking action a_1 and then a_2 is $M_1 M_2$. This is illustrated by Figure 2.2 The inner product of two vectors in the definition of matrix product corresponds to the reachability of a state from another state through all possible intermediate states.

Now we can compute for all pairs s, s' of states whether s' is reachable from s by a sequence of actions. Let M be the matrix that is the (Boolean) sum of the matrices of the individual actions.

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

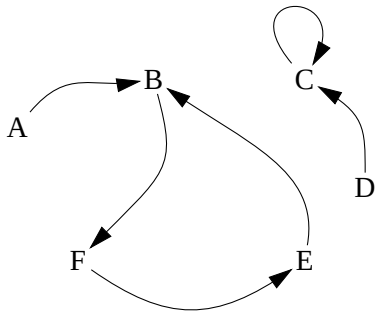
$\times$

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	1	0	0	0	0	0
D	0	0	0	1	0	0
E	0	0	0	0	1	0
F	0	0	0	1	0	0

$=$

	A	B	C	D	E	F
A	0	0	0	0	0	1
B	0	0	0	**1**	0	0
C	1	0	0	0	0	0
D	1	0	0	0	0	0
E	0	0	0	0	0	1
F	0	0	0	0	1	0

Figure 2.2: Matrix product corresponds to sequential composition.



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

Figure 2.3: A transition graph and the corresponding matrix M

Then define

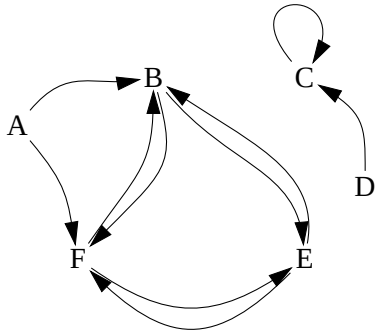
$$R_0 = I_{n\times n}$$
$$R_i = R_{i-1} + MR_{i-1} \text{ for } i \geq 1.$$

Here n is the number of states and $I_{n\times n}$ is the unit matrix of size n. By Tarski's fixpoint theorem $R_i = R_j$ for some $i \geq 0$ and all $j \geq i$ because of the monotonicity property that every element that is 1 for some i is 1 also for all $j > i$. Matrix $R_i = M^0 \cup M^1 \cup \cdots \cup M^i$ represents reachability by i actions or less.

2.2 Classical propositional logic

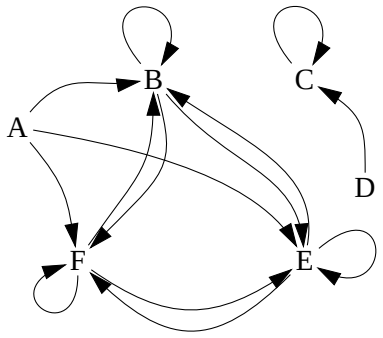
Let A be a set of propositional variables (atomic propositions). We define the set of propositional formulae inductively as follows.

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.



	A	B	C	D	E	F
A	0	1	0	0	0	1
B	0	0	0	0	1	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	1
F	0	1	0	0	1	0

Figure 2.4: A transition graph extended with composed paths of length 2 and the corresponding matrix $M + M^2$



	A	B	C	D	E	F
A	0	1	0	0	1	1
B	0	1	0	0	1	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	1	1
F	0	1	0	0	1	1

Figure 2.5: A transition graph extended with composed paths of length 3 and the corresponding matrix $M + M^2 + M^3$

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. ■

Notice 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.

set	formula
$T \cup U$	$T \vee U$
$T \cap U$	$T \wedge U$
$\overline{T}$	$\neg T$
$T \backslash U$	$T \wedge \neg U$
$\emptyset$	$\bot$
the universal set	$\top$

question about sets	question about formulae
$T \subseteq U$?	$\models T \rightarrow U$?
$T \subset U$?	$\models T \rightarrow U$ and $\not\models U \rightarrow T$?
$T = U$?	$\models T \leftrightarrow U$?

Table 2.1: Correspondence between set-theoretical and logical operations

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$\neq$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, that is, a function $s : A \rightarrow \{0, 1\}$.

Since we identify states with valuations of state variables, we can now identify sets of states with propositional formulae over the state variables. This allows us to perform set-theoretic operations on sets as logical operations and test relations between sets by inference in the propositional logic as summarized in Table 2.1

The actions of a succinct transition system are described by operators. An operator has two components. The precondition describes the set of states in which the action can be taken. The effect describes the successor states of each state in terms of the changes made to the values of the state variables.

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 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* $\Pi = \langle A, I, O, G, V\rangle$*, construct 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.

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.

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

■

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 2.2 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.*

$$c \rhd (e_1 \wedge \cdots \wedge e_n) \equiv (c \rhd e_1) \wedge \cdots \wedge (c \rhd e_n) \quad (2.1)$$
$$c_1 \rhd (c_2 \rhd e) \equiv (c_1 \wedge c_2) \rhd e \quad (2.2)$$
$$(c_1 \rhd e) \wedge (c_2 \rhd e) \equiv (c_1 \vee c_2) \rhd e \quad (2.3)$$
$$e \wedge (c \rhd e) \equiv e \quad (2.4)$$
$$e \equiv \top \rhd e \quad (2.5)$$
$$e_1 \wedge (e_2 \wedge e_3) \equiv (e_1 \wedge e_2) \wedge e_3 \quad (2.6)$$
$$e_1 \wedge e_2 \equiv e_2 \wedge e_1 \quad (2.7)$$
$$c \rhd \top \equiv \top \quad (2.8)$$
$$e \wedge \top \equiv e \quad (2.9)$$

Table 2.2: Equivalences on effects

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 2.2. 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. □

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

Table 2.3: Equivalences on nondeterministic effects

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 2.3.

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

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

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. □

Chapter 3

Deterministic planning

The simplest planning problems involves finding a sequence of actions that lead from a given initial state to a goal state. Only deterministic actions are considered. Determinism and the uniqueness of the initial state mean that the state of the transition system after executing any sequence of actions starting in the initial state is exactly predictable. This is not the case if actions are nondeterministic or there are several initial states. The problem instances in this chapter are deterministic succinct transition systems as defined in Section 2.3.1.

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, $e = e_1 \wedge \cdots \wedge e_n$:

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

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, $e = c \rhd e'$:

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

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. □

Notice that any operator $\langle c, e \rangle$ can be expressed in normal form in terms of $EPC_a(e)$ as

$$\left\langle c, \bigwedge_{a \in A} (EPC_a(e) \rhd a) \wedge (EPC_{\neg a}(e) \rhd \neg a) \right\rangle.$$

The formula $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$ 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 $s \models EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$ if and only if $s' \models a$.*

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

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

Case 2: Assume that $s \models a \wedge \neg EPC_{\neg a}(e)$. 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 $s \not\models EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$. Hence $s \models \neg EPC_a(e) \wedge (\neg a \vee EPC_{\neg a}(e))$.

Case 1: Assume that $s \models a$. Now $s \models EPC_{\neg a}(e)$ because $s \models \neg a \vee EPC_{\neg a}(e)$, 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 $s \models \neg EPC_a(e)$, 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 $EPC_l(e)$ 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,*

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

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 4.2 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 $\{a, \neg a\} \not\subseteq [e]^{det}_s$ 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 $app_o(s) \models \theta$. Hence $s \models \phi'_r$ iff $app_o(s) \models \phi'$ by the truth-definition of $\neg$.

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

Inductive case 3, $\phi' = \theta \wedge \theta'$: By the induction hypothesis $s \models \theta_r$ iff $app_o(s) \models \theta$, and $s \models \theta'_r$ iff $app_o(s) \models \theta'$. Hence $s \models \phi'_r$ iff $app_o(s) \models \phi'$ 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 $regr_{\langle b,c\rangle}(a) = a \wedge b$. 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 $regr_{o_k;\ldots;o_n}(G) \models regr_{o_{k+1};\ldots;o_n}(G)$ 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 $app_{o_{k+1};\ldots;o_n}(s) \models G$ for any s such that $s \models regr_{o_{k+1};\ldots;o_n}(G)$. Since $app_{o_1;\ldots;o_{k-1}}(I) \models regr_{o_k;\ldots;o_n}(G)$ and $regr_{o_k;\ldots;o_n}(G) \models regr_{o_{k+1};\ldots;o_n}(G)$ also $app_{o_1;\ldots;o_{k-1}}(I) \models regr_{o_{k+1};\ldots;o_n}(G)$. 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. $\mathit{regr}_{\langle\phi,a\rangle}(a)$ is satisfiable if and only if ϕ is satisfiable because $\mathit{regr}_{\langle\phi,a\rangle}(a) \equiv \phi$.

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 $\mathit{regr}_{\langle\phi\vee\neg a,b\rangle}(a \wedge b)$ 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 $\mathit{regr}_o(\phi) \not\models \phi$ or that $\mathit{regr}_o(\phi)$ 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 lecture.

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 $\mathit{EPC}_a(e_2) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e_2))$. Condition for this formula to be true after o_1 is obtained by regressing with e_1, leading to

$$\begin{aligned}
&\mathit{regr}_{e_1}(\mathit{EPC}_a(e_2) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e_2)))\\
&= \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)) \vee (\mathit{regr}_{e_1}(a) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)))\\
&= \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)) \vee ((\mathit{EPC}_a(e_1) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e_2))) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2))).
\end{aligned}$$

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

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

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*

$$\left\langle c, \bigwedge_{a\in A} \begin{pmatrix} ((\mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)) \vee (\mathit{EPC}_a(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)))) \rhd a) \wedge \\ ((\mathit{regr}_{e_1}(\mathit{EPC}_{\neg a}(e_2)) \vee (\mathit{EPC}_{\neg a}(e_1) \wedge \neg \mathit{regr}_{e_1}(\mathit{EPC}_a(e_2)))) \rhd \neg a) \end{pmatrix} \right\rangle$$

where $c = c_1 \wedge \mathit{regr}_{e_1}(c_2) \wedge \bigwedge_{a\in A} \neg(\mathit{EPC}_a(e_1) \wedge \mathit{EPC}_{\neg a}(e_1))$.

Notice 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 $a \in A$ $s \models regr_{e_1}(EPC_a(e_2)) \vee (EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2)))$, that is, $a \in [o_1 \circ o_2]_s^{det}$. If $s \models regr_{e_1}(EPC_a(e_2))$ then $s \not\models regr_{e_1}(EPC_{\neg a}(e_2)) \vee \neg regr_{e_1}(EPC_a(e_2))$. Otherwise $s \models EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2))$ and hence $s \not\models EPC_{\neg a}(e_1)$. Hence in both cases $s \not\models regr_{e_1}(EPC_{\neg a}(e_2)) \vee (EPC_{\neg a}(e_1) \wedge \neg regr_{e_1}(EPC_a(e_2)))$, that is, $\neg a \notin [o_1 \circ o_2]_s^{det}$. Therefore $app_{o_1 \circ o_2}(s)$ is defined.

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

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

   If $s \models regr_{e_1}(EPC_a(e_2))$ then by Theorem 3.7 and Lemma 3.3 $a \in [e_1]^{det}_{app_{o_1}(s)}$. Hence $app_{o_1;o_2}(s) \models a$.

   Assume $s \models EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2))$. Hence by Lemma 3.3 $a \in [e_1]^{det}_s$ and $app_{o_1}(s) \models a$, and $app_{o_1}(s) \not\models EPC_{\neg a}(e_2)$ and $\neg a \notin [e_2]^{det}_{app_{o_1}(s)}$. Hence $app_{o_1;o_2}(s) \models a$.

2. Assume $s \models a$ and $s \not\models regr_{e_1}(EPC_{\neg a}(e_2)) \vee (EPC_{\neg a}(e_1) \wedge \neg regr_{e_1}(EPC_a(e_2)))$.

   Since $s \not\models regr_{e_1}(EPC_{\neg a}(e_2))$ by Theorem 3.7 $app_{o_1}(s) \not\models EPC_{\neg a}(e_2)$ and hence $\neg a \notin [e_2]^{det}_{app_{o_1}(s)}$.

   Since $s \not\models EPC_{\neg a}(e_1) \wedge \neg regr_{e_1}(EPC_a(e_2))$ by Lemma 3.3 $\neg a \notin [e_1]^{det}_s$ or $app_{e_1}(s) \models EPC_a(e_2)$ and hence by Theorem 3.7 $a \in [e_2]^{det}_{app_{o_1}(s)}$.

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

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

1. If $a \in [e_2]^{det}_{app_{o_1}(s)}$ then by Lemma 3.3 $app_{o_1}(s) \models EPC_a(e_2)$. By Theorem 3.7 $s \models regr_{e_1}(EPC_a(e_2))$.

2. If $a \in [e_1]_s^{det}$ and $\neg a \notin [e_2]_{app_{o_1}(s)}^{det}$ then by Lemma 3.3 $app_{o_1}(s) \not\models EPC_{\neg a}(e_2)$. By Theorem 3.7 $s \models EPC_a(e_1) \wedge \neg regr_{e_1}(EPC_{\neg a}(e_2))$.
3. If $s \models a$ and $\neg a \notin [e_2]_{app_{o_1}(s)}^{det}$ and $\neg a \notin [e_1]_s^{det}$ then by Lemma 3.3 $app_{o_1}(s) \not\models EPC_{\neg a}(e_2)$. By Theorem 3.7 $s \not\models regr_{e_1}(EPC_{\neg a}(e_2))$.

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

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

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.*

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

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.

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

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.

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.*

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

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 asat(D, ϕ) 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 asat(D, ϕ).

The function asat(D, ϕ) 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.

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

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 $\text{asat}(D, \phi)$ 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, $\phi = \top$: $\text{asat}(D, \top)$ 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 $\text{asat}(D, a)$ 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 $\text{asat}(D, \neg a)$ 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 $\text{asat}(D, \phi_1)$ and $\text{asat}(D, \phi_2)$ returns true. Hence $\text{asat}(D, \phi_1 \vee \phi_2)$ 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 $\text{asat}(D, \phi_1)$ and $\text{asat}(D, \phi_2)$ return true. Hence $\text{asat}(D, \phi_1 \wedge \phi_2)$ 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 $\text{asat}(\emptyset, a \wedge \neg a)$ 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: $\text{asat}(D, \phi)$ 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.*

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

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

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

Notice 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 asat(D, ϕ) 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 asat(D, ϕ). 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 $D_{i-1}^{max} \backslash \{l \in L | o \in O, \text{asat}(D_{i-1}^{max}, \mathit{EPC}_{\bar{l}}(o))\} \subseteq D_{i-1}^+ \backslash \{l \in L | o \in O, \text{cost}(\mathit{EPC}_{\bar{l}}(o), i) < i\}$. By the induction hypothesis $D_{i-1}^{max} \subseteq D_{i-1}^+$. It is sufficient to show that cost$(\mathit{EPC}_{\bar{l}}(o), i) < i$ implies 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: cost$(\phi, i) < i$ implies asat(D_{i-1}^{max}, ϕ)=true.

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

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

Base case 3, $\phi = a$: If 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 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 cost$(\phi_1 \vee \phi_2, i) < i$. Since cost$(\phi_1 \vee \phi_2, i) = \min(\text{cost}(\phi_1, i), \text{cost}(\phi_2, i))$, either cost$(\phi_1, i) < i$ or cost$(\phi_2, i) < i$. By the induction hypothesis cost$(\phi_1, i) < i$ implies asat(D_{i-1}^{max}, ϕ_1), and cost$(\phi_2, i) < i$ implies asat(D_{i-1}^{max}, ϕ_2). Hence either asat(D_{i-1}^{max}, ϕ_1) or asat(D_{i-1}^{max}, ϕ_2). Therefore by definition asat$(D_{i-1}^{max}, \phi_1 \vee \phi_2)$.

Inductive case 6, $\phi = \phi_1 \wedge \phi_2$: Assume cost$(\phi_1 \wedge \phi_2, i) < i$. Since $i \geq 1$ and cost$(\phi_1 \vee \phi_2, i) =$ cost(ϕ_1, i) + cost(ϕ_2, i), both cost$(\phi_1, i) < i$ and cost$(\phi_2, i) < i$. By the induction hypothesis cost$(\phi_1, i) < i$ implies asat(D_{i-1}^{max}, ϕ_1), and cost$(\phi_2, i) < i$ implies asat(D_{i-1}^{max}, ϕ_2). Hence both asat(D_{i-1}^{max}, ϕ_1) an asat(D_{i-1}^{max}, ϕ_2). Therefore by definition asat$(D_{i-1}^{max}, \phi_1 \wedge \phi_2)$.

Inductive case 7, $\phi = \neg\neg\phi_1$: By the induction hypothesis cost$(\phi_1, i) < i$ implies asat(D_{i-1}^{max}, ϕ_1). By definition cost$(\neg\neg\phi_1, i) =$ cost(ϕ_1, i) and asat$(D, \neg\neg\phi) =$ asat(D, ϕ). By the induction hypothesis cost$(\neg\neg\phi_1, i) < i$ implies 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 distance 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. Notice 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.*

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

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.

□

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 $\delta_I^{max}(G) = \infty$.

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

Figure 3.1: Algorithm for finding a relaxed plan

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$(D_{i-1}^{max}, EPC_l(o))$. Hence there is $T_i \subseteq O$ so that $N_i \subseteq \{l \in L | o \in T_i, \text{asat}(D_{i-1}^{max}, EPC_l(o))\}$ 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 $G_i = \bigwedge_{l \in N_{i+1}} \bigvee \{EPC_l(o) | o \in T_{i+1}\}$. By definition of T_{i+1} for every $l \in N_{i+1}$ there is $o \in T_{i+1}$ such that $\text{asat}(D_i^{max}, EPC_l(o)) = \text{true}$. By definition of $\text{asat}(D_i^{max}, \phi_1 \vee \phi_2)$ and $\text{asat}(D_i^{max}, \phi_1 \wedge \phi_2)$ for ϕ_1 and ϕ_2 also $\text{asat}(D_i^{max}, G_i) = \text{true}$.

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

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.

[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

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}.

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.

```
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

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 $\overline{l} \in [e]_s^{det}$ and o is applicable in s by Lemma 3.3 $s \models EPC_{\overline{l}}(o)$. 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 $s \models C \cup \{EPC_{\overline{l}}(o)\}$ 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 $C \cup \{EPC_{\overline{l}}(o)\} \not\models \neg EPC_{\overline{l'}}(o)$ 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 $C \cup \{EPC_{\overline{l}}(o)\} \not\models l'$ 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*. □

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 4.2, 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*∧*BonC*. Regression with the operator that moves B onto C from the table yields *AonB* ∧ *Bclear* ∧ *Cclear* ∧ *BonT*. This formula does not correspond to an intended blocks world state because *AonB* is incompatible with *Bclear*, and indeed, ¬*AonB* ∨ ¬*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 $\tau_A(o)$ that represents operator $o = \langle c, e\rangle$ is defined by*

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

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

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

■

Lemma 3.42 *Let s and s' be states and o an operator. Let $v : A \cup A' \to \{0, 1\}$ 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 $s' = \mathit{app}_o(s)$.

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

Assume $s' = \mathit{app}_o(s)$. Since s' is defined, $v \models \tau_A(o)$ and $v \models \bigwedge_{a\in A} \neg(\mathit{EPC}_a(e) \wedge \mathit{EPC}_{\neg a}(e))$. By Lemma 3.4 $v \models \mathit{EPC}_a(e) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e))$ 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. Notice 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 3.7.

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. ■

3.6.4 Parallel application of operators

For states s and sets T of operators we define $app_T(s)$ as the result of simultaneously applying all operators $o \in T$: the preconditions of all operators in T must be true in s and the state $app_T(s)$ is obtained from s by making the literals in $\bigcup_{\langle p,e\rangle \in T}[e]^{det}_s$ true. Analogously to sequential plans we can define $app_{T_1;T_2;\ldots;T_n}(s)$ as $app_{T_n}(\cdots app_{T_2}(app_{T_1}(s))\cdots)$.

Next we show how the translation of deterministic operators into the propositional logic in Section 3.6.2 can be extended to the simultaneous application of operators as in $app_T(s)$.

Consider the formula $\tau_A(o)$ representing one operator $o = \langle c, e\rangle$.

$$
c \wedge \bigwedge_{a\in A}((EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))) \leftrightarrow a') \wedge \bigwedge_{a\in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e)).
$$

This formula can be rewritten to the following logically equivalent formula that separately says which state variables are changed by the operator and which state variables retain their values.

$$
\begin{array}{l}
c\wedge \\
\bigwedge_{a\in A}(EPC_a(e) \rightarrow a')\wedge \\
\bigwedge_{a\in A}(EPC_{\neg a}(e) \rightarrow \neg a')\wedge \\
\bigwedge_{a\in A}((a \wedge \neg a') \rightarrow EPC_{\neg a}(e))\wedge \\
\bigwedge_{a\in A}((\neg a \wedge a') \rightarrow EPC_a(e))
\end{array}
$$

We use this formulation of $\tau_A(o)$ as basis of obtaining encodings of planning that allow *several operators in parallel*. Every operator applied at a given time point causes its effects to be true and requires its precondition to be true. This is expressed by the first three conjuncts. The last two conjuncts say that, assuming the operator that is applied is the only one, certain state variables retain their value. These formulae have to be modified to accommodate the possibility of executing several operators in parallel.

We introduce propositional variables o for denoting the execution of operators $o \in O$.

Definition 3.47 *Let A be the set of state variables and O a set of operators. Let the formula $\tau_A(O)$ denote the conjunction of formulae*

$$(o \rightarrow c) \wedge$$
$$\bigwedge_{a \in A} (o \wedge \mathit{EPC}_a(e) \rightarrow a') \wedge$$
$$\bigwedge_{a \in A} (o \wedge \mathit{EPC}_{\neg a}(e) \rightarrow \neg a')$$

for all $\langle c, e \rangle \in O$ and

$$\bigwedge_{a \in A} ((a \wedge \neg a') \rightarrow ((o_1 \wedge \mathit{EPC}_{\neg a}(e_1)) \vee \cdots \vee (o_n \wedge \mathit{EPC}_{\neg a}(e_n))) \wedge$$
$$\bigwedge_{a \in A} ((\neg a \wedge a') \rightarrow ((o_1 \wedge \mathit{EPC}_a(e_1)) \vee \cdots \vee (o_n \wedge \mathit{EPC}_a(e_n))))$$

where $O = \{o_1, \ldots, o_n\}$ and $e_1, \ldots, e_n$ are the respective effects.

The difference to the definition of $\tau_A(o)$ in Section 3.6.2 is that above the formulae do not assume that there is only one operator explaining the changes that take place.

The formula $\tau_A(O)$ matches the definition of $\mathit{app}_T(s)$.

Lemma 3.48 *Let s and s' be states and O and $T \subseteq O$ sets of operators. Let $v : A \cup A' \cup O \rightarrow \{0, 1\}$ be a valuation such that*

1. *for all $o \in O$, $v(o) = 1$ iff $o \in T$,*
2. *for all $a \in A$, $v(a) = s(a)$, and*
3. *for all $a \in A$, $v(a') = s'(a)$.*

Then $v \models \tau_A(O)$ if and only if $s' = \mathit{app}_T(s)$.

Proof: For the proof from right to left we assume that $s' = \mathit{app}_T(s)$ and show that $v \models \tau_A(O)$.

For the formulae $o \rightarrow c$ consider any $o = \langle c, e \rangle \in O$. If $o \notin T$ then $v \not\models o$ and $v \models o \rightarrow c$. So assume $o \in T$. By assumption s is a state such that $\mathit{app}_T(s)$ is defined. Hence $s \models c$. Hence $v \models o \rightarrow c$.

For the formulae $o \wedge \mathit{EPC}_a(e) \rightarrow a'$ consider any $o = \langle c, e \rangle \in O$. If $o \notin T$ then $v \not\models o$ and $v \models o \wedge \mathit{EPC}_l(e) \rightarrow l$ for all literals l. So assume $o \in T$. Now $v \models o \wedge \mathit{EPC}_l(e) \rightarrow l$ because if $s \models \mathit{EPC}_l(e)$ then $l \in [e]_s^{det}$ by Lemma 3.3 and $s' \models l$. Proof for $o \wedge \mathit{EPC}_{\neg a}(e) \rightarrow \neg a'$ is analogous.

For the formulae $((a \wedge \neg a') \rightarrow ((o_1 \wedge \mathit{EPC}_{\neg a}(e_1)) \vee \cdots \vee (o_n \wedge \mathit{EPC}_{\neg a}(e_n))$ consider any $a \in A$. According to the definition of $s' = \mathit{app}_T(s)$, a can be true in s and false in s' only if $\neg a \in [o]_s^{det}$ for some $o \in T$. By Lemma 3.3 $\neg a \in [o]_s^{det}$ if and only if $s \models \mathit{EPC}_{\neg a}(o)$. So if the antecedent of $(a \wedge \neg a') \rightarrow ((o_1 \wedge \mathit{EPC}_{\neg a}(o_1)) \vee \cdots \vee (o_m \wedge \mathit{EPC}_{\neg a}(o_m)))$ is true, then one of the disjuncts of the consequent is true, where $O = \{o_1, \ldots, o_m\}$. The proof for the change from false to true is analogous.

For the proof from left to right we assume $v \models \tau_A(O)$ and show that $s' = \mathit{app}_T(s)$.

The precondition c of every $o \in T$ is true in s because $v \models o$ and $v \models o \rightarrow c$, and $s' \models [e]_s^{det}$ for every $o = \langle c, e \rangle \in T$ because $v \models o$ and $v \models o \wedge \mathit{EPC}_l(e) \rightarrow l$ for every literal l. This also means that $[T]_s^{det}$ is consistent and $\mathit{app}_T(s)$ is defined.

For state variables a not occurring in $[T]_s^{det}$ we have to show that $s(a) = s'(a)$. Since a does not occur in $[T]_s^{det}$, for every $o \in \{o_1, \ldots, o_m\} = O = \{\langle c_1, e_1 \rangle, \ldots, \langle c_m, e_m \rangle\}$ either $o \notin T$ or both

$a \notin [e]_s^{det}$ and $\neg a \notin [e]_s^{det}$. Hence either $v \not\models o$ or (by Lemma 3.3) $v \models \neg(EPC_a(e)) \wedge \neg EPC_{\neg a}(e)$. This together with the assumptions that $v \models (a \wedge \neg a') \rightarrow ((o_1 \wedge EPC_{\neg a}(e_1)) \vee \cdots \vee (o_m \wedge EPC_{\neg a}(e_m)))$ and $v \models (\neg a \wedge a') \rightarrow ((o_1 \wedge EPC_a(o_1)) \vee \cdots \vee (o_m \wedge EPC_a(e_m)))$ implies $v \models (a \rightarrow a') \wedge (\neg a \rightarrow \neg a')$. Therefore every $a \in A$ not occurring in $[T]_s^{det}$ remains unchanged. Hence $s' = app_T(s)$. □

Example 3.49 Let $o_1 = \langle \neg LAMP1, LAMP1 \rangle$ and $o_2 = \langle \neg LAMP2, LAMP2 \rangle$. The application of none, one or both of these operators is described by the following formula.

$$
\begin{array}{l}
(\neg LAMP1 \wedge LAMP1') \rightarrow ((o_1 \wedge \top) \vee (o_2 \wedge \bot)) \\
(LAMP1 \wedge \neg LAMP1') \rightarrow ((o_1 \wedge \bot) \vee (o_2 \wedge \bot)) \\
(\neg LAMP2 \wedge LAMP2') \rightarrow ((o_1 \wedge \bot) \vee (o_2 \wedge \top)) \\
(LAMP2 \wedge \neg LAMP2') \rightarrow ((o_1 \wedge \bot) \vee (o_2 \wedge \bot)) \\
o_1 \rightarrow LAMP1' \\
o_1 \rightarrow \neg LAMP1 \\
o_2 \rightarrow LAMP2' \\
o_2 \rightarrow \neg LAMP2
\end{array}
$$

■

3.6.5 Partially-ordered 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.

In Section 3.6.4 we have shown how to represent the parallel application of operators in the propositional logic. However, this definition is too loose because it allows plans that cannot be executed.

Example 3.50 The operators $\langle a, \neg b \rangle$ and $\langle b, \neg a \rangle$ may be executed simultaneously resulting in a state satisfying $\neg a \wedge \neg b$, although this state is not reachable by the two operators sequentially. ■

A realistic way of interpreting parallelism in partially ordered plans is that any total ordering of the simultaneous operators is executable and results in the same state in all cases. This is the definition used in planning research so far.

Definition 3.51 (Step plans) *For a set of operators O and an initial state I,* a step plan for O and I *is a sequence $T = \langle T_0, \ldots, T_{l-1} \rangle$ 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. *$s_0 = I$,*
2. *for all $i \in \{0, \ldots, l-1\}$ and every total ordering $o_1, \ldots, o_n$ of T_i, $app_{o_1;\ldots;o_n}(s_i)$ is defined and equals s_{i+1}.*

Theorem 3.52 *Testing whether a sequence of sets of operators is a step plan is co-NP-hard.*

Proof: The proof is by reduction from the co-NP-complete validity problem TAUT. Let ϕ be any propositional formula. Let $A = \{a_1, \ldots, a_n\}$ be the set of propositional variables occurring in ϕ. Our set of state variables is A. Let $o_z = \langle \phi, \top \rangle$ and $O = \{\langle \top, a_1 \rangle, \ldots, \langle \top, a_n \rangle, o_z\}$. 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 $T = \langle O \rangle$ is a step plan for O and s.

Assume ϕ is a tautology. Now for any total ordering $o_0, \ldots, o_n$ of O 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 step plan.

Assume T is a step plan. Let v be any valuation. We show that $v \models \phi$. Let $O_v = \{\langle \top, a \rangle | a \in A, v \models a\}$. The operators O can be ordered to $o_0, \ldots, o_n$ so that the operators $O_v = \{o_0, \ldots, o_k\}$ precede o_z and $O \backslash (O_v \cup \{o_z\})$ follow o_z. Since T is a 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. □

To avoid intractability it is better to restrict to a class of step plans that are easy to recognize. One such class is based on the notion of *interference*.

Definition 3.53 (Affect) *Let A be a set of state variables and $o = \langle c, e \rangle$ and $o' = \langle c', e' \rangle$ operators over A. Then o* affects *o' if there is $a \in A$ such that*

1. *a is an atomic effect in e and a occurs in a formula in e' or it occurs negatively in c', or*
2. *$\neg a$ is an atomic effect in e and a occurs in a formula in e' or it occurs positively in c'.*

Definition 3.54 (Interference) *Operators o and o'* interfere *if o affects o' or o' affects o.*

Testing for interference of two operators is easy polynomial time computation. Non-interference not only guarantees that a set of operators is executable in any order, but it also guarantees that the result equals to applying all the operators simultaneously.

Lemma 3.55 *Let s be a state and T a set of operators so that $app_T(s)$ is defined and no two operators interfere. Then $app_T(s) = app_{o_1;\ldots;o_n}(s)$ for any total ordering $o_1, \ldots, o_n$ of T.*

Proof: Let $o_1, \ldots, o_n$ be any total ordering of T. 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 $[o_i]^{det}_s = [o_i]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$. Since o_i does not interfere with operators $o_{i+1}, \ldots, o_n$, no state variable occurring in $[o_i]^{det}_s$ occurs in an antecedent of a conditional effect or in the precondition of $o_{i+1}, \ldots, o_n$, that is, these state variables do not change. Since $[o_i]^{det}_s = [o_i]^{det}_{app_{o_1;\ldots;o_{i-1}}(s)}$ this also holds when o_i is applied in $app_{o_1;\ldots;o_{i-1}}(s)$. This completes the induction proof.

Since $app_T(s)$ is defined, the precondition of every $o \in T$ is true in s and $[o]^{det}_s$ is consistent. By the fact we established above, the precondition of every $o \in T$ 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 $\{o_1, \ldots, o_k\} \subseteq T \backslash \{o\}$. Hence any total ordering of the operators is executable. By the fact we established above, $[o]^{det}_s = [o]^{det}_{app_{o_1;\ldots;o_k}(s)}$ for every $\{o_1, \ldots, o_k\} \subseteq T \backslash \{o\}$. Hence every operator causes the same changes no matter what the total ordering is. Since $app_T(s)$ is defined, no operator in T undoes the effects of another operator. Hence the same state $s' = app_T(s)$ is reached in every case. □

For finding plans by using the translation of parallel actions from Section 3.6.4 it remains to encode the condition that no two parallel actions are allowed to interfere.

Definition 3.56 *Define*

$$\mathcal{R}_2(A, A', O) = \tau_A(O) \wedge \bigwedge \{\neg(o \wedge o') | \{o, o'\} \subseteq O, o \neq o', o \text{ and } o' \text{interfere}\}$$

Definition 3.57 *Let $\langle A, I, O, G\rangle$ be a deterministic succinct transition system. Define*

$$\Phi^{par}_n = \iota^0 \wedge \mathcal{R}_2(A^0, A^1, O^0) \wedge \mathcal{R}_2(A^1, A^2, O^1) \wedge \cdots \wedge \mathcal{R}_2(A^{n-1}, A^n, O^{n-1}) \wedge G^n$$

where $A^i = \{a^i | a \in A\}$ for all $i \in \{0, \ldots, n\}$ and $O^i = \{o^i | o \in O\}$ for all $i \in \{1, \ldots, n\}$ and $\iota^0 = \bigwedge\{a^0 | a \in A, I(a) = 1\} \cup \{\neg a^0 | a \in A, I(a) = 0\}$ and G^n is G with every $a \in A$ replaced by a^n.

If Φ^{par}_n is satisfiable and v is a valuation such that $v \models \Phi^{par}_n$, then define $T_i = \{o \in O | v \models o^i\}$ for every $i \in \{1, \ldots, n\}$. Then $\langle T_1, \ldots, T_n\rangle$ is a plan for the transition system, that is, $app_{T_1;\ldots;T_n}(I) \models G$.

It may be tempting to think that non-interference implies that the actions occurring in parallel in a plan could always be executed simultaneously in the real world. This however is not the case. For genuine temporal parallelism the formalization of problems as operators has to fulfill much stronger criteria than when sequential execution is assumed.

Example 3.58 Consider the operators

$$\begin{aligned}\text{transport-A-with-truck-1} &= \langle \text{AinFreiburg}, \text{AinStuttgart} \wedge \neg\text{AinFreiburg}\rangle \\ \text{transport-B-with-truck-1} &= \langle \text{BinFreiburg}, \text{BinKarlsruhe} \wedge \neg\text{BinFreiburg}\rangle\end{aligned}$$

which formalize the transportation of two objects with one vehicle. The operators do not interfere, and our notion of plans allows the simultaneous execution of these operators. However, these actions cannot really be simultaneous because the corresponding real world actions involve the same vehicle going to different destinations. ■

3.7 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. There are also other notions of parallel plans that may lead to much more efficient planning [Rintanen *et al.*, 2005]. 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 with-

out 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 [2004c] 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

Extensions to nondeterministic planning

The techniques discussed in Chapter 3 can be generalized to nondeterministic conditional planning problems.

Deterministic planning is the problem of finding a path from the initial state to any of the goal states. This problem is also implicitly a subproblem in more general nondeterministic planning problems [Howard, 1960; Puterman, 1994; Boutilier *et al.*, 1999; Bonet and Geffner, 2000; Cimatti *et al.*, 2003; Smallwood and Sondik, 1973; Kaelbling *et al.*, 1998; Madani *et al.*, 2003] and the techniques in the previous chapter can be helpful in solving them as well. Nondeterministic planning problems fundamentally differ from the basic deterministic planning problem, for example by being provably exponentially more difficult [Rintanen, 2004a]. These techniques should therefore only be viewed as *implementation techniques*. Algorithms for the more general problems fundamentally differ from those for deterministic planning.

Planning without observability can be viewed as a path existence problem similarly to the classical deterministic planning problem. As there may be several initial states and one state may have several successors, there may be several possible states at any step of plan execution. These state sets are known as *belief states*. For nondeterministic problems without observability planning can be formalized as finding a path in the space of belief states. In this setting the problem of computing the successor or the predecessors of a belief state with respect to an operator arises. The techniques discussed in this chapter may be used for representing belief states and computing their successors and predecessors, and can also be applied for conditional planning with partial and full observability.

Some algorithms for conditional planning (with full or partial observability) involve testing whether the current (incomplete) candidate plan can reach the goal states or whether it can also reach states that are not goal states. This question can be answered by techniques that extend those given for deterministic planning in Section 3.6.

4.1 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.

4.1.1 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 4.1 (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 4.2 *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 4.3 Let $o = \langle d, (b | \neg c) \rangle$. Then

$$\begin{aligned} \mathit{regr}_o^{nd}(b \leftrightarrow c) &= \mathit{regr}_{\langle d,b\rangle}(b \leftrightarrow c) \wedge \mathit{regr}_{\langle d,\neg c\rangle}(b \leftrightarrow c) \\ &= (d \wedge (\top \leftrightarrow c)) \wedge (d \wedge (b \leftrightarrow \bot)) \\ &\equiv d \wedge c \wedge \neg b. \end{aligned}$$

■

4.1.2 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 4.4 *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 2.3 to satisfy Assumption 4.4: $((d \rhd a)|b) \wedge ((\neg d \rhd \neg a)|c)$ 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.

$$\begin{aligned}
\tau_B^{nd}(e) &= \tau_B(e) \text{ when } e \text{ is deterministic} \\
\tau_B^{nd}(e_1|\cdots|e_n) &= \tau_B^{nd}(e_1) \vee \cdots \vee \tau_B^{nd}(e_n) \\
\tau_B^{nd}(e_1 \wedge \cdots \wedge e_n) &= \tau_{B\setminus(B_2\cup\cdots\cup B_n)}^{nd}(e_1) \wedge \tau_{B_2}^{nd}(e_2) \wedge \cdots \wedge \tau_{B_n}^{nd}(e_n) \\
&\quad \text{where } B_i = \mathit{changes}(e_i) \text{ for all } i \in \{2,\ldots,n\}
\end{aligned}$$

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 4.4 guarantees that for each $\tau_B^{nd}(e)$ all state variables changed by e are in B.

Example 4.5 We translate the effect

$$e = (a|(d \rhd a)) \wedge (c|d)$$

into a propositional formula. The set of state variables is $A = \{a, b, c, d\}$.

$$\begin{aligned}
\tau_{\{a,b,c,d\}}^{nd}(e) &= \tau_{\{a,b\}}^{nd}(a|(d \rhd a)) \wedge \tau_{\{c,d\}}^{nd}(c|d) \\
&= (\tau_{\{a,b\}}^{nd}(a) \vee \tau_{\{a,b\}}^{nd}(d \rhd a)) \wedge (\tau_{\{c,d\}}^{nd}(c) \vee \tau_{\{c,d\}}^{nd}(d)) \\
&= ((a' \wedge (b \leftrightarrow b')) \vee (((a \vee d) \leftrightarrow a') \wedge (b \leftrightarrow b'))) \wedge \\
&\quad ((c' \wedge (d \leftrightarrow d')) \vee ((c \leftrightarrow c') \wedge d'))
\end{aligned}$$

■

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 4.6 *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 4.7 *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 $v \models \tau^{nd}_A(o)$. Let $o = \langle c, e\rangle$. Since $\mathit{img}_o(s)$ is non-empty, $s \models c$. It remains to show that $v \models \tau^{nd}_A(e)$.

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 $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

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 $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

Inductive case 1, $e = e_1 \wedge \cdots \wedge e_n$: By definition $\tau^{nd}_B(e_1 \wedge \cdots \wedge e_n) = \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_1) \wedge \tau^{nd}_{B_2}(e_2) \wedge \cdots \wedge \tau^{nd}_{B_n}(e_n)$ 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 4.4 $s(a) = s'(a)$ 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 $s \cup s'[A'/A] \models \tau^{nd}_{B_i}(e_i)$ for all $i \in \{2, \ldots, n\}$ and $s \cup s'[A'/A] \models \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_i)$, and therefore $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

Inductive case 2, $e = e_1|\cdots|e_n$: By definition $\tau^{nd}_B(e_1|\cdots|e_n) = \tau^{nd}_B(e_1) \vee \cdots \vee \tau^{nd}_B(e_n)$. 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 $s \cup s'[A'/A] \models \tau^{nd}_B(e_i)$. As $\tau^{nd}_B(e_i)$ is one of the disjuncts of $\tau^{nd}_B(e)$ finally $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

For the proof from left to right to right assume that $v \models \tau^{nd}_B(e)$ 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 $v \models \tau^{nd}_B(e)$ 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 4.4 sets of the state variables occurring in $e_1, \ldots, e_n$ are disjoint. By definition $\tau^{nd}_B(e_1 \wedge \cdots \wedge e_n) = \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_1) \wedge \tau^{nd}_{B_2}(e_2) \wedge \cdots \wedge \tau^{nd}_{B_n}(e_n)$ 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 $v \models \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_1) \wedge \tau^{nd}_{B_2}(e_2) \wedge \cdots \wedge \tau^{nd}_{B_n}(e_n)$.

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 $\tau^{nd}_B(e_1|\cdots|e_n) = \tau^{nd}_B(e_1) \vee \cdots \vee \tau^{nd}_B(e_n)$. Because $v \models \tau^{nd}_B(e_1|\cdots|e_n)$, $v \models \tau^{nd}_B(e_i)$ 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 $s' \in \mathit{img}_o(s)$. □

4.2 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.

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.

4.2.1 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 4.8 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 4.9 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

$$\exists B.\phi = \exists b_1.(\exists b_2.(\ldots \exists b_n.\phi \ldots))$$
$$\forall B.\phi = \forall b_1.(\forall b_2.(\ldots \forall b_n.\phi \ldots)).$$

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 4.10 *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 4.1: 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. □

4.2.2 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 4.11 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 4.1 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 4.12 *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 4.13 *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 4.14 *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 $s'[A'/A] \models \exists A.(\phi \wedge \tau_A^{nd}(o))$ L4.12
iff there is valuation s of A such that $(s \cup s'[A'/A]) \models \phi \wedge \tau_A^{nd}(o)$ L4.10
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 $s' \in \mathit{img}_o(s)$ L4.7
iff $s' \in \mathit{img}_o(T)$.

Theorem 4.15 *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 $(s \cup s'_0) \models \tau_A^{nd}(o)$ L4.10
iff there is $s' : A \rightarrow \{0,1\}$ such that $s' \models \phi$ and $(s \cup s'_0) \models \tau_A^{nd}(o)$ L4.12
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 $s' \in \mathit{img}_o(s)$ L4.7
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 4.16 *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 $s \models \exists A'.\tau_A^{nd}(o)$ L4.10
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 $s \models \exists A'.\tau_A^{nd}(o)$ L4.12
iff $s' \notin \mathit{img}_o(s)$ or $s' \models \phi$ for all $s' : A \rightarrow \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L4.7
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 $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$ L4.10
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' : A \rightarrow \{0,1\}$ with $s' \in \mathit{img}_o(s)$ L4.7
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 4.17 *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 4.1.*

Proof: By Theorems 4.2 and 4.16 $\{s \in S | s \models regr_o(\phi)\} = spreimg_o(\{s \in S | s \models \phi\}) = \{s \in S | s \models spreimg_o(\phi)\}$. □

Example 4.18 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

$$\begin{aligned}
\exists a'b'c'.((a' \wedge b') \wedge \tau_A^{nd}(o)) &\equiv \exists a'b'c'.((a' \wedge b') \wedge c \wedge a' \wedge ((b \vee a) \leftrightarrow b') \wedge (c \leftrightarrow c')) \\
&\equiv \exists a'b'c'.(a' \wedge b' \wedge c \wedge (b \vee a) \wedge c') \\
&\equiv \exists b'c'.(b' \wedge c \wedge (b \vee a) \wedge c') \\
&\equiv \exists c'.(c \wedge (b \vee a) \wedge c') \\
&\equiv c \wedge (b \vee a)
\end{aligned}$$

■

Hence regression for nondeterministic operators (Definition 4.1) 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.

Notice 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.

4.2.3 An algorithm for constructing acyclic plans

Next we present an algorithm for constructing acyclic plans for nondeterministic problem with full observability. Acyclicity means that during any execution of the plan no state is visited more than once. Not all nondeterministic planning problems that have an intuitively acceptable solution have a solution as an acyclic plan. For a more detailed discussion of this topic and related algorithms see [Cimatti *et al.*, 2003].

The basic algorithm is for transition systems as in Definition 2.1 but the techniques in Section 4.2 can be directly applied to obtain a logic-based algorithm for succinct transition systems (Definition 2.8 in Section 2.3) that can be implemented easily by using any publicly available BDD package.

In the first phase the algorithm computes distances of the states. In the second phase the algorithm constructs a plan based on the distances.

Let G be a set of states and O a set of operators. Then we define the *backward distance sets* D_i^{bwd} for G, O that consist of those states for which there is a guarantee of reaching a state in G with at most i operator applications.

$$\begin{aligned} D_0^{bwd} &= G \\ D_i^{bwd} &= D_{i-1}^{bwd} \cup \bigcup_{o\in O} \mathit{spreimg}_o(D_{i-1}^{bwd}) \text{ for all } i \geq 1 \end{aligned}$$

Definition 4.19 *Let G be as set of states and O a set of operators, and let $D_0^{bwd}, D_1^{bwd}, \ldots$ be the backward distance sets for G and O. Then* the backward distance *of a state s to G is*

$$\delta_G^{bwd}(s) = \begin{cases} 0 \text{ if } s \in G \\ i \text{ if } s \in D_i^{bwd} \backslash D_{i-1}^{bwd} \end{cases}$$

If $s \notin D_i^{bwd}$ for all $i \geq 0$ then $\delta_G^{bwd}(s) = \infty$.

Example 4.20 We illustrate the distance computation by the diagram in Figure 4.1. The set of states with distance 0 is the set of goal states G. States with distance i are those for which there

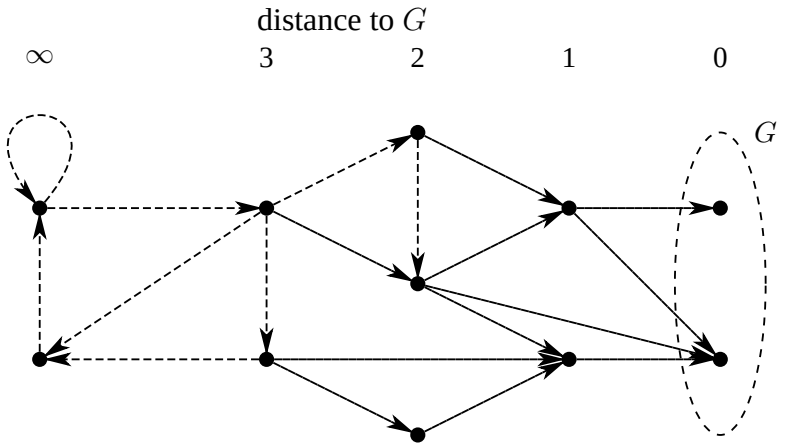


Figure 4.1: Goal distances in a nondeterministic transition system

is an action that always leads to states with distance $i-1$ or smaller. In this example the action depicted by the solid arrow has this property for every state. The dashed arrows depict the second action which for no state is guaranteed to get closer to the goal states. States for which there is no finite upper bound on the number of actions for reaching a goal state have distance ∞. ■

Given the backward distance sets we can construct a plan covering all states having a finite backward distance. Let $S' \subseteq S$ be those states having a finite backward distance. The plan π is defined by assigning for every $s \in S$ such that $\delta_G^{bwd}(s) \geq 1$ $\pi(s)$ any operator $o \in O$ such that $\mathit{img}_o(s) \subseteq D_{i-1}^{bwd}$ where $i = \delta_G^{bwd}(s)$.

The plan execution starts from one of the initial states. As we have full observability, we may observe the current state s and then execute the action corresponding to the operator $\pi(s)$, reaching one of the successor states $s' \in \mathit{img}_o(s)$. The plan execution proceeds by repeatedly observing the new current state s' and executing the associated action $\pi(s')$ until the current state is a goal state.

Lemma 4.21 *Let a state s be in D_j. Then there is a plan that reaches a goal state from s by at most j operator applications.*

The algorithm can be implemented by using logic-based data structures and operations defined in Section 4.2 by representing the set of goal states as a formula, using the logic-based operation $\mathit{spreimg}_o(\phi)$ instead of the set-based operation $\mathit{spreimg}_o(T)$ for computing the sets D_i^{bwd} that are also represented as formulae, and replacing all set-theoretic operations like $\cup$ and $\cap$ by the respective logical operations $\vee$ and $\wedge$.

4.3 Planning as satisfiability in the propositional logic and QBF

The techniques presented in Sections 3.6 and 3.6.5 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 $\mathit{img}_T(S) = \bigcup_{s \in S} \mathit{img}_T(s)$.

4.3.1 Advanced translation of nondeterministic operators into propositional logic

In Section 4.1.2 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 $\mathit{EPC}_l(e)$ 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 $\mathit{EPC}_l^{nd}(e, \sigma)$. The sequence σ of integers is used for deriving unique names for auxiliary variables in $\mathit{EPC}_l^{nd}(e, \sigma)$. The sequences correspond to paths in the tree formed by nested nondeterministic choices and conjunctions.

$$
\begin{array}{ll}
\mathit{EPC}_l^{nd}(e, \sigma) & = \mathit{EPC}_l(e) \text{ if } e \text{ is deterministic} \\
\mathit{EPC}_l^{nd}(e_1|e_2, \sigma) & = (x_\sigma \wedge \mathit{EPC}_l^{nd}(e_1, \sigma 1)) \vee (\neg x_\sigma \wedge \mathit{EPC}_l^{nd}(e_2, \sigma 1)) \\
\mathit{EPC}_l^{nd}(e_1 \wedge \cdots \wedge e_n, \sigma) & = \mathit{EPC}_l^{nd}(e_1, \sigma 1) \vee \cdots \vee \mathit{EPC}_l^{nd}(e_n, \sigma n)
\end{array}
$$

The translation of nondeterministic operators into the propositional logic is similar to the translation for deterministic operators given in Section 3.6.4. 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 $\Omega(o)$.

$$
\begin{array}{l}
(a \wedge \neg a') \rightarrow ((o_1 \wedge \mathit{EPC}_{\neg a}^{nd}(e_1, \Omega(o_1))) \vee \cdots \vee (o_n \wedge \mathit{EPC}_{\neg a}^{nd}(e_n, \Omega(o_n)))) \\
(\neg a \wedge a') \rightarrow ((o_1 \wedge \mathit{EPC}_{a}^{nd}(e_1, \Omega(o_1))) \vee \cdots \vee (o_n \wedge \mathit{EPC}_{a}^{nd}(e_n, \Omega(o_n))))
\end{array}
$$

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.

$$\begin{array}{l}(o \rightarrow c) \wedge \\ \bigwedge_{a \in A}(o \wedge \mathit{EPC}_a^{nd}(e, \Omega(o)) \rightarrow a') \wedge \\ \bigwedge_{a \in A}(o \wedge \mathit{EPC}_{\neg a}^{nd}(e, \Omega(o)) \rightarrow \neg a')\end{array}$$

Example 4.22 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 $\mathit{EPC}_a^{nd}(e, \sigma)$ and $\mathit{EPC}_{\neg a}^{nd}(e, \sigma))$ for all $a \in A$.

Definition 4.23 *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.*

$$\begin{array}{rcl}[e]_s^{\sigma,v} & = & [e]_s^{det} \textit{ if } e \textit{ is deterministic} \\ [e_1|e_2]_s^{\sigma,v} & = & \begin{cases} [e_1]_s^{\sigma 1,v} \textit{ if } v(x_\sigma) = 1 \\ [e_2]_s^{\sigma 1,v} \textit{ if } v(x_\sigma) = 0 \end{cases} \\ [e_1 \wedge \cdots \wedge e_n]_s^{\sigma,v} & = & [e_1]_s^{\sigma 1,v} \cup \cdots \cup [e_n]_s^{\sigma n,v}\end{array}$$

Lemma 4.24 *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 4.25 *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 (Section 4.3.2) 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.

4.3.2 Finding plans by evaluation of QBF

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 3.6.5.

Theorem 4.26 *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 4.24 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 4.25 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 4.24 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 4.24 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 4.25 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 4.25 $v_{plan} \cup v_{nd} \cup s_{i-1}[A^{i-1}/A] \cup s_i[A^i/A] \models \mathcal{R}_3(A^{i-1}, A^i, O^{i-1}, X^{i-1})$. 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 4.24 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 $s_n[A^n/A] \subseteq v_{exec}$. □

4.4 Literature

The state-space traversal techniques based on existential abstraction were first used in connection with verification methods for model-checking [Burch *et al.*, 1994; Clarke *et al.*, 1994] based on ordered binary decision diagrams (BDDs) [Bryant, 1992]. The model-checking problem is closely related to the deterministic planning problem and to the problem of testing whether a given conditional plan (program, controller) satisfies its specification, for example, reaches the goal states. Not surprisingly, these state-space traversal techniques have been used as an implementation technique for many algorithms for conditional planning [Hoey *et al.*, 1999; Cimatti *et al.*, 2003; Bertoli *et al.*, 2001; Rintanen, 2005]. When probabilities are involved, like when implementing MDP algorithms, a generalization of BDDs called algebraic decision diagrams (ADDs) is used [Fujita *et al.*, 1997; Bahar *et al.*, 1997].

The satisfiability planning approach of Kautz and Selman [1992; 1996] was first applied in planning with nondeterministic actions and several initial states by Rintanen [1999]. The deterministic planning problem restricted to polynomial size plans is NP-complete, which makes it possible to reduce the problem efficiently to SAT. Nondeterministic planning under the same plan size restriction does not appear to be in NP because the corresponding decisions problems are complete for Σ_2^p and Π_2^p. Hence reduction to SAT is not in general feasible but reduction to QBF is. Rintanen's QBF have the prefix $\exists\forall\exists$ corresponding to the structure of nondeterministic planning problems: *there is* a plan such that *for all* eventualities *there is* an execution leading to a goal state. This idea is applicable to very general forms of conditional planning with partial observability. A variant of Rintanen's approach has been used in some later works by replacing the outermost $\exists$ quantification by an ad hoc search algorithm that goes through candidate plans and tests whether a candidate plan is a valid plan by a satisfiability test of an unquantified propositional formula [Castellini *et al.*, 2003]. This satisfiability test is essentially the same as the one done in *bounded model-checking* [Biere *et al.*, 1999] for testing whether a given transition system ($\sim$ transition system controlled by a plan) can reach a state satisfying a given property.

The heuristics in Section 3.4 can be generalized to nondeterministic operators and used to solve more general planning problems by heuristic search algorithms. The choice of algorithm depends on the definition of plans which is determined by assumptions concerning observability.

Without observability plans are sequences of operators and can be found by standard heuristic search algorithms like A$*$ and IDA$*$ by search in the belief space.

If observations are possible, plan search can be viewed as search in an and-or tree. Heuristic search algorithms for and-or trees are for example AO$*$ [Martelli and Montanari, 1973; 1978; Nilsson, 1980] and LAO$*$ [Hansen and Zilberstein, 2001]. Other applicable algorithms include RTDP [Barto *et al.*, 1995] and LRTDP [Bonet and Geffner, 2003].

The most straightforward heuristics for these planning problems ignore nondeterminism and observability by replacing each nondeterministic operator by a number of deterministic operators and assuming full observability. This can be justified by efficiency grounds. Other heuristics attempt to improve the informativeness by taking observability and nondeterminism better into account. For more on the topic see for example [Bryce and Kambhampati, 2004; Rintanen, 2004b].

Bibliography

[Allen *et al.*, 1990] J. Allen, J. A. Hendler, and A. Tate, editors. *Readings in Planning*. Morgan Kaufmann Publishers, 1990.

[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.

[Bahar *et al.*, 1997] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design: An International Journal*, 10(2/3):171–206, 1997.

[Barto *et al.*, 1995] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[Bertoli *et al.*, 2001] 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.

[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.

[Bonet and Geffner, 2003] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, pages 12–21, 2003.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[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.

[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.

[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.

[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.

[Dijkstra, 1976] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, New Jersey, 1976.

[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.*, 1969] G. Ernst, A. Newell, and H. Simon. *GPS: A Case Study in Generality and Problem Solving*. Academic Press, 1969.

[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.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(2-3):189–208, 1971.

[Fujita *et al.*, 1997] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. *Formal Methods in System Design: An International Journal*, 10(2/3):149–169, 1997.

[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.

[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.

[Green, 1969] C. Green. Application of theorem-proving to problem solving. In D. E. Walker and L. M. Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 219–239. William Kaufmann, 1969.

[Hansen and Zilberstein, 2001] E. A. Hansen and S. Zilberstein. LAO$*$: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 29(1-2):35–62, 2001.

[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.

[Hoey *et al.*, 1999] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In K. B. Laskey and H. Prade, editors, *Uncertainty in Artificial Intelligence, Proceedings of the Fifteenth Conference (UAI-99)*, pages 279–288. Morgan Kaufmann Publishers, 1999.

[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.

[Kaelbling *et al.*, 1998] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[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.

[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.

[Korf, 1985] R. E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[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.

[Martelli and Montanari, 1973] A. Martelli and U. Montanari. Additive AND/OR graphs. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 1–11, 1973.

[Martelli and Montanari, 1978] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 12(12):1025–1039, 1978.

[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.

[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.

[Nilsson, 1980] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[Pearl, 1984] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[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. 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*. Morgan Kaufmann Publishers, 2005. to appear.

[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, 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[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.

[Selman *et al.*, 1996] B. Selman, D. G. Mitchell, and H. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):459–465, 1996.

[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.

[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.

[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.

Index