Chapter 4

Conditional planning

Now we relax the two assumptions that characterize deterministic planning, the determinism of the operators and the restriction to one initial state. Instead of an initial state, we will have a formula describing a set of initial states, and our definition of operators will be extended to cover nondeterministic actions.

These extensions to the planning problem mean that the notion of plans as sequences of operators is not sufficient, because the states that will be visited are not uniquely determined by the actions taken so far: different initial states may require different actions, and nondeterministic actions lead to several alternative successor states.

Plans will be mappings from the observations made so far to the next actions to be taken. There are several possibilities in representing such mapping. Our definition of plans has the form of programs consisting of operators, sequences of operators, and conditional that choose subplans based on observations.

What observations can be made has a strong effect on how planning can be done. There are two special cases we will discuss separately from the general conditional planning problem, those with no observations possible and with everything observable.

When there are no observations, the definition of plans reduces to sequences of actions like in deterministic planning, but executing the plans does not always generate the same sequence of states because of nondeterminism and multiple initial states.

For the fully observable case planning algorithms are much simpler than when observability is only partial. In this case plans can alternatively be defined as mappings from states to actions, and there is no need for the plans to have memory in the way program-like plans have, a form of program counter that keeps track which location of the plan is currently executed.

In this chapter we first discuss nondeterministic actions and transition systems, then define what conditional plans are, and then discuss algorithms for the three types of conditional planning, starting from the simplest case of planning with full observability, followed by planning without observability, and finally the general partially observable planning problem. The chapter is concluded by a discussion of the computational complexity of conditional planning.

4.1 Nondeterministic operators

There is often uncertainty about what the exacts effects of an action are. This is because not all aspects of the world can be exactly formalized, and part of the things that are not formalized may

affect the outcomes of the actions.

Consider for example a robot that plays basket ball. However well the robot is designed, there is still always small uncertainty about the exact physical properties of the ball and the hands the robot uses for throwing the ball. Therefore it is possible to predict the outcome of throwing the ball only up to a certain precision, and a ball thrown by the robot may still miss the basket. This would be a typical situation in which we would formalize an action as nondeterministic. It succeeds with a certain probability, and fails otherwise, and the exact conditions that lead to success or failure are outside the formalization of the action.

In other cases nondeterminism arises because formalizing all the things affecting the outcomes of an action does not bring further benefit. Consider a robot that makes and serves coffee for the members of the research lab. It might be well known that certain lab members never drink coffee, that certain lab members always drink coffee right after lunch, and so on. But it would often not be very relevant for the robot to know these things, as its task is simply to make and serve a cup of coffee whenever somebody requests it to do so. So for the coffee making robot we could just formalize the event that somebody requests coffee as a nondeterministic event, even though there are well known deeper regularities that govern these requests.

In this section we extend the definition of operators first given in Section 2.3 to cover nondeterminism and discuss two normal forms for nondeterministic operators, We then present a translation of nondeterministic operators into the propositional logic, and in the next sections we discuss several planning algorithms that can be efficiently implemented with binary decision diagrams that represent transition relations corresponding to nondeterministic actions.

Probabilities can often be associated with the alternative nondeterministic effects an operator may have, and we include the probabilities in our definition of nondeterministic operators. However, the algorithms discussed in this chapter ignore these probabilities, and they will be only needed later for the probabilistic variants of conditional planning in Chapter 5.

Definition 4.1 *Let A be a set of state variables. A* nondeterministic operator *is a pair $\langle c, e \rangle$ where c is a propositional formula over A describing the precondition, and e is a nondeterministic effect. Effects are recursively defined as follows.*

  1. *a and $\neg a$ for state variables $a \in A$ are effects.*

  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 conjunction $\top$.)*

  3. *$c \rhd e$ is an effect over A if c is a formula over A and e is an effect over A.*

  4. *$p_1 e_1 | \cdots | p_n e_n$ is an effect over A if $e_1, \ldots, e_n$ for $n \geq 2$ are effects over A, $p_i > 0$ for all $i \in \{1, \ldots, n\}$ and $\sum_{i=1}^{n} p_i = 1$.*

The definition extends Definition 2.7 by allowing nondeterministic choice as $p_1 e_1 | \cdots | p_n e_n$.

Next we give a formal semantics for the application of a nondeterministic operator. The definition of deterministic operator application (Definition 2.8) assigned a state to every state and operator. The new definition assigns a probability distribution over the set of successor states for a given state and operator.

Definition 4.2 (Nondeterministic operator application) *Let $\langle c, e \rangle$ be an operator over A. Let s be a state, that is an assignment of truth values to A. The operator is applicable in s if $s \models c$.*

Recursively assign each effect e a set $[e]_s$ of pairs $\langle p, l \rangle$ where p is a probability $0 < p \leq 1$ and l is a set of literals a and $\neg a$ for $a \in A$.

1. *$[a]_s = \{\langle 1, \{a\} \rangle\}$ and $[\neg a]_s = \{\langle 1, \{\neg a\} \rangle\}$ for $a \in A$.*

2. *$[e_1 \wedge \cdots \wedge e_n]_s = \{\langle \Pi_{i=1}^n p_i, \bigcup_{i=1}^n f_i \rangle | \langle p_1, f_1 \rangle \in [e_1]_s, \ldots, \langle p_n, f_n \rangle \in [e_n]_s\}$.*

3. *$[c' \triangleright e']_s = [e']_s$ if $s \models c'$ and $[c' \triangleright e']_s = \{\langle 1, \emptyset \rangle\}$ otherwise.*

4. *$[p_1 e_1 | \cdots | p_n e_n]_s = \{\langle p_1 \cdot p, e \rangle | \langle p, e \rangle \in [e_1]_s\} \cup \cdots \cup \{\langle p_n \cdot p, e \rangle | \langle p, e \rangle \in [e_n]_s\}$*

Above in (4) the union of sets is defined so that for example $\{\langle 0.2, \{a\} \rangle\} \cup \{\langle 0.2, \{a\} \rangle\} = \{\langle 0.4, \{a\} \rangle\}$, that is, same sets of changes are combined by summing their probabilities.

The successor states of s under the operator are ones that are obtained from s by making the literals in f for $\langle p, f \rangle \in [e]_s$ true and retaining the truth-values of state variables not occurring in f. The probability of a successor state is the sum of the probabilities p for $\langle p, f \rangle \in [e]_s$ that lead to it.

Each $\langle p, l \rangle$ means that with probability p the literals that become true are those in l, and hence indicate the probabilities of the possible successor states of s. For any $[e]_s = \{\langle p_1, l_1 \rangle, \ldots, \langle p_n, l_n \rangle\}$ the sum of probabilities is $\sum_{i=1}^n p_i = 1$.

In non-probabilistic variants of planning we also use a semantics that ignores the probabilities. The following definition gives those successor states that have a non-zero probability according to the preceding definition.

Definition 4.3 (Nondeterministic operator application II) *Let $\langle c, e \rangle$ be an operator over A. Let s be a state, that is an assignment of truth values to A. The operator is applicable in s if $s \models c$. Recursively assign each effect e a set $[e]_s$ of literals a and $\neg a$ for $a \in A$.*

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

2. *$[e_1 \wedge \cdots \wedge e_n]_s = \{\bigcup_{i=1}^n f_i | f_1 \in [e_1]_s, \ldots, f_n \in [e_n]_s\}$.*

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

4. *$[p_1 e_1 | \cdots | p_n e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$*

The successor states under $\langle c, e \rangle$ are obtained from s by assigning the sets of literals in $[e]_s$ true.

4.1.1 Normal forms for nondeterministic operators

We can generalize the normal form defined in Section 2.3.2 to nondeterministic effects and operators. In the normal formal form the nondeterministic choices together with conjunctions are outside, and all atomic effects are as consequents of conditionals.

For showing that every nondeterministic effect can be transformed into normal form we have extended our set of equivalences on effects to cover nondeterministic choice. The whole set of equivalences is given in Table 4.1.

$$c \triangleright (e_1 \wedge \cdots \wedge e_n) \quad \equiv \quad (c \triangleright e_1) \wedge \cdots \wedge (c \triangleright e_n) \tag{4.1}$$

$$c \triangleright (c' \triangleright e) \quad \equiv \quad (c \wedge c') \triangleright e \tag{4.2}$$

$$c \triangleright (p_1 e_1 | \cdots | p_n e_n) \quad \equiv \quad p_1(c \triangleright e_1) | \cdots | p_n(c \triangleright e_n) \tag{4.3}$$

$$(c_1 \triangleright e) \wedge (c_2 \triangleright e) \quad \equiv \quad (c_1 \vee c_2) \triangleright e \tag{4.4}$$

$$e \wedge (c \triangleright e) \quad \equiv \quad e \tag{4.5}$$

$$e \quad \equiv \quad \top \triangleright e \tag{4.6}$$

$$e \wedge (p_1 e_1 | \cdots | p_n e_n) \quad \equiv \quad p_1(e \wedge e_1) | \cdots | p_n(e \wedge e_n) \tag{4.7}$$

$$p_1(p_1' e_1' | \cdots | p_n' e_n') | p_2 e_2 | \cdots | p_n e_n \quad \equiv \quad (p_1 p_1') e_1' | \cdots | (p_1 p_n') e_n' | p_2 e_2 | \cdots | p_n e_n \tag{4.8}$$

$$p_1(e' \wedge (c \triangleright e_1)) | p_2 e_2 | \cdots | p_n e_n \quad \equiv \quad (c \triangleright (p_1(e' \wedge e_1) | p_2 e_2 | \cdots | p_n e_n)) \tag{4.9}$$

$$\wedge (\neg c \triangleright (p_1 e' | p_2 e_2 | \cdots | p_n e_n)) \tag{4.10}$$

Table 4.1: Equivalences on effects

Definition 4.4 (Normal form for nondeterministic operators) *An effect is in* normal form *if it can be derived as follows.*

A deterministic effect is in normal form if it is a conjunction (0 or more conjuncts) of effects $c \triangleright p$ and $c \triangleright \neg p$, with at most one occurrence of p and $\neg p$ for any state variable $p \in A$.

A nondeterministic effect is in normal form if it is $p_1 e_1 | \cdots | p_n e_n$ for deterministic effects e_i that are in normal form, or it is a conjunction of nondeterministic effects in normal form.

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

Theorem 4.5 *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: By using equivalences 4.1, 4.2 and 4.3 in Table 4.1 we can transform any effect so that all atomic effects l occur as consequents of conditional $c \triangleright l$. By further using equivalence 4.7 we can transform the effect to normal form. $\qquad\square$

Example 4.6 The effect

$$a \triangleright (0.3b | 0.7(c \wedge f)) \wedge (0.2(d \wedge e) | 0.8(b \triangleright e))$$

in normal form is

$$(0.3(a \triangleright b) | 0.7((a \triangleright c) \wedge (a \triangleright f))) \wedge (0.2((\top \triangleright d) \wedge (\top \triangleright e)) | 0.8(b \triangleright e)).$$

$\blacksquare$

In certain cases, for example for defining regression for nondeterministic operators, it is best to restrict to operators in a slightly more restrictive normal form, in which nondeterminism may appear only at the topmost structure in the effect.

Definition 4.7 (Normal form II for nondeterministic operators) *An effect is in* normal form II *if it can be derived as follows.*

A deterministic effect is in normal formal II if it is a conjunction (0 or more conjuncts) of effects $c \rhd p$ and $c \rhd \neg p$, with at most one occurrence of p and $\neg p$ for any state variable $p \in A$.

A nondeterministic effect is in normal form II if it is of form $p_1 e_1 | \cdots | p_n e_n$ where e_i are deterministic effects in normal form II.

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

4.1.2 Translation of nondeterministic operators into propositional logic

In Section 3.5.2 we gave a translation of deterministic operators into the propositional logic. In this section we extend this translation to nondeterministic operators.

For expressing the translation we define for a given effect e a set *changes*(e) of state variables as follows. This is the set of state variables possibly changed by the effect, or in other words, the set of state variables occurring in the effect not in the antecedent c of a conditional $c \rhd e$.

$$
\begin{aligned}
changes(a) &= \{a\} \\
changes(\neg a) &= \{a\} \\
changes(c \rhd e) &= changes(e) \\
changes(e_1 \wedge \cdots \wedge e_n) &= changes(e_1) \cup \cdots \cup changes(e_n) \\
changes(p_1 e_1 | \cdots | p_n e_n) &= changes(e_1) \cup \cdots \cup changes(e_n)
\end{aligned}
$$

We make the following assumption to slightly simplify the translation.

Assumption 4.8 *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 $(0.5a | 0.5b) \wedge (0.5\neg a | 0.5c)$ that may make a simultaneously true and false. It also rules out effects like $(0.5(d \rhd a) | 0.5b) \wedge (0.5(\neg d \rhd \neg a) | c)$ that are well-defined and could be translated into the propositional logic. However, the additional complexity to the translation outweights the benefit of allowing them.

We define the translation of effects satisfying Assumption 4.8 into propositional logic recursively. The problem in the translation that does not show up with deterministic operators is that for nondeterministic choices $p_1 e_1 | \cdots | p_n e_n$ the formula for each alternative e_i has to express for exactly the same set of state variables what changes take or do not take place. This becomes a bit tricky when we have a lot of nesting of nondeterministic choice and conjunctions.

Now we give the translation of an effect e (in normal form) restricted to state variables B. This means that only state variables in B may occur in e in atomic effects (but do not have to), and the formula does not say anything about the change of state variables not in B (but may of course refer to them in antecedents of conditionals.)

$$
\begin{aligned}
\mathrm{PL}_B(e) &= \bigwedge_{a \in B} \left(((a \wedge \neg EPC_{\neg a}(e)) \vee EPC_a(e)) \leftrightarrow a' \right) \\
&\quad \text{when } e \text{ is deterministic} \\
\mathrm{PL}_B(p_1 e_1 | \cdots | p_n e_n) &= \mathrm{PL}_B(e_1) \vee \cdots \vee \mathrm{PL}_B(e_n) \\
\mathrm{PL}_B(e_1 \wedge \cdots \wedge e_n) &= \mathrm{PL}_{B \setminus (B_2 \cup \cdots \cup B_n)}(e_1) \wedge \mathrm{PL}_{B_2}(e_2) \wedge \cdots \wedge \mathrm{PL}_{B_n}(e_n) \\
&\quad \text{where } B_i = changes(e_i) \text{ for all } i \in \{1, \ldots, n\}
\end{aligned}
$$

The first part of the translation $\mathrm{PL}_B(e)$ for deterministic e is the translation of deterministic effects we presented in Section 3.5.2, but restricted to state variables in B. The other two cover all

nondeterministic effects in normal form. The idea of the translation of a conjunction $e_1 \wedge \cdots \wedge e_n$ of nondeterministic effects is that only the translation of the first effect e_1 indicates when state variables occurring in B do not change.

Additionally, we require that operators are not applied in states in which some state variable would be set simultaneously both true and false.

$$
\begin{aligned}
\mathrm{XPL}_B(e) &= \bigwedge_{a \in B} \left(\neg(EPC_{\neg a}(e) \wedge EPC_a(e)) \right) \\
&\qquad \text{when } e \text{ is deterministic} \\
\mathrm{XPL}_B(p_1 e_1 | \cdots | p_n e_n) &= \mathrm{XPL}_B(e_1) \wedge \cdots \wedge \mathrm{XPL}_B(e_n) \\
\mathrm{XPL}_B(e_1 \wedge \cdots \wedge e_n) &= \mathrm{XPL}_{B \setminus (B_2 \cup \cdots \cup B_n)}(e_1) \wedge \mathrm{XPL}_{B_2}(e_2) \wedge \cdots \wedge \mathrm{XPL}_{B_n}(e_n) \\
&\qquad \text{where } B_i = changes(e_i) \text{ for all } i \in \{1, \ldots, n\}
\end{aligned}
$$

The translation of an effect e in normal form into the propositional logic is $\mathrm{PL}_A(e) \wedge \mathrm{XPL}_A(e)$ where A is the set of all state variables.

Example 4.9 We translate the effect

$$
e = (0.5A | 0.5(C \rhd A)) \wedge (0.5B | 0.5C)
$$

into a propositional formula. The set of state variables is $A = \{A, B, C, D\}$.

$$
\begin{aligned}
\mathrm{PL}_{\{A,B,C,D\}}(e) &= \mathrm{PL}_{\{A,D\}}(0.5A | 0.5(C \rhd A)) \wedge \mathrm{PL}_{\{B,C\}}(0.5B | 0.5C) \\
&= (\mathrm{PL}_{\{A,D\}}(A) \vee \mathrm{PL}_{\{A,D\}}(C \rhd A)) \wedge \\
&\quad (\mathrm{PL}_{\{B,C\}}(B) \vee \mathrm{PL}_{\{B,C\}}(C)) \\
&= ((A' \wedge (D \leftrightarrow D')) \vee (((A \vee C) \leftrightarrow A') \wedge (D \leftrightarrow D'))) \wedge \\
&\quad ((B' \wedge (C \leftrightarrow C')) \vee ((B \leftrightarrow B') \wedge C'))
\end{aligned}
$$

$\blacksquare$

4.1.3 Operations on nondeterministic transitions represented as formulae

In Section 3.7.1 we discussed the image and preimage computations of transition relations expressed as propositional formulae. In this section we consider also nondeterministic transition relations and want to compute the set of states from which reaching a state in a given set of states is certain, not just possible. The (weak) preimage operation in Section 3.7 does not do this. For example, the weak preimage of a with respect to the relation $\{\langle b, a \rangle, \langle b, c \rangle\}$ is $\{b\}$, although also c is a possible successor state of b.

The strong preimage of a set of states consists of those states from which only states inside the given set are reached. This is formally defined as follows.

$$
spreimg_R(S) = \{s | s' \in S, \langle s, s' \rangle \in R, img_R(s) \subseteq S\}
$$

Lemma 4.10 *Images, strong preimages and weak preimages of sets of states are related to each other as follows.*

1. *$spreimg_o(S) \subseteq wpreimg_o(S)$*

2. *$img_o(spreimg_o(S)) \subseteq S$*

3. *$wpreimg_o(S) = spreimg_o(S)$ when o is deterministic.*

Proof: □

Strong preimages can be computed by formula manipulation when sets of states and transition relations are represented as propositional formulae.

$$(\forall A'.(\mathcal{R}_o(A, A') \rightarrow (\phi[a'_1/a_1, \ldots, a'_n/a_n]))) \wedge (\exists A'.\mathcal{R}_o(A, A'))$$

Here $\forall a.\phi$ is *universal abstraction* which is defined analogously to existential abstraction as

$$\forall a.\phi = \phi[\top/a] \wedge \phi[\bot/a].$$

4.1.4 Regression for nondeterministic operators

Regression for deterministic operators was given as Definition 3.6. It is straightforward to generalize this definition for nondeterministic operators in the second (more restricted) normal form.

Definition 4.11 (Regression) *Let ϕ be a propositional formula describing a set of states. Let $\langle z, e \rangle$ be an operator in normal form II with $e = p_1 e_1 | \cdots | p_n e_n$.*
The regression *of ϕ with respect to $o = \langle z, e \rangle$ is defined as the formula $regr_o(\phi) = regr_{\langle z, e_1 \rangle}(\phi) \wedge \cdots \wedge regr_{\langle z, e_n \rangle}(\phi)$ where $regr_{\langle z, e \rangle}(\phi)$ refers to regression of deterministic operators as given in Definition 3.6.*

It is presumably possible to define regression for nodeterministic operators in the first normal with no restriction on nesting of nondeterminism and conjunctions, but the definition is more complicated, and we do not discuss the topic further here.

Theorem 4.12 *Let $S' = \{s'|s' \models \phi\}$. Then $spreimg_o(S) = \{s|s \models regr_o(\phi)\}$.*

Proof: This is because a state in ϕ has to be reached no matter which effect e_i is chosen, so we take the intersection/conjunction of the states obtained by regression with $\langle z, e_1 \rangle, \ldots, \langle z, e_n \rangle$. □

Example 4.13 Let $o = \langle A, (0.5B|0.5\neg C) \rangle$. Then

$$\begin{aligned} regr_o(B \leftrightarrow C) &= regr_{\langle A, B \rangle}(B \leftrightarrow C) \wedge regr_{\langle A, \neg C \rangle}(B \leftrightarrow C) \\ &= (A \wedge (\top \leftrightarrow C)) \wedge (A \wedge (B \leftrightarrow \bot)) \\ &\equiv (A \wedge C) \wedge (A \wedge \neg B) \\ &\equiv A \wedge C \wedge \neg B \end{aligned}$$

∎

4.2 Problem definition

We state the conditional planning problem in the general form. Because the number of observations that are possible has a very strong effect on the type of solution techniques that are applicable, we will discuss algorithms for three classes of planning problems that are defined in terms of restrictions on the set B of observable state variables.

Definition 4.14 *A 5-tuple* $\langle A, I, O, G, B \rangle$ *consisting of a set A of state variables, a propositional formula I over A, a set O of operators over A, a propositional formula G over A, and a set $B \subseteq A$ of state variables is* a problem instance in nondeterministic planning.

The set B did not appear in the definition of deterministic planning. This is the set of *observable state variables*. The idea is that plans can make decisions about what operations to apply and how the execution proceeds based on the values of the observable state variables. Restrictions on observability and sensing emerge because of various restrictions on the sensors human beings and robots have: typically only a small part of the world can be observed.

The task in nondeterministic planning is the same as in deterministic planning (Section 3.1): to find a plan that starting from any state in I is guaranteed to reach a state in G.

However, because of nondeterminism and the possibility of more than one initial state, it is in general not possible to use the same sequence of operators for reaching the goals from all the initial states, and a more general notion of plans has to be used.

Nondeterministic planning problems under certain restrictions have very different properties than the problem in its full generality. In Chapter 3 we had the restriction to one initial state (I was defined as a valuation) and deterministic operators. We relax these two restrictions in this chapter, but still consider two special cases obtained by restrictions on the set B of observable state variables.

1. Full observability.

   This is the most direct extension of the deterministic planning problem of the previous chapter. The difference is that we have to use a more general notion of plans with branches (and with loops, if there is no upper bound on the number of actions that might be needed to reach the goals.)

2. No observability.

   Planning without observability can be considered more difficult than planning with full observability, although they are in many respects not directly comparable.

   The main difference to deterministic planning as discussed in Chapter 3 and to planning with full observability is that during plan execution it is not known what the actual current state is, and there are several possible current states. This complication means that planning takes place in *the belief space*: the role of individual states in deterministic planning is taken by sets of states, called *belief states*.

   Because no observations can be made, branching is not possible, and plans are still just sequences of actions, just like in deterministic planning with one initial state.

The type of observability we consider in this lecture is very restricted as only values of individual state variables can be observed (as opposed to arbitrary formulae) and observations are independent of what operators have been executed before. Hence we cannot for example directly express special sensing actions. However, extensions to the above definition like sensing actions can be relatively easily reduced to the basic definition but we will not discuss this topic further.

4.2.1 Conditional plans

Plans are directed graphs with nodes of degree 1 labeled with operators and edges from nodes of degree ≥ 2 labeled with formulae.

Definition 4.15 *Let* $\langle A, I, O, G, B \rangle$ *be a problem instance in nondeterministic planning. A con-ditional plan is a triple* $\langle N, b, l \rangle$ *where*

- *N is a finite set of nodes,*

- *$b \in N$ is the initial node,*

- *$l : N \to (O \times N) \cup 2^{\mathcal{L} \times N}$ is a function that assigns each node an operator and a successor node $\langle o, n \rangle \in O \times N$ or a set of conditions and successor nodes $\langle \phi, n \rangle$.*

  Here ϕ are formulae over B.

Plan execution begins from the initial node b, and the sequence of operators and states generated when executing a plan is determined as follows.

Let $n \in N$ be a node in the plan. If $l(n) = \langle o, n' \rangle$ then n is an operator node. If $l(n) = \emptyset$ then n is a terminal node. Otherwise n is a branch node and $l(n) = \{\langle \phi_1, n_1 \rangle, \ldots, \langle \phi_m, n_m \rangle\}$ for some m.

Execution in an operator node with label $\langle o, n \rangle$ proceeds by applying operator o and making n the current plan node.

Execution in a branch node n with label $l(n) = \{\langle \phi_1, n_1 \rangle, \ldots, \langle \phi_m, n_m \rangle\}$ proceeds by evaluating the formulae ϕ_i with respect to the valuation s of the observable state variables, and if $s \models \phi_i$, then making n_i the current plan node.[1]

Plan execution ends in a terminal node $n \in N, l(n) = \emptyset$.

The plans can of course be written in the same form as programs in conventional programming languages by using *case* statements for branching and *goto* statements for jumping to the successor nodes of a plan node.

Example 4.16 Consider the plan $\langle N, b, l \rangle$ for a problem instance with the operators $O = \{o_1, o_2, o_3\}$, where

$$
\begin{aligned}
N &= \{1, 2, 3, 4, 5\} \\
b &= 1 \\
l(1) &= \langle o_3, 2 \rangle \\
l(2) &= \{\langle \phi_1, 1 \rangle, \langle \phi_2, 3 \rangle, \langle \phi_3, 4 \rangle\} \\
l(3) &= \langle o_2, 4 \rangle \\
l(4) &= \{\langle \phi_4, 1 \rangle, \langle \phi_5, 5 \rangle\} \\
l(5) &= \emptyset
\end{aligned}
$$

This could be visualized as the program.

```
1:  o3
2:  CASE
        φ1: GOTO 1
        φ2: GOTO 3
        φ3: GOTO 4
3:  o2
4:  CASE
        φ4: GOTO 1
        φ5: GOTO 5
5:
```

---

[1]The result of plan execution is undefined if there are several formulae true in the state s.

Every plan $\langle N, b, l \rangle$ can be written as such a program. Nodes n with $l(n) = \emptyset$ corresponds to *goto*s to the program's last label after which nothing follows. ∎

A plan is *acyclic* if it is a directed acyclic graph in the usual graph theoretic sense.

4.2.2 Execution graph

We define the satisfaction of plan objectives in terms of the transition system that is obtained when the original transition system is being controlled by a plan, that is, the plan chooses which of the transitions possible in a state is taken. For goal reachability, without unbounded looping it would be required that any maximal path from an initial state has finite length and ends in a goal state. With unbounded looping it would be required that from any state to which there is a path from an initial state that does not visit a goal state there is a path of length ≥ 0 to a goal state.

Definition 4.17 (Execution graph of a plan) *Let* $\langle A, I, O, G, B \rangle$ *be a problem instance and* $\pi = \langle N, b, l \rangle$ *be a plan. Then we define* the execution graph *of* π *as a pair* $\langle M, E \rangle$ *where*

1. $M = S \times N$, *where* S *is the set of Boolean valuations of* A,

2. $E \subseteq M \times M$ *has an edge from* $\langle s, n \rangle$ *to* $\langle s', n' \rangle$ *if and only if*

   (a) $n \in N$ *is an operator node with* $l(n) = \langle o, n' \rangle$ *and* $s' \in img_o(s)$, *or*
   (b) $n \in N$ *is a branch node with* $\langle \phi, n' \rangle \in l(n)$ *and* $s' = s$ *and* $s \models \phi$.

Definition 4.18 (Reachability goals RG) *A plan* $\pi = \langle N, b, l \rangle$ *solves a problem instance* $\langle A, I, O, G, B \rangle$ *under the* Reachability *(RG) criterion if its execution graph fulfills the following.*
For all states s *such that* $s \models I$, *for every* (s', n) *to which there is a path from* (s, b) *that does not visit* (s''', n'') *for any* s''' *such that* $s''' \models G$ *and terminal node* n'' *there is also a path from* (s', n) *to some* (s'', n') *such that* $s'' \models G$ *and* n' *is a terminal node.*

This plan objective with unbounded looping can be interpreted probabilistically. For every nondeterministic choice in an operator we have to assume that each of the alternatives has a non-zero probability. Then for goal reachability, a plan with unbounded looping is simply a plan that has no finite upper bound on the length of its executions, but that with probability 1 eventually reaches a goal state. A non-looping plan also reaches a goal state with probability 1, but there is a finite upper bound on the execution length.

Definition 4.19 (Maintenance goals MG) *A plan* $\pi = \langle N, b, l \rangle$ *solves a problem instance* $\langle A, I, O, G, B \rangle$ *under the* Maintenance *(MG) criterion if its execution graph fulfills the following.*
For all states s *and* s' *and plan nodes* $n \in N$ *such that* $s \models I$, *if there is a path of length* ≥ 0 *from* (s, b) *to some* (s', n), *then* $s' \models G$ *and* (s', n) *has a successor.*

We can also define a plan objective that combines the reachability and maintenance criteria: visit infinitely often one of the goal states. This is a proper generalization of both of the criteria because we can rather easily reduce both special cases to the general case. Algorithms for the general case generalize algorithms for both special cases.

4.3 Planning with full observability

When during plan execution the current state is always exactly known, plans can be found by the same kind of state space traversal algorithms already used for deterministic planning in Section 3.7.

The differences to algorithms for deterministic planning stem from nondeterminism. The main difference is that successor states are not uniquely determined by the current state and the action, and different action may be needed for each successor state. Further, nondeterminism may require loops. Consider tossing a die until it yields 6. Plan for this task involves tossing the die over and over, and there is no upper bound on the number of tosses that might be needed.[2] Hence we need plans with loops for representing the sequences of actions of unbounded length required for solving the problem.

Below in Section 4.3.1 we first discuss the simplest algorithm for planning with nondeterminism and full observability. The plans this algorithm produces are acyclic, and the algorithm does not find plans for problem instances that only have plans with loops. Then in Section 4.3.2 we present an algorithm that also produces plans with loops. The structure of the algorithm is more complicated. Efficient implementation of these algorithms requires the use of binary decision diagrams or similar representations of sets and transition relations, as discussed in Section 3.7. Like in the BDD-based algorithms for deterministic planning, these algorithm assign a distance to all the states, with a different meaning of distance in different algorithms, and based on the distances either synthesize a program-like plan, or a plan execution mechanism uses the distances directly for selecting the operators to execute. The algorithms in this section are best implemented by representing the formulae as BDDs.

Deterministic planning has both a forward and a backward algorithm that are similar to each other, as described in Section 3.7. However, for nondeterministic problems forward search does not seem to be a good way of doing planning. For backward distances, distance i of state s means that there is a plan for reaching the goals with at most i operators. But there does not seem to be a useful interpretation of the distances computed forwards from the initial states as images of nondeterministic operators. That a goal state or all goal states can be reached by applying some i nondeterministic operators does not say anything about the possible plans, because executing those i operators might also lead to states that are not goal states and from which goal states could be much more difficult to reach.

4.3.1 An algorithm for constructing acyclic plans

The algorithm for constructing acyclic plans is an extension of the algorithm for deterministic planning given in Section 3.7.3. In the first phase the algorithm computes distances of the states. In the second phase the algorithm constructs a plan based on the distances. The distance computation is almost identical to the algorithm for deterministic planning. The only difference is the use of strong preimages instead of the preimages.[3] The second phase is more complicated, and uses the distances for constructing a plan according to Definition 4.15.

The algorithm is given in Figure 4.1. We call the distances computed by the algorithm *strong*

---

[2]However, for every $p > 0$ there is a finite plan that reaches the goal with probability p or higher.

[3]The algorithm for deterministic planning could use the slightly more complicated strong preimage computation just as well, because strong and weak preimages coincide for deterministic operators. However, this would not have any advantage for deterministic planning.

procedure FOplan(I,O,G)
$D_0 := G$;
$i := 0$;
while $I \not\subseteq D_i$ **and** $(i = 0$ **or** $D_{i-1} \neq D_i)$ **do**
    $i := i + 1$;
    $D_i := D_{i-1} \cup \bigcup_{o \in O} spreimg_o(D_{i-1})$;
end
$N := \emptyset$;
$l(j) := \emptyset$ for all j;
cnt := 1;
FOplanconstruct(0,I);

Figure 4.1: Algorithm for nondeterministic planning with full observability
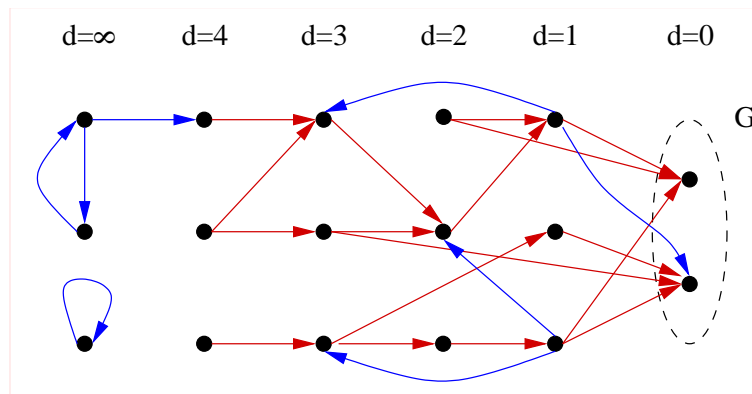

Figure 4.2: Goal distances in a nondeterministic transition system

distances because they are tight upper bounds on the number of operators needed for reaching the goals: if the distance of a state is i, then no more than i operators are needed, but it may be possible that a goal state is also reached with less than i operators if the relevant operators are nondeterministic and the right nondeterministic effects take place.

Example 4.20 We illustrate the distance computation by the diagram in Figure 4.2. The set of states with distance 0 is the set of goal states G. States with distance i are those for which at least one action always leads to states with distance $i - 1$ or smaller. In this example the action depicted by the red arrow has this property for every state. States for which there is no finite upper bound on the number of actions for reaching a goal state have distance ∞. ∎

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 distances alone could be directly used by a plan execution mechanism. The plan execution proceeds by observing the current state, looking up its distance j such that $s \in D_j \backslash D_{j-1}$, selecting an operator $o \in O$ so that $img_o(\{s\}) \subseteq D_{j-1}$, and executing the operator.

Similarly, a mapping from states to operators could be directly constructed. This kind of plan is called *memoryless* because the plan execution mechanism does not have to keep track of plan

procedure FOplanconstruct(n,S)
if $S \subseteq G$ **then return**;   *(* Goal reached for all states. *)*
for each $o \in O$
   $S' :=$ the maximal subset of S such that progress(o, S');
  if $S' \neq \emptyset$ **then**   *(* Is operator o useful for some of the states? *)*
    begin
      $S := S \backslash S'$;
      cnt := cnt+2;
      $N := N \cup \{$cnt-2,cnt-1$\}$;   *(* Create two new plan nodes. *)*
      $l(n) := l(n) \cup \{\langle S',\text{cnt}-2 \rangle\}$;   *(* First is reached from node n. *)*
      $l(\text{cnt}-2) := \langle o,\text{cnt}-1 \rangle$;   *(* Second is an operator node. *)*
      FOplanconstruct(cnt-1,$img_o(S')$);   *(* Continue from successors of S'. *)*
    end
end
if $S \neq \emptyset$ **then**   *(* If something remains in S they must be goal states. *)*
  begin
    cnt := cnt+1;
    $l(n) := l(n) \cup \{\langle S,\text{cnt}-1 \rangle\}$;   *(* Create a terminal node for them. *)*
  end

Figure 4.3: Algorithm for extracting an acyclic plan from goal distances

procedure progress(o, S)
for $j := 1$ **to** i **do**   *(* Does o take all states closer to goals? *)*
  if $img_o(S \cap D_j) \not\subseteq D_{j-1}$ **then return** false;
end
return true;

Figure 4.4: Test whether successor states are closer to the goal states

nodes. It just chooses the next operator on the basis of the current state. This corresponds to a plan that consists of a loop in which each operator is selected for some subset of possible current states, a terminal node is selected for the goal states, and then the loop repeats again.

Memoryless plans are sufficienty powerful only for the simplest form of conditional planning in which the current state can be observed uniquely (full observability). Later we will see that when there are restrictions on which observations can be made it is necessary to have memory in the plan.

Figure 4.3 gives an algorithm for generating a plan according to Definition 4.15. The algorithm works forward starting from the set of initial states. Every operator is tried out, and for an operator that takes some of the states toward the goals a successor node is created, and the algorithm is recursively called for the states that are reached by applying the operator.

The function *progress* which is give in Figure 4.4 tests for a given operator and a set S of states that for every state $s \in S$ all the successor states are at least one step closer to the goals.

procedure prune(O,W,G);
$i := 0$;
$W_0 := W$;
repeat
  $i := i + 1$;
  $k := 0$;
  $S_0 := G$;   *(* States from which G is reachable with 0 steps. *)*
  repeat
    $k := k + 1$;   *(* States from which G is reachable with $\leq k$ steps. *)*
    $S_k := S_{k-1} \cup \bigcup_{o \in O}(wpreimg_o(S_{k-1}) \cap spreimg_o(W_{i-1}))$;
  until $S_k = S_{k-1}$;   *(* States that stay within W_{i-1} and eventually reach G. *)*
  $W_i := W_{i-1} \cap S_k$;
until $W_i = W_{i-1}$;   *(* States in W_i stay within W_i and eventually reach G. *)*
return W_i;

Figure 4.5: Algorithm for detecting a loop that eventually makes progress

4.3.2 An algorithm for constructing plans with loops

There are many nondeterministic planning problems that require plans with loops because there is no finite upper bound on the number of actions that might be needed for reaching the goals. These plan executions with an unbounded length cannot be handled in acyclic plans of a finite size. For unbounded execution lengths we have to allow loops (cycles) in the plans.

Example 4.22   ■

The problem is those states that do not have a finite strong distance as defined Section 4.3.1. Reaching the goals from these states is either impossible or there is no finite upper bound on the number of actions that might be needed. For the former states nothing can be done, but the latter states can be handled by plans with loops.

We present an algorithm based on a generalized notion of distances that does not require reachability by a finitely bounded number of actions. The algorithm is based on the procedure *prune* that identifies a set of states for which reaching a goal state eventually is guaranteed. The procedure *prune* given in Figure 4.5.

Lemma 4.23 (Procedure prune) *Let O be a set of operators and W and G sets of states. Then $W' = prune(O,W,G)$ is a set such that $W' \subseteq W$ and there is function $x : W' \to O$ such that*

    1. for every $s \in W'$ there is a sequence $s_0, s_1, \ldots, s_n$ with $n \geq 0$ such that $s = s_0$, $s_n \in G$ and $s_{i+1} \in img_{x(s_i)}(\{s_i\})$ for all $i \in \{0, \ldots, n-1\}$,

    2. $img_{x(s)}(\{s\}) \subseteq W'$ for every $s \in W' \backslash G$, and

    3. for no $s \in W \backslash W'$ there is a plan that guarantees reaching a state in G.

Proof:

Let W_0 be the value of W when the procedure is called, and $W_1, W_2, \ldots$ the values of W at the end of the *repeat-until* loop on each iteration.

Induction hypothesis: If $i \geq 1$ then there is function $x : W_i \to O$ such that

1. for every $s \in W_i$ there is a sequence $s_0, s_1, \ldots, s_n$ with $n \geq 0$ such that $s = s_0$, $s_n \in G$ and $s_{j+1} \in img_{x(s_j)}(\{s_j\})$ for all $j \in \{0, \ldots, n-1\}$, and

2. $img_{x(s)}(\{s\}) \subseteq W_{i-1}$ for every $s \in W_i \backslash G$.

Base case $i = 0$: Trivial because nothing about W_i is claimed.

Inductive case $i \geq 1$:

For the inner *repeat-until* loop we prove inductively the following. Let $S_0 = G$ be the value of S before the loop, and $S_1, S_2, \ldots$ the values of S in the end of each iteration.

Induction hypothesis: If $i \geq 1$ then there is function $x : S_k \to O$ such that

1. for every $s \in S_k$ there is a sequence $s_0, s_1, \ldots, s_n$ with $n \in \{0, \ldots, k\}$ such that $s = s_0$, $s_n \in G$ and $s_{j+1} \in img_{x(s_j)}(\{s_j\})$ for all $j \in \{0, \ldots, n-1\}$, and

2. $img_{x(s)}(\{s\}) \subseteq W_{i-1}$ for every $s \in S_k \backslash G$.

Base case $k = 0$:

1. Because $S_0 = G$, for every $s \in S_0$ there is the sequence of states $s_0 = s$ such that the initial state is in S_0 and the final state is in G.

2. Because $S_0 = G$ there are no states in $S_0 \backslash G$.

Inductive case $k \geq 1$: Let s be a state in S_k. If $s \in S_{k-1}$ then we obtain the property by the induction hypothesis.

Otherwise $s \in S_k \backslash S_{k-1}$. Therefore by definition of S_k, $s \in wpreimg_o(S_{k-1}) \cap spreimg_o(W_{i-1})$ for some $o \in O$.

1. Because $s \in wpreimg_o(S_{k-1})$, there is a state $s' \in S_{k-1}$ such that $s' \in img_o(\{s\})$. By the induction hypothesis there is a sequence of states starting from s' that ends in a goal state. For s such a sequence is obtained from the sequence of s' by prefixing it with s. The corresponding operator is assigned to s by x.

2. Because $s \in spreimg_o(W_{i-1})$, by Lemma 4.10 $img_o(\{s\}) \subseteq W_{i-1}$.

This completes the inner induction. To establish the induction step of the outer induction consider the following. The inner repeat-until loops ends when $S_k = S_{k-1}$. This means that $S_z = S_k$ for all $z \geq k$. Hence the upper bound $n \leq k$ on the length of sequences $s_0, s_1, \ldots, s_n$ is infinite. The outer induction hypothesis is obtained from the inner induction hypothesis by removing the upper bound $n \leq k$ and replacing S_k by W_i. By definition $W_i = W_{i-1} \cap S_k$. **What happens here???**

kesken This finishes the outer induction proof. The claim of the lemma is obtained from the outer induction hypothesis by noticing that the outer loop exits when $W_i = W_{i-1}$ (it will exit after a finite number of steps because the cardinality of $W_0 = W$ is finite and it decreases on every iteration) and then we can replace both W_i and W_{i-1} by W' to obtain the claim of the lemma. $\square$

Our first algorithm, given in Figure 4.6, is directly based on the procedure *prune* and identifying a set of states from which a goal state is reachable by some execution and no execution leads to a state outside the set.

procedure FOplanL2(I,O,G)
$W_0 := G$;
$i := 0$;
while $I \not\subseteq$ prune(O,W_i,G) **and** ($i = 0$ **or** $W_{i-1} \neq W_i$) **do**
    $i := i + 1$;
    $W_i := W_{i-1} \cup \bigcup_{o \in O} wpreimg_o(W_{i-1})$;
end
$S := G$;
$i := 0$;
$D_i := G$;
$L := $ prune(O,W_i,G);
repeat
    $S' := S$;
    $S := S \cup \bigcup_{o \in O}(wpreimg_o(S) \cap spreimg_o(L \cup S))$;
    $i := i + 1$;
    $D_i := L \cap S$;
until $S = S'$

Figure 4.6: Algorithm for nondeterministic planning with full observability

procedure FOplanMAINTENANCE(I,O,G)
$i := 0$;
$G_0 := G$;
repeat
    $i := i + 1$;   *(* Subset of G_{i-1} from which G_{i-1} can be always reached. *)*
    $G_i := \bigcup_{o \in O} (spreimg_o(G_{i-1}) \cap G_{i-1})$;
until $G_i = G_{i-1}$;
return G_i;

Figure 4.7: Algorithm for nondeterministic planning with full observability and maintenance goals

4.3.3   An algorithm for constructing plans for maintenance goals

There are many important planning problems in which the objective is not to reach a goal state and then stop execution. When the objective is to keep the state of the system in any of a number of goal states indefinitely, we talk about *maintenance goals*.

Plans that satisfy a maintenance goal have only infinite executions.

Figure 4.7 gives an algorithm for finding plans for maintenance goals. The algorithm starts with the set G of all states that satisfy the property to be maintained. Then iteratively such states are removed from G for which the satisfaction of the property cannot be guaranteed in the next time point. More precisely, the sets G_i for $i \geq 0$ consist of all those states in which the goal objective can be maintained for the next i time points. For some i the sets G_i and G_{i-1} coincide, and then $G_j = G_i$ for all $j \geq i$. This means that starting from the states in G_i the goal objective can be maintained indefinitely.

Theorem 4.24 *Let I be a set of initial states, O a set of operator and G a set of goal states. Let G' be the set returned by the procedure FOplanMAINTENANCE in Figure 4.7.*
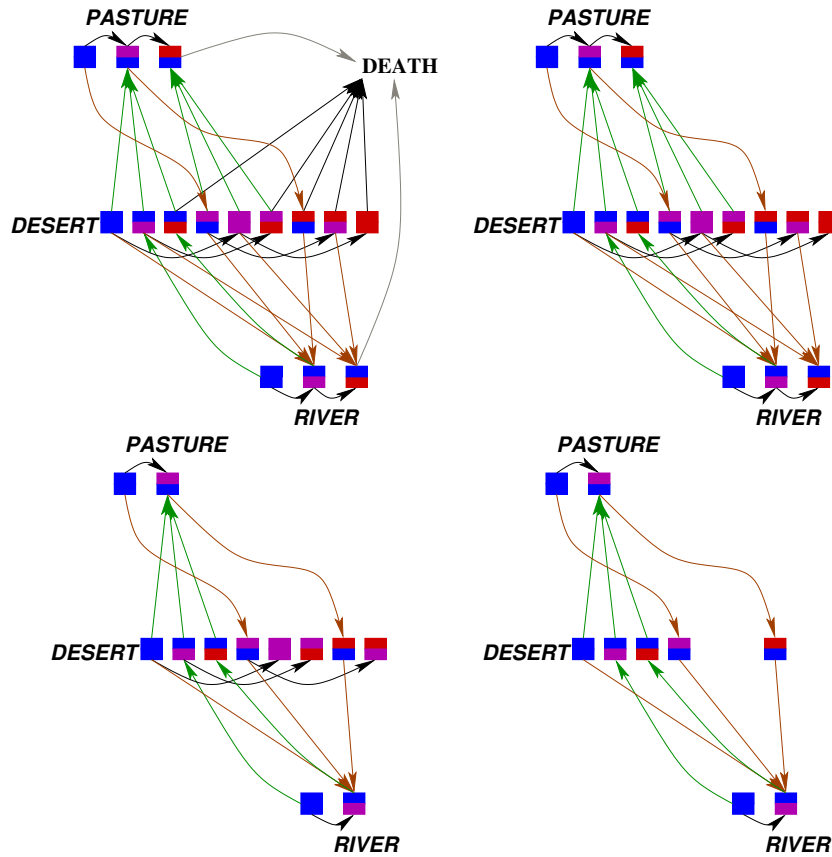
Figure 4.8: Example run of the algorithm for maintenance goals

Then for every state $s \in G'$ there is an operator $o \in O$ such that $img_o(\{s\}) \subseteq G'$. If $I \subseteq G'$ then the corresponding plan satisfies the maintenance criterion for $\langle I, O, G \rangle$.

Proof: $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Example 4.25 Consider the problem depicted in Figure 4.8. An animal may drink at a river and eat at a pasture. To get from the river to the pasture it must go through a desert. Its hunger and thirst increase after every time period. If either one reaches level 3 the animal dies. The hunger and thirst levels are indicated by different colors: the upper halves of the rectangles show thirst level and the lower halves the hunger level, and blue means no hunger or thirst and red means much hunger or thirst. The upper left diagram shows all the possible actions the animal can take. The objective of the animal is to stay alive. The three iterations of the algorithm for finding a plan that satisfies the goal of staying alive are depicted by the remaining three diagrams. The diagram on upper right depicts all the states that satisfy the goal. The diagram on lower left depicts all the states that satisfy the goal and after which the satisfaction of the goal can be guaranteed for at least one time period. The diagram on lower right depicts all the states that satisfy the goal and after which the satisfaction of the goal can be guaranteed for at least two time periods.

Further iterations of the algorithm do not eliminate further states, and hence the last diagram depicts all those states for which the satisfaction of the goal can be guaranteed indefinitely.
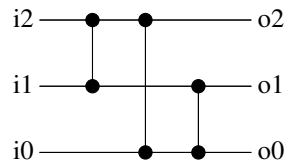
Figure 4.9: A sorting network with three inputs

Hence the only plan says that the animal has to go continuously back and forth between the pasture and the river. The only choice the animal has is in the beginning if in the initial state it is not at all hungry or thirsty. For instance, if it is in the desert initially, then it may freely choose whether to first go to the pasture or the river.   ∎

4.4   Planning with partial observability

4.4.1   Planning without observability by heuristic search

Planning under unobservability is similar to deterministic planning in the sense that the problem is to find a path from the initial state(s) to the goal states. For unobservable planning, however, the nodes in the graph do not correspond to individual states but to belief states, and the size of the belief space is exponentially higher than the size of the state space. Algorithms for deterministic planning have direct counterparts for unobservable planning, which is not the case for conditional planning with full or partial observability.

Example 4.26 A sorting network [Knuth, 1998, Section 5.3.4 in 2nd edition] consists of a sequence of gates acting on a number of input lines. Each gate combines a comparator and a swapper: if the first value is greater than the second, then swap them. The goal is to sort any given input sequence. The sorting network always has to perform the same operations irrespective of the input, and hence constructing a sorting network corresponds to planning without observability. Figure 4.9 depicts a sorting network with three inputs. An important property of sorting networks is that any network that sorts any sequence of zeros and ones will also sort any sequence of arbitrary numbers. Hence it suffices to consider Boolean 0-1 input values only.

Construction of sorting networks is essentially a planning problem without observability, because there are several initial states and a goal state has to be reached by using the same sequence of actions irrespective of the initial states.

For the 3-input sorting net the initial states are $000, 001, 010, 011, 100, 101, 110, 111$. and the goal states are $000, 001, 011, 111$ Now we can compute the images and strong preimages of the three sorting actions, sort12, sort02 and sort01 respectively starting from the initial or the goal states. These yield the following belief states at different stages of the sorting network.

$000, 001, 010, 011, 100, 101, 110, 111$	initially
$000, 001, 011, 100, 101, 111$	after sort12
$000, 001, 011, 101, 111$	after sort02
$000, 001, 011, 111$	after sort01

∎

The most obvious approaches to planning with unobservability is to use regression, strong preimages or images, and to perform backward or forward search in the belief space. The dif-

ference to forward search with deterministic operators and one initial state is that belief states are used instead of states. The difference to backward search for deterministic planning is that regression for nondeterministic operators has to be used and testing whether (a subset of) the initial belief state has been reached involves the co-NP-hard inclusion test $\models I \rightarrow regr_o(\phi)$ for the belief states. With one initial state this is an easy polynomial time test $I \models regr_o(\phi)$ of whether $regr_o(\phi)$ is true in the initial state.

Deriving good heuristics for heuristic search in the belief space is more difficult than in deterministic planning. The main approaches have been to use distances in the state space as an estimate for distances in the belief space, and to use the cardinalities of belief spaces as a measure of progress.

Many problems cannot be solved by blindly taking actions that reduce the cardinality of the current belief state: the cardinality of the belief state may stay the same or increase during plan execution, and hence the decrease in cardinality is not characteristic to belief space planning in general, even though in many problems it is a useful measure of progress.

Similarly, distances in the state space ignore the most distinctive aspect of planning with partial observability: the same action must be used in two states if the states are not observationally distinguishable. A given (optimal) plan for an unobservable problem may increase the actual current state-space distance to the goal states (on a given execution) when the distance in the belief-space monotonically decreases, and vice versa. Hence, the state space distances may yield wildly misleading estimates of the distances in the corresponding belief space.

Heuristics based on state-space distances

The most obvious distance heuristics are based on the strong distances in the state space.

$$
\begin{aligned}
D_0 &= G \\
D_{i+1} &= D_i \cup \bigcup_{o \in O} spreimg_o(D_i) \text{ for all } i \geq 1
\end{aligned}
$$

A lower bound on plan length for belief state Z is j if $Z \subseteq D_j$ and $Z \not\subseteq D_{j-1}$.

Next we derive distance heuristics for the belief space based on state space distances. Strong distances yield an admissible distance heuristic for belief states.

Definition 4.27 (State space distance) *The* state space distance *of a belief state B is $d \geq 1$ when $B \subseteq D_d$ and $B \not\subseteq D_{d-1}$, and it is 0 when $B \subseteq D_0 = G$.*

Even though computing the exact distances for the operator based representation of state spaces is PSPACE-hard, the much higher complexity of planning problems with partial observability still often justifies it: this computation would in many cases be an inexpensive preprocessing step, preceding the much more expensive solution of the partially observable planning problem. Otherwise cheaper approximations can be used.

Heuristics based on belief state cardinality

The second heuristic that has been used in algorithms for partial observability is simply based on the cardinality of the belief states.

In forward search, prefer operators that maximally decrease the cardinality of the belief state.

In backward search, prefer operators that maximally increase the cardinality of the belief state.

These heuristics are not in general admissible, because there is no direct connection between the distance to a goal belief state and the cardinalities of the current belief state and a goal belief state. The belief state cardinality can decrease or increase arbitrarily much by one step.

4.4.2 Planning without observability by evaluation of QBF

In this section we extend the techniques from Section 3.5 to unobservable planning. Because of nondeterminism and several initial states, one plan may have several different executions. It turns out that propositional logic is not suitable for representing planning with unobservability, and the language of quantified Boolean formulae is needed instead. Intuitively, the reason for this is that we have to quantify over an exponential number of plan executions: we want to say "there is a plan so that for all executions...", and expressing this concisely in the propositional logic does not seem possible. We theoretically justify this in Section 4.5.4 by showing that testing the existence of plans for problems instances without observability even when restricting to plans with a polynomial length is complete for the complexity class Σ_2, and not contained in NP as the corresponding problem for deterministic planning. This strongly suggests, because of widely accepted complexity theoretic conjectures, that there is no efficient representation of the problem in the propositional logic.

In Section 4.1.2 we showed how nondeterministic operators can be translated into formulae in the propositional logic. The purpose of that translation was the use of the formulae in BDD-based planning algorithms for computing the images and preimages of sets of states. For the QBF representation of nondeterministic operators we have to have a possibility to universally quantify over all possible successor states an operator produces, and this cannot be easily expressed with the formulae derived in Section 4.1.2, so we give a new translation that uses quantified Boolean formulae (see Section 2.2.1.)

Translation of nondeterministic operators into propositional logic

For handling nondeterminism 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. For an effect with n nondeterministic alternatives this can be achieved by using $m = \lceil \log_2 n \rceil$ universally quantified auxiliary variables. For every valuation of these variables one of the alternative effects is chosen.

We assign to every atomic effect a formula that is true if and only if that effect takes place. This is similar to and extends the functions $EPC_l(e)$ in Definition 3.3. The extension concerns nondeterminism: for literal l to become true, the auxiliary variables for nondeterminism have to have values corresponding to an effect making l true.

The condition for atomic effect l to take place when effect e is executed is $nEPC_l(e, \sigma, t)$. The sequence σ of integers is for deriving unique names for auxiliary variables in $nEPC_l(e, \sigma, t)$, and t is formula on the auxiliary variables for deciding when to execute e. The effect is assumed to be in normal form I.

$$
\begin{aligned}
nEPC_l(e, \sigma, t) &= EPC_l(e) \wedge t \text{ if } e \text{ is deterministic} \\
nEPC_l(p_1 e_1 | \cdots | p_n e_n, \sigma, t) &= nEPC_l(e_1, \sigma; 1, c_1^n(\sigma) \wedge t) \vee \cdots \vee nEPC_l(e_n, \sigma; n, c_n^n(\sigma) \wedge t) \\
nEPC_l(e_1 \wedge \cdots \wedge e_n, \sigma, t) &= nEPC_l(e_1, \sigma; 1, t) \vee \cdots \vee nEPC_l(e_n, \sigma; n, t)
\end{aligned}
$$

The function $c_i^n(\sigma)$ constructs a formula for selecting the ith effect from n alternatives. This formula is usually a conjunction of literals over the auxiliary propositions $A_\sigma^m = \{a_{\sigma,1}, \ldots, a_{\sigma,m}\}$

corresponding to one valuation of A_σ^m. Here $m = \lceil \log_2 n \rceil$. When n is not a power of 2, the last effect e_n corresponds to more than one valuation of A_σ^m. Define

$$d_i^m(\sigma) = \bigwedge(\{a_{\sigma,j} \in A_\sigma^m | j\text{th bit of } i - 1 \text{ is } 1\}\{\neg a_{\sigma,j} | a_{\sigma,j} \in A_\sigma^m, j\text{th bit of } i - 1 \text{ is } 0\}).$$

When $i \in \{1, \ldots, n - 1\}$ (and in the special case $i = n = 2^m$), we define $c_i^n(\sigma)$ as $d_i^m(\sigma)$. When $i = n$ we define $c_i^n(\sigma)$ as

$$d_n^m(\sigma) \vee \cdots \vee d_{2^m}^m(\sigma)$$

Hence effects e_1 to e_{n-1} correspond to binary encodings of numbers 0 to $n - 2$ and e_n covers all the remaining valuations of A_σ^m.

The following frame axioms express the conditions under which the state variable $a \in A$ may change from false to true and from true to false. We assume that the operators in $O = \{o_1, \ldots, o_n\}$ have a unique numbering $1, \ldots, n$.

$$(\neg a \wedge a') \rightarrow ((o_1 \wedge nEPC_a(e_1, 1, \top)) \vee \cdots \vee (o_n \wedge nEPC_a(e_n, n, \top)))$$
$$(a \wedge \neg a') \rightarrow ((o_1 \wedge nEPC_{\neg a}(e_1, 1, \top)) \vee \cdots \vee (o_n \wedge nEPC_{\neg a}(e_n, n, \top)))$$

For every operator $o = \langle z, e \rangle \in O$ we have formulae for describing values of state variables in the predecessor and in the successor states when the operator is applied. Let $A = \{a_1, \ldots, a_k\}$ be the state variables. The formulae describing the effects and preconditions of the operator $o_i \in O$ are the following.

$$
\begin{aligned}
(o_i \wedge nEPC_{a_1}(e_i, i, \top)) &\rightarrow a_1' \\
(o_i \wedge nEPC_{\neg a_1}(e_i, i, \top)) &\rightarrow \neg a_1' \\
&\vdots \\
(o_i \wedge nEPC_{a_k}(e_i, i, \top)) &\rightarrow a_k' \\
(o_i \wedge nEPC_{\neg a_k}(e_i, i, \top)) &\rightarrow \neg a_k' \\
o_i &\rightarrow z
\end{aligned}
$$

Example 4.28 Consider the operators $o_1 = \langle A, (0.5B|0.5(C \rhd D)) \rangle$ and $o_2 = \langle B, (0.5(D \rhd B)|0.5C) \rangle$. The application of these operators is described by the following formula.

example missing ∎

Two operators may be applied in parallel only if they do not interfere, so we have

$$\neg o_i \vee \neg o_j$$

for all operators i and j such that $i \neq j$ and the operators interfere.

The conjunction of all the above formulae is denoted by

$$\mathcal{R}_3(A, A')$$

When renaming the propositions for time point t, also the propositions o for operators $o \in O$ and the propositions $a \in A$ must be renamed, and for this we use then notation

$$\mathcal{R}_3^t(A^t, A^{t+1}).$$

Finding plans by evaluating QBF

In deterministic planning in propositional logic (Section 3.5) 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 is nondeterminism, the problem is to find a seqence of operators so that a goal state is reached for all possible executions of the sequence of operators. The number of executions of one sequence of operators may be higher than one because there may be several initial states and because the operators may be deterministic. Expressing the quantification over all possible executions of a sequence of operators cannot be concisely expressed in the propositional logic, and this is the reason why quantified Boolean formulae have to be used instead.

$$\exists V_{plan}$$
$$\forall V_{exec}$$
$$\exists V_{rest}$$
$$I^0 \rightarrow (\mathcal{R}_3(A_0, A_1) \wedge \mathcal{R}_3(A_1, A_2) \wedge \cdots \wedge \mathcal{R}_3(A_{n-1}, A_n) \wedge G^n)$$

Here $V_{exec} = A_0 \cup A^0 \cup \cdots \cup A^{t-1}$ where A is the set of auxiliary variables occurring in $nEPC_l(e, \epsilon, \top)$ for some $\langle c, e \rangle \in O$ and $l \in \{a, \neg a\}$ for some $a \in A$. The plan is expressed in terms of the variables o^i where $o \in O$ and $i \in \{0, \ldots, t-1\}$. The truth-values of the remaining variables $V_{rest} = A_1 \cup \cdots \cup A_t$ are determined by the operators and the execution chosen by propositions in V_{exec}.

There are algorithms for evaluating QBF that extend the Davis-Putnam procedure and that traversing and-or trees. And-nodes correspond to universally quantified propositions and or-nodes correspond to existentially quantified propositions. These algorithms return the valuation of the outermost existential propositions if the QBF has value *true*.

Finding plans for nondeterministic problems without observability may be more efficient than using standard search algorithms with regression or image/preimage computation with BDDs when the plans are short and there are many operators that can be applied in parallel. If long plans are required and there is little parallelism, the algorithms that traverse the belief space appear to be more efficient.

4.4.3   Algorithms for planning with partial observability

Planning with partial observability is much more complicated than its two special cases with full and no observability. Like planning without observability, the notion of belief states becomes very important. Like planning with full observability, formalization of plans as sequences of operators is insufficient. However, plans also cannot be formalized as mappings from states to operators because partial observability implies that the current state is not necessarily unambiguously known. Hence we will need the general definition of plans introduced in Section 4.2.1.

When executing operator o in belief state B the set of possible successor states is $img_o(B)$, and based on the observation that are made, this set is restricted to $B' = img_o(B) \cap C$ where C is the equivalence class of observationally indistinguishable states corresponding to the observation.

In planning with unobservability, a backward search algorithm starts from the goal belief state and uses regression or strong preimages for finding predecessor belief states until a belief state covering the initial belief state is found.

With partial observability, plans do not just contain operators but may also branch. With branching the sequence of operators may depend on the observations, and this makes it possible to reach

goals also when no fixed sequence of operators reaches the goals. Like strong preimages in backward search correspond to images, the question arises what does branching correspond to in backward search?

Assume that we have for belief states B_1 and B_2 respectively the plans π_1 and π_2 that reach the goals, and that these belief states are observationally distinguishable, that is, they are included in different observational classes. Now we can construct a plan π_{12} that starts with a branch node that makes an observation and continues with π_1 or with π_2, depending on which observation was made. If we are initially in any state in $B_1 \cup B_2$, the plan π_{12} always takes us to a goal state. We can continue extending π_{12} with operators. For example, if $B = \text{wpreimg}_o(B_1 \cup B_2)$, then the plan that first executes the operator o and then continues with π will lead to a goal state starting from any state in B.

Next we formalize these ideas and derive an algorithm that constructs branching plans in the backward direction starting from the goal states.

Let $\Pi = \langle C_1, \ldots, C_n \rangle$ be a partition of the state space to observational classes, each consisting of observationally indistinguishable states.

Sets of belief states generated by traversing the belief space backwards starting from the goal states contain many regularities induced by observability. For example, if we have plans for reaching the goals from three belief states B_1, B_2 and B_3, and these have non-empty intersections with the n observational classes, we may construct 3^n different branching plans for 3^n different sets of states. These 3^n sets have a concise representation in a factored form, simply as

$$\langle \{B_1 \cap C_1, B_2 \cap C_1, B_3 \cap C_1\}, \{B_1 \cap C_2, B_2 \cap C_2, B_3 \cap C_2\}, \ldots, \{B_1 \cap C_n, B_2 \cap C_n, B_3 \cap C_n\}\rangle$$

from which the sets can be obtained by taking the Cartesian product and then the union of the n components of each of the 3^n tuples. This motivates the following definitions.

Definition 4.29 (Factored belief space) *Let $\Pi = \langle C_1, \ldots, C_n \rangle$ be a partition of the set of all states. Then a factored belief space is $\langle G_1, \ldots, G_n \rangle$ where $s \subset s'$ for no $\{s, s'\} \subseteq G_i$ and $G_i \subseteq 2^{C_i}$ for all $i \in \{1, \ldots, n\}$.*

Intuitively, a factored belief space is a set of belief states, partitioned to subsets corresponding to the observational classes. This is just a technical definition that makes it easier to talk about the belief states corresponding to the same observational class. Notice the minimality condition: none of the belief states in a factored belief space may be a subset of another. We want to have the minimality condition because we use factored belief spaces as representations of those sets of states for which a plan exists. If a plan exists for some belief state B, then the same plan also works for any belief state B' such that $B' \subseteq B$.

The factored representation of a one-element set S of states is simply $\mathcal{F}(S) = \langle \{C_1 \cap S\}, \ldots, \{C_n \cap S\}\rangle$. When it is obvious from the context, we often write simply S instead of $\mathcal{F}(S)$.

When we have two sets of belief states in the factored form, we may combine them and keep the result in the factored form.

Definition 4.30 (Combination of factored belief spaces) *Let $G = \langle G_1, \ldots, G_n \rangle$ and $H = \langle H_1, \ldots, H_n \rangle$ be factored belief spaces. Define $G \oplus H$ as $\langle G_1 \uplus H_1, \ldots, G_n \uplus H_n \rangle$, where the operation $\uplus$ takes union of two sets of sets and eliminates sets that are not set-inclusion maximal. It is formally defined as $G \uplus H = \{R \in G \cup H \mid R \subset K \text{ for no } K \in G \cup H\}$.*

Important in this combination operation is that the minimality condition is preserved: any belief state that is a subset of another belief state is eliminated.

The combination operator has the following properties.

Lemma 4.31 (Belief spaces with $\oplus$ are commutative monoids) *The operator $\oplus$ is associative, commutative and its identity element is $\langle \emptyset, \ldots, \emptyset \rangle$.*

A factored belief space $G = \langle G_1, \ldots, G_n \rangle$ can be viewed as representing the set of sets of states $\mathrm{flat}(G) = \{s_1 \cup \cdots \cup s_n | s_i \in G_i \text{ for all } i \in \{1, \ldots, n\}\}$, and its cardinality is $|G_1| \cdot |G_2| \cdot \ldots \cdot |G_n|$. The cardinality may be exponential on the size of the factored representation. Assuming that we have a plan for all belief states in G, we also have a plan for any sets in $\mathrm{flat}(G)$. This plan starts by a branch according to an observation C that is made, and then follows the plan for the respective belief state $B \cap C$.

Definition 4.32 (Inclusion relation on belief spaces) *A factored belief space G is included in factored belief space H if for all $S \in \mathrm{flat}(G)$ there is $S' \in \mathrm{flat}(H)$ such that $S \subseteq S'$. We write this $G \sqsubseteq H$.*

The definitions have the property that $S \in \mathrm{flat}(G)$ if and only if $\mathcal{F}(S) \sqsubseteq G$.

We discuss the complexity of certain operations on belief spaces. The basic operations needed in the planning algorithms are testing the membership of a set of states in a factored belief space, and finding a set of states whose preimage with respect to an operator is not contained in the belief space. This last operation is needed in the backup steps of our planning algorithm: find a plan that covers belief states for which we did not have a plan earlier.

Theorem 4.33 *Testing $G \sqsubseteq H$ for factored belief spaces G and H is polynomial time.*

Proof: Testing $\langle G_1, \ldots, G_n \rangle \sqsubseteq \langle H_1, \ldots, H_n \rangle$ is simply by testing whether for all $i \in \{1, \ldots, n\}$ and all $s \in G_i$ there is $t \in H_i$ such that $s \subseteq t$. $\square$

Example 4.34 Consider the blocks world with three blocks with the goal state in which all the blocks are on the table. There are three operators, each of which picks up one block (if there is nothing on top of it) and places it on the table. We can only observe which blocks are not below another block. This splits the state space to seven observational classes, corresponding to the valuations of the state variables clear-A, clear-B and clear-C in which at least one block is clear.

The plan construction steps are given in Figure 4.10. Starting from the top left, the first diagram depicts the goal belief state. The second diagram depicts the belief states obtained by computing the strong preimage of the goal belief state with respect to the move-A-onto-table action and splitting the set of states to belief states corresponding to the observational classes. The next two diagrams are similarly for strong preimages of move-B-onto-table and move-C-onto-table.

The fifth diagram depicts the computation of the strong preimage from the union of two existing belief states in which the block A is on the table and C is on B or B is on C. In the resulting belief state A is the topmost block in a stack containing all three blocks. The next two diagrams similarly construct belief states in which respectively B and C are the topmost blocks.

The last three diagrams depict the most interesting cases, constructing belief states that subsume two existing belief states in one observational class. The first diagram depicts the construction of the belief state consisting of both states in which A and B are clear and C is under either A or B.
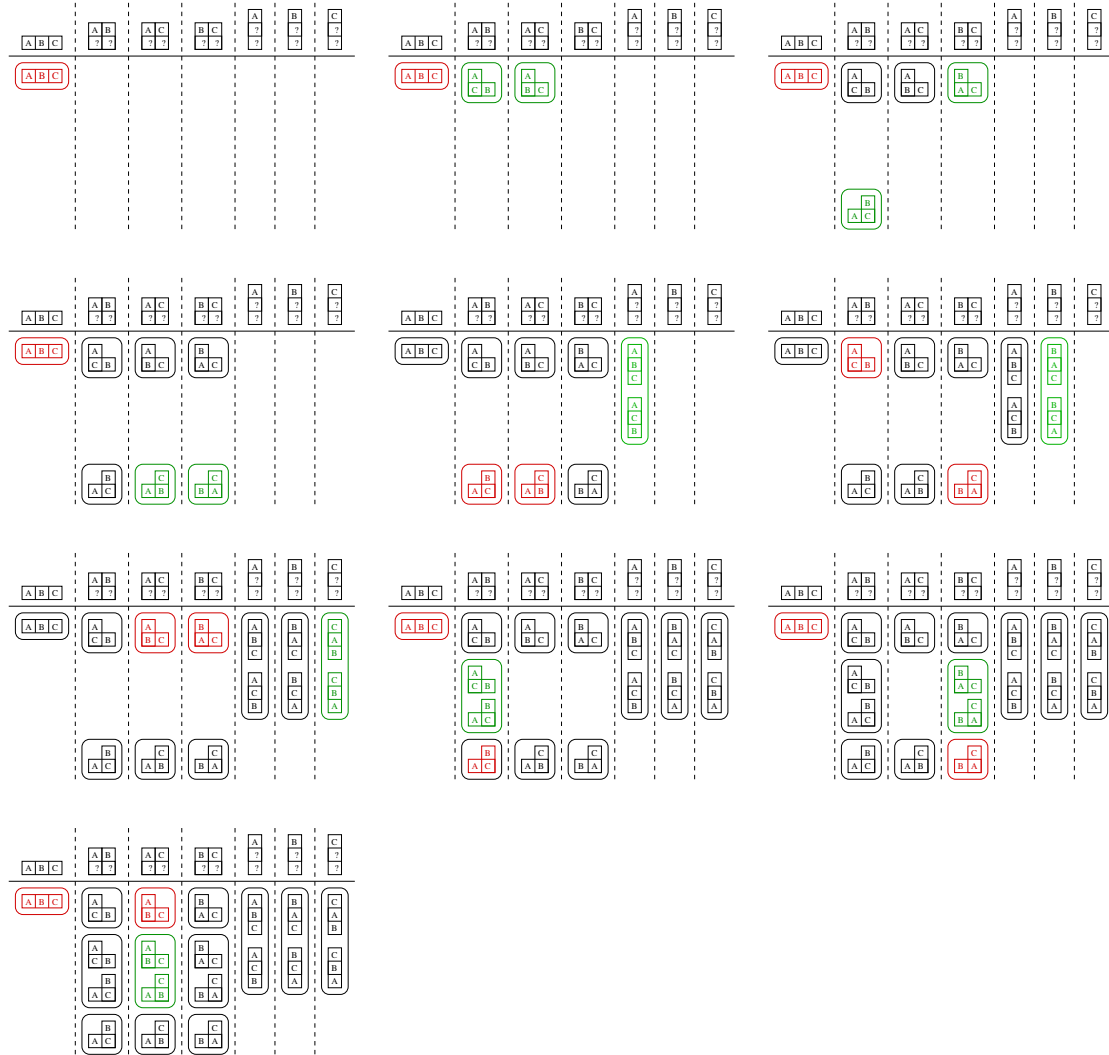
Figure 4.10: Solution of a simple blocks world problem

This belief state is obtained as the strong preimage of the union of two existing belief states, the one in which all blocks are on the table and the one in which A is on the table and B is on top of C. The action that moves A onto the table yields the belief state because if A is on C all blocks will be on the table and if A is already on the table nothing will happen. Construction of the belief states in which B and C are clear and A and C are clear is analogous and depicted in the last two diagrams.

The resulting plan reaches the goal state from any state in the blocks world. The plan in the program form is given in Figure 4.11 (order of construction is from the end to the beginning.)

∎

The algorithm we give for extending factored belief spaces by computing the preimage of a combination of some of its belief states is based on exhaustive search and runs in worst-case exponential time. The algorithm is justified by the following theorem that shows that finding new belief states is NP-hard. The proof is a reduction from SAT: represent each clause as the set of

```
16:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-C THEN GOTO 15
  IF clear-B AND clear-C THEN GOTO 13
  IF clear-A AND clear-B THEN GOTO 11
  IF clear-A THEN GOTO 5
  IF clear-B THEN GOTO 7
  IF clear-C THEN GOTO 9
15:
  move-C-onto-table
14:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-C THEN GOTO 1
13:
  move-B-onto-table
12:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-B AND clear-C THEN GOTO 3
11:
  move-A-onto-table
10:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-B THEN GOTO 2
9:
  move-C-onto-table
8:
  IF clear-A AND clear-C THEN GOTO 1
  IF clear-B AND clear-C THEN GOTO 2
7:
  move-B-onto-table
6:
  IF clear-A AND clear-B THEN GOTO 1
  IF clear-B AND clear-C THEN GOTO 3
5:
  move-A-onto-table
4:
  IF clear-A AND clear-B THEN GOTO 2
  IF clear-A AND clear-C THEN GOTO 3
3:
  move-C-onto-table
  GOTO end
2:
  move-B-onto-table
  GOTO end
1:
  move-A-onto-table
end:
```

Figure 4.11: A plan for a partially observable blocks world problem

literals that are not in it, and then a satisfying assignment is a set of literals that is not included in any of the sets.

Theorem 4.35 *Testing whether $G = \langle G_1, \ldots, G_n \rangle$ contains a set S of states such that $spreimg_o(S)$ is not in G is NP-complete. This holds also for deterministic operators o.*

Proof: Membership in NP is trivial: nondeterministically choose $s_i \in G_i$ for every $i \in \{1, \ldots, n\}$, compute the preimage r of $s_1 \cup \cdots \cup s_n$ in deterministic polynomial time, and verify in polynomial time that the intersection $r \cap C_i$ of the preimage with one of the observational classes C_i is not in G_i.

Let $T = \{E_1, \ldots, E_m\}$ be a set of clauses over a set of propositional variables $A = \{a_1, \ldots, a_k\}$. We construct a factored belief space based on a state space in which the variables a and $\widehat{a}$ for $a \in A$ and all these variables with a replaced by z, are the states. The variables $\widehat{z}$ represent negative literals. Define

$$
\begin{aligned}
E'_i &= (A \backslash E_i) \cup \{\widehat{a} | a \in A, \neg a \notin E_i\} \text{ for } i \in \{1, \ldots, m\} \\
G &= \langle \{E'_1, \ldots, E'_m\}, \{\{z_1\}, \{\widehat{z}_1\}\}, \ldots, \{\{z_k\}, \{\widehat{z}_k\}\} \rangle
\end{aligned}
$$

Let o map a_i to z_i and $\widehat{a}_i$ to $\widehat{z}_i$ for all $i \in \{1, \ldots, k\}$.

We claim that T is satisfiable if and only if flat(G) contains a belief state B such that $spreimg_o(B)$ is not in G.

Assume T is satisfiable, that is, there is M such that $M \models T$. Define $M' = \{z_i | a_i \in A, M \models a_i\} \cup \{\widehat{z}_i | a_i \in A, M \not\models a_i\}$. Clearly, M' is a belief state in G. Define $M'' = \{a_i \in A | M \models a_i\} \cup \{\widehat{a}_i | a_i \in A, M \not\models a_i\}$. Clearly, M'' is the preimage of M' with respect to o.

We show that M'' is not in G. Take any $i \in \{1, \ldots, m\}$. Because $M \models E_i$, there is $a_j \in A$ such that $a_j \in E_i$ and $M \models a_j$ (the case $\neg a \in E_i$ goes similarly.) Now $a_j \in M''$. By definition now $a_j \notin E'_j$. As this holds for all $i \in \{1, \ldots, m\}$, M'' is not a subset of any E_i, and hence it does not belong to G.

Assume there is belief state B in G such that the preimage of B with respect to o is not in G. Clearly, B is a subset of $A \cup \{\widehat{a} | a \in A\}$ with at most one of a_i or $\widehat{a}_i$ for any $i \in \{1, \ldots, k\}$. Define a propositional model M such that $M \models a$ if and only if $a \in B$. We show that $M \models T$. Take any clause E_i from T. As B is not in G, $B \not\subseteq E'_i$. Hence there is a_j or $\widehat{a}_j$ in $B \backslash E'_i$. Consider the case with a_j ($\widehat{a}_j$ goes similarly.) As $a_j \notin E'_i$, $a_j \in E_i$. By definition of M, $M \models a_j$ and hence $M \models E_i$. As this holds for all $i \in \{1, \ldots, m\}$, $M \models T$. This completes the proof. $\qquad \square$

Example 4.36 The construction in the above proof can be illustrated by the following example. We use an operator that maps a variable x to the variable x_0. Let $T = \{A \vee B \vee C, \neg A \vee B, \neg C\}$. The corresponding factored belief space is

$$
\begin{aligned}
\langle \ & \{\{\widehat{A}, \widehat{B}, \widehat{C}\}, \{A, \widehat{B}, C, \widehat{C}\}, \{A, \widehat{A}, B, \widehat{B}, C\}\}, \\
& \{\{A_0\}, \{\widehat{A}_0\}\}, \\
& \{\{B_0\}, \{\widehat{B}_0\}\}, \\
& \{\{C_0\}, \{\widehat{C}_0\}\} \rangle.
\end{aligned}
$$

$\blacksquare$

procedure findnew(o,A,F,H);
if $F = \langle\rangle$ **and** $spreimg_o(A) \not\subseteq S$ for no $S \in$ flat(H) **then return** A;
if $F = \langle\rangle$ **then return** $\emptyset$;
F is $\langle\{f_1,\ldots,f_m\}, F_2, \ldots, F_k\rangle$ for $k \geq 1$;
for $i := 1$ **to** m **do**
   $S :=$ findnew($o,A \cup f_i, \langle F_2, \ldots, F_k\rangle, H$);
   if $S \neq \emptyset$ **then return** S;
end;
return $\emptyset$

Figure 4.12: An algorithm for finding new belief states

Next we give an algorithm for constructing conditional plans. The basic step in the algorithm is finding a belief state for which a plan can be shown to exist, based on a set of belief states with plans.

The procedure in Figure 4.12 performs this step: it finds a set S of states that is not contained in H and that is the strong preimage of a set S' of states in F with respect to an operator o. The procedure runs in exponential time on the size of F, and consumes space linear in the size of F. By Theorem 4.35 this is the best that can be expected (unless it turns out that $P = NP$).

Lemma 4.37 *The procedure call findnew($o,\emptyset,H,H'$) returns a set S' such that $S' = spreimg_o(S)$ for some $S \in$ flat(H) and $S' \subseteq S''$ for no $S'' \in$ flat(H'), and if no such set exists it returns $\emptyset$.*

Proof: The procedure goes through the elements $\langle S_1, \ldots, S_n\rangle$ of $F_1 \times \cdots \times F_n$ and tests whether $spreimg_o(S_1 \cup \cdots \cup S_n)$ is in H. The sets $S_1 \cup \cdots \cup S_n$ are the elements of flat(F). The traversal through $F_1 \times \cdots \times F_n$ is by generating a search tree with elements of F_1 as children of the root node, elements of F_2 as children of every child of the root node, and testing whether the strong preimage is in it. $\qquad\square$

The implementation of the procedure can be improved in many ways. The sets $f_1, \ldots, f_m$ can be ordered according to cardinality so that the bigger preimages are tried out first and a new belief state is found sooner. Also other kinds of heuristics could be applied here, for example ones that would try to produce belief states closer to the initial state for example according to the heuristics discussed in Section 4.4.1.

Define $altimg_o(S)$ as $img_o(wpreimg_o(S))$. This is the set of states that could have been reached when a state in S was reached instead. Now $S \subseteq altimg_o(S)$, and for deterministic operators $S = altimg_o(S)$.

Pruning techniques based o strong and weak preimages of f_i are the following.

1. Let o be deterministic. If $spreimg_o(f_i) \subseteq spreimg_o(f_j)$ and $i > j$, or $spreimg_o(f_i) \subset spreimg_o(f_j)$, then we can ignore f_i.

   If the strong preimage of f_i is smaller than that of f_j, the strong preimage that is found with f_i cannot be bigger than that with f_j, and hence using f_i is unnecessary.

2. Pruning techniques for nondeterministic operators are more complicated.

   If $wpreimg_o(f_i) \subseteq wpreimg_o(f_j)$ and $altimg_o(f_i) \cap f_i \subseteq altimg_o(f_j)$ and $i > j$, or $wpreimg_o(f_i) \subset wpreimg_o(f_j)$ and $altimg_o(f_i) \cap f_i \subset altimg_o(f_j)$ then f_i can be ignored.

```
procedure plan(I,O,G);
H := F(G);
progress := true;
while progress and I ⊈ S for all S ∈ flat(H) do
  progress := false;
  for each o ∈ O do
    S := findnew(o,∅,H,H);
    if S ≠ ∅ then
      begin
        H := H ⊕ F(spreimg_o(S));
        progress := true;
      end;
  end;
end;
if I ⊆ S for some S ∈ flat(H) then return true
else return false;
```

Figure 4.13: A backward search algorithm for partially observable planning

kesken

A more advanced version of this technique can be utilized during search. If sets included in $C_1, \ldots, C_k$ have already been chosen and their union is B, states $s \in f_i$ such that $altimg_o(\{s\}) \cap ((\bigcup_{i\{1,\ldots,k\}} C_i) \backslash B) \neq \emptyset$ do not help in finding a new (bigger) belief state.

kesken

Figure 4.13 shows an algorithm for finding plans for partially observable problems. The algorithm uses the subprocedure *findnew* for extending the belief space (this is the NP-hard subproblem from Theorem 4.35). The plans the algorithm produces are not guaranteed to be optimal because it does not produce all possible plans in a breadth-first manner.

We have not here described the book-keeping needed for outputting a plan, and the algorithm just returns *true* or *false* depending on whether a plan exists or not. Extending the algorithm with the necessary book-keeping is straightforward.

Lemma 4.38 *Assume $S \in flat(H)$. Then there is $S' \in flat(H \oplus G)$ so that $S \subseteq S'$.*

Lemma 4.39 *Let $S_1, \ldots, S_n$ be sets of states so that for every $i \in \{1, \ldots, n\}$ there is $S'_i \in flat(H)$ such that $S_i \subseteq S'_i$, and there is no observational class C such that for some $\{i, j\} \subseteq \{1, \ldots, n\}$ both $i \neq j$ and $S_i \cap C \neq \emptyset$ and $S_j \cap C \neq \emptyset$. Then there is $S' \in flat(H)$ such that $S_1 \cup \cdots \cup S_n \subseteq S'$.*

Theorem 4.40 *Whenever there exists a finite acyclic plan for a problem instance, the algorithm in Figure 4.13 returns* true.

Proof: So assume there is a plan for a problem instance $\langle A, I, O, G, B \rangle$. Label all nodes of the plan as follows. The root node N is labeled with I, that is, $l(N) = I$. When possible parent nodes of a node n are labeled, we can compute the label for n. Let $\langle o_1, n \rangle, \ldots, \langle o_m, n \rangle$ be the annotations of all operator nodes $n_1, \ldots, n_m$ in the plan with n as the child node, and let $\{\langle \phi_1, n \rangle, \ldots\}, \ldots, \{\langle \phi_k, n \rangle, \ldots\}$ the respective annotations of all branch nodes $n'_1, \ldots, n'_k$ with n

as one of the child nodes. Then the label of n is $img_{o_1}(l(n_1)) \cup \cdots \cup img_{o_m}(l(n_m)) \cup (l(n_1') \cap \phi_1) \cup \cdots \cup (l(n_k') \cap \phi_k)$. This labeling simply says what are the possible current states for every node of the plan when the plan is executed starting from some initial state.

We show that – *assuming that the algorithm does not terminate earlier after producing a superset of I* – the algorithm determines that for all node labels a plan for reaching G exists if plans exist for its child nodes.

Induction hypothesis: For each plan node n such that all paths to a terminal node have length i or less, its label $S = l(n)$ is a subset of some $S' \in$ flat(H), where H is the value of the program variable H after the *while* loop exits and H could not be extended further.

Base case $i = 0$: Terminal nodes of the plan are labeled with subsets of G. By Lemma 4.38, $G' \in$ flat(H) for some set G' such that $G \subseteq G'$ because G was in H initially.

Inductive case $i \geq 1$: Let n be a plan node. By the induction hypothesis for all child nodes n' of n, $l(n') \subseteq S$ for some $S \in$ flat(H).

If n is a branch node with child nodes $n_1, \ldots, n_k$ and respective conditions $\phi_1, \ldots, \phi_k$, then $l(n) \cap \phi_1, \ldots, l(n) \cap \phi_k$ all occupy disjoint observational classes and superset of $l(n) \cap \phi_i$ for every $i \in \{1, \ldots, k\}$ is in flat(H). Hence by Lemma 4.39 $l(n) \subseteq S$ for some $S \in$ flat(H).

If n is an operator node with operator o and child node n', then $img_o(l(n)) \subseteq l(n')$, and by the induction hypothesis $l(n') \subseteq S'$ for some $S' \in$ flat(H). We have to show that $l(n) \subseteq S''$ for some $S'' \in$ flat(H). Assume that there is no such S''. But now by Lemma 4.37 findnew$(o,\emptyset,H,H)$ would return S''' such that $spreimg_o(S''') \subseteq S$ for no $S \in$ flat(H), and the *while* loop could not have exited with H, contrary to our assumption about H. $\qquad\square$

Theorem 4.41 *Let $\Pi = \langle A, I, O, G, B \rangle$ be a problem instance. If procedure plan(I,O,G) in Figure 4.13 returns* true*, then Π has a solution plan.*

Proof: Let $H^0, H^1, \ldots$ be the sequence of factored belief spaces H produced by the algorithm. We show that for all $i \geq 0$, for every set of states in H^i there is a plan that reaches G.

Induction hypothesis: H^i contains only such sets $S \in$ flat(H^i) for which a plan reaching G exists.

Base case $i = 0$: Initially $H^0 = \mathcal{F}(G)$ and the only set in H^0 is G. The empty plan reaches G from G.

Inductive case $i \geq 1$: H^{i+1} is obtained as $H^i \oplus \mathcal{F}(spreimg_o(S))$ where $S =$findnew$(o,\emptyset,H^i,H^i)$. By Lemma 4.37 $S \in$ flat(H^i) and $spreimg_o(S) \subseteq S'$ for no $S' \in$ flat(H^i). Because S is in H^i, there is a plan π for reaching G from S. The plan that executes o followed by π reaches G from $spreimg_o(S)$.

Let Z be any member of flat(H^{i+1}). We show that for Z there is a plan for reaching G. The plan for Z starts by a branch[4]. We show that for every possible observation, corresponding to one observational class, there is a plan that reaches G. Let C_j be the jth observational class. When observing C_j, the current state is in $Z_j = Z \cap C_j$. Now for Z_j there is $Z_j' \in H_j^{i+1}$ with $Z_j \subseteq Z_j'$, where H_j^{i+1} is the jth component of H^{i+1}. Now by induction hypothesis there is a plan for Z_j' if $Z_j' \in H_j^i$, and if $Z_j' \in H_j^{i+1} \backslash H_j^i$, then for branch corresponding to C_j we use the plan for $spreimg_o(S)$, as Z_j' must be $spreimg_o(S) \cap C_j$. $\qquad\square$

---

[4]Some of the branches might not be needed, and if the intersection of Z with only one observational class is non-empty the plan could start with an operator node instead of a degenerate branch node.

4.5   Computational complexity

In this section we analyze the computational complexity of the main decision problems related
to nondeterministic planning. The conditional planning problem is a generalization of the deter-
ministic planning problem from Chapter 3, and therefore the plan existence problem is at least
PSPACE-hard. In this section we discuss the computational complexity of each of the three plan-
ning problems, the fully observable, the unobservable, and the general partially observable plan-
ning problem, showing them respectively complete for the complexity classes EXP, EXPSPACE
and 2-EXP.

4.5.1   Planning with full observability

We first show that the plan existence problem for nondeterministic planning with full observability
is EXP-hard and then that the problem is in EXP.

   The EXP-hardness proof in Theorem 4.42 is by simulating polynomial-space alternating Tur-
ing machines by nondeterministic planning problems with full observability and the using the fact
that the complexity classes EXP and APSPACE are the same (see Section 2.4.) The most inter-
esting thing in the proof is the representation of alternation. Theorem 3.42 already showed how
deterministic Turing machines with a polynomial space bound are simulated, and the difference is
that we now have nondeterminism, that is, a configuration of the TM may have several successor
configurations, and that there are both $\forall$ and $\exists$ states.[5]

   The $\forall$ states mean that all successor configurations must be accepting (terminal or non-terminal)
configurations. The $\exists$ states mean that at least one successor configuration must be an accepting
(terminal or non-terminal) configuration. Both of these requirements can be represented in the
nondeterministic planning problem.

   The transitions from a configuration with a $\forall$ state will correspond to one nondeterministic oper-
ator. That all successor configurations must be accepting (terminal or non-terminal) configurations
corresponds to requirement in planning that from all successor states of a state a goal state must
be reached.

   Every transition from a configuration with $\exists$ state will correspond to a deterministic operator,
that is, the transition may be chosen, as only one of the successor configurations needs to be
accepting.

Theorem 4.42 *The problem of testing the existence of an acyclic plan for problem instances with
full observability is EXP-hard.*

Proof: Let $\langle \Sigma, Q, \delta, q_0, g \rangle$ be any alternating Turing machine with a polynomial space bound $p(x)$.
Let σ be an input string of length n.

   We construct a problem instance in nondeterministic planning with full observability for sim-
ulating the Turing machine. The problem instance has a size that is polynomial in the size of the
description of the Turing machine and the input string.

   The set A of state variables in the problem instance consists of

   1. $q \in Q$ for denoting the internal states of the TM,

   2. s_i for every symbol $s \in \Sigma \cup \{|, \square\}$ and tape cell $i \in \{0, \ldots, p(n)\}$, and

---

[5]Restricting the proof of Theorem 4.42 to $\exists$ states with nondeterministic transitions would yield a proof of the
NPSPACE-hardness of deterministic planning, but this is not interesting as PSPACE=NPSPACE.

3. h_i for the positions of the R/W head $i \in \{0, \ldots, p(n) + 1\}$.

The unique initial state of the problem instance represents the initial configuration of the TM. The corresponding formula is the conjunction of the following literals.

1. q_0

2. $\neg q$ for all $q \in Q \backslash \{q_0\}$.

3. s_i for all $s \in \Sigma$ and $i \in \{1, \ldots, n\}$ such that ith input symbol is s.

4. $\neg s_i$ for all $s \in \Sigma$ and $i \in \{1, \ldots, n\}$ such that ith input symbol is not s.

5. $\neg s_i$ for all $s \in \Sigma$ and $i \in \{0, n + 1, n + 2, \ldots, p(n)\}$.

6. $\square_i$ for all $i \in \{n + 1, \ldots, p(n)\}$.

7. $\neg \square_i$ for all $i \in \{0, \ldots, n\}$.

8. $|_0$

9. $\neg|_i$ for all $n \in \{1, \ldots, p(n)\}$

10. h_1

11. $\neg h_i$ for all $i \in \{0, 2, 3, 4, \ldots, p(n) + 1\}$

The goal is the following formula.

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}$$

Next we define the operators. All the transitions may be nondeterministic, and the important thing is whether the transition is for a $\forall$ state or an $\exists$ state.[6] For a given input symbol and a $\forall$ state, the transition corresponds to one nondeterministic operator, whereas for a given input symbol and an $\exists$ state the transitions corresponds to a set of deterministic operators.

To define the operators, we first define effects corresponding to all possible transitions.

For all $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$, $i \in \{0, \ldots, p(n)\}$ and $\langle s', q', m \rangle \in (\Sigma \cup \{|\}) \times Q \times \{L, N, R\}$ define the effect $\tau_{s,q,i}(s', q', m)$ as $\alpha \wedge \kappa \wedge \theta$ where the effects α, κ and θ are defined as follows.

The effect α describes what happens to the tape symbol under the R/W head. If $s = s'$ then $\alpha = \top$ as nothing on the tape changes. Otherwise, $\alpha = \neg s_i \wedge s'_i$ to denote that the new symbol in the ith tape cell is s' and not s.

The effect κ describes the change to the internal state of the TM. Again, either the state changes or does not, so $\kappa = \neg q \wedge q'$ if $q \neq q'$ and $\top$ otherwise. We define $\kappa = \neg q$ when $i = p(n)$ and $m = R$ so that when the space bound gets violated, no accepting state can be reached.

The effect θ describes the movement of the R/W head. Either there is movement to the left, no movement, or movement to the right. Hence

$$\theta = \begin{cases} \neg h_i \wedge h_{i-1} & \text{if } m = L \\ \top & \text{if } m = N \\ \neg h_i \wedge h_{i+1} & \text{if } m = R \end{cases}$$

---

[6]No operators are needed for accepting or rejecting states.

By definition of TMs, movement at the left end of the tape is always to the right. Similarly, we have state variable for R/W head position $p(n) + 1$ and moving to that position is possible, but no transitions from that position are possible, as the space bound has been violated.

Now, these effects that represent possible transitions are used in the operators that simulate the ATM. Operators for existential states $q, g(q) = \exists$ and for universal states $q, g(q) = \forall$ differ. Let $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q, i \in \{0, \ldots, p(n)\}$ and $\delta(s, q) = \{\langle s_1, q_1, m_1 \rangle, \ldots, \langle s_k, q_k, m_k \rangle\}$.

If $g(q) = \exists$, then define k deterministic operators

$$\begin{aligned} o_{s,q,i,1} &= \langle h_i \wedge s_i \wedge q, \tau_{s,q,i}(s_1, q_1, m_1) \rangle \\ o_{s,q,i,2} &= \langle h_i \wedge s_i \wedge q, \tau_{s,q,i}(s_2, q_2, m_2) \rangle \\ &\vdots \\ o_{s,q,i,k} &= \langle h_i \wedge s_i \wedge q, \tau_{s,q,i}(s_k, q_k, m_k) \rangle \end{aligned}$$

That is, the plan determines which transition is chosen.

If $g(q) = \forall$, then define one nondeterministic operator

$$\begin{aligned} o_{s,q,i} = \langle h_i \wedge s_i \wedge q, \quad (\tau_{s,q,i}(s_1, q_1, m_1)| \\ \tau_{s,q,i}(s_2, q_2, m_2)| \\ \vdots \\ \tau_{s,q,i}(s_k, q_k, m_k)) \rangle. \end{aligned}$$

That is, the transition is chosen nondeterministically.

We claim that the problem instance has a plan if and only if the Turing machine accepts without violating the space bound.

If the Turing machine violates the space bound, the state variable $h_{p(n)+1}$ becomes true and an accepting state cannot be reached because no operator will be applicable.

Otherwise, we show inductively that from a computation tree of an accepting ATM we can extract a conditional plan that always reaches a goal state, and vice versa. For obtaining an correspondence between conditional plans and computation trees it is essential that the plans are acyclic.

kesken

So, because all alternating Turing machines with a polynomial space bound can be in polynomial time translated to a nondeterministic planning problem, all decision problems in APSPACE are polynomial time many-one reducible to nondeterministic planning, and the plan existence problem is APSPACE-hard and consequently EXP-hard. □

We can extend Theorem 4.42 to general plans with loops. The problem looping plans cause in the proofs of this theorem is that a Turing machine computation of infinite length is not accepting but the corresponding infinite length zero-probability plan execution is allowed to be a part of plan and would incorrectly count as an accepting Turing machine computation.

To eliminate infinite plan executions we have to modify the Turing machine simulation. This is by counting the length of the plan executions and failing when at least one state or belief state must have been visited more than once. This modification makes infinite loops ineffective, and any plan containing a loop can be translated to a finite non-looping plan by unfolding the loop. In the absence of loops the simulation of alternating Turing machines is faithful.

Theorem 4.43 *The plan existence problem for problem instances with full observability is EXP-hard.*

Proof: This is an easy extension of the proof of Theorem 4.42. If there are n state variables, an acyclic plan exists if and only if a plan with execution length at most 2^n exists, because visiting any state more than once is unnecessary. Plans that rely on loops can be invalidated by counting the number of actions taken and failing when this exceeds 2^n. This counting can be done by having $n + 1$ auxiliary state variables $c_0, \ldots, c_n$ that are initialized to false. Every operator $\langle p, e \rangle$ is extended to $\langle p, e \wedge t \rangle$ where t is an effect that increments the binary number encoded by $c_0, \ldots, c_n$ by one until the most significant bit c_n becomes one. The goal G is replaced by $G \wedge \neg c_n$.

Then a plan exists if and only if an acyclic plan exists if and only if the alternating Turing machine accepts. $\square$

Theorem 4.44 *The problem of testing the existence of a plan for problem instances with full observability is in EXP.*

Proof: The algorithm in Section 4.3.2 runs in exponential time in the size of the problem instance. $\square$

4.5.2 Planning without observability

The plan existence problem of conditional planning with unobservability is more complex than that of conditional planning with full observability.

To show the unobservable problem EXPSPACE-hard by a direct simulation of exponential space Turing machines, the first problem is how to encode the tape of the TM. With polynomial space, as in the PSPACE-hardness and APSPACE-hardness proofs of deterministic planning and conditional planning with full observability, it was possible to represent all the tape cells as the state variables of the planning problem. With an exponential space bound this is not possible any more, as we would need an exponential number of state variables, and the planning problem could not be constructed in polynomial time.

Hence we have to find a more clever way of encoding the working tape. It turns out that we can use the uncertainty about the initial state for this purpose. When an execution of the plan that simulates the Turing machine is started, we randomly choose one of the tape cells to be the *watched* tape cell. This is the only cell of the tape for which the current symbol is represented in the state variables. On all transitions the plan makes, if the watched tape cell changes, the change is reflected in the state variables.

That the plan corresponds to a simulation of the Turing machine it is tested whether the transition the plan makes when the current tape cell is the watched tape cell is the one that assumes the current symbol to be the one that is stored in the state variables. If it is not, the plan is not a valid plan. Because the watched tape cell could be any of the exponential number of tape cells, all the transitions the plan makes are guaranteed to correspond to the contents of the current tape cell of the Turing machine, so if the plan does not simulate the Turing machine, the plan is not guaranteed to reach the goal states.

The proof requires both several initial states and unobservability. Several initial states are needed for selecting the watched tape cell, and unobservability is needed so that the plan can-

not cheat: if the plan can determine what the current tape cell is, it could choose transitions that do not correspond to the Turing machine on all but the watched tape cell. Because of unobservability all the transitions have to correspond to the Turing machine.

Theorem 4.45 *The problem of testing the existence of a plan for problem instances with unobservability is EXPSPACE-hard.*

Proof: Proof is a special case of the proof of Theorem 4.48. We do not have $\forall$ states and restrict to deterministic Turing machines. Nondeterministic Turing machines could be simulated for a NEXPSPACE-hardness proof, but it is already known that EXPSPACE = NEXPSPACE, so this additional generality would not bring anything.

Let $\langle \Sigma, Q, \delta, q_0, g \rangle$ be any deterministic Turing machine with an exponential space bound $e(x)$. Let σ be an input string of length n. We denote the ith symbol of σ by σ_i.

The Turing machine may use space $e(n)$, and for encoding numbers from 0 to $e(n) + 1$ corresponding to the tape cells we need $m = \lceil \log_2(e(n) + 2) \rceil$ Boolean state variables.

We construct a problem instance in nondeterministic planning without observability for simulating the Turing machine. The problem instance has a size that is polynomial in the size of the description of the Turing machine and the input string.

We cannot have a state variable for every tape cell because the reduction from Turing machines to planning would not be polynomial time. It turns out that it is not necessary to encode the whole contents of the tape in the transition system of the planning problem, and that it suffices to keep track of only one tape cell (which we will call the *watched tape cell*) that is randomly chosen in the beginning of every execution of the plan.

The set A of state variables in the problem instance consists of

1. $q \in Q$ for denoting the internal states of the TM,

2. w_i for $i \in \{0, \ldots, m-1\}$ for the watched tape cell $i \in \{0, \ldots, e(n)\}$,

3. s for every symbol $s \in \Sigma \cup \{|, \square\}$ for the contents of the watched tape cell,

4. h_i for $i \in \{0, \ldots, m-1\}$ for the position of the R/W head $i \in \{0, \ldots, e(n) + 1\}$.

The uncertainty in the initial state is about which tape cell is the watched one. Otherwise the formula encodes the initial configuration of the TM, and it is the conjunction of the following formulae.

1. q_0

2. $\neg q$ for all $q \in Q \backslash \{q_0\}$.

3. Formulae for having the contents of the watched tape cell in state variables $\Sigma \cup \{|, \square\}$.

$$
\begin{array}{rcl}
| & \leftrightarrow & (w = 0) \\
\square & \leftrightarrow & (w > n) \\
s & \leftrightarrow & \bigvee_{i \in \{1, \ldots, n\}, \sigma_i = s} (w = i) \text{ for all } s \in \Sigma
\end{array}
$$

4. $h = 1$ for the initial position of the R/W head.

So the initial state formula allows any values for state variables w_i and the values of the state variables $s \in \Sigma$ are determined on the basis of the values of w_i. The expressions $w = i$, $w > i$ denote the obvious formulae for testing integer equality and inequality of the numbers encoded by $w_0, w_1, \ldots$. Later we will also use effects $h := h + 1$ and $h := h - 1$ that represent incrementing and decrementing the number encoded by $h_0, h_1, \ldots$.

The goal is the following formula.

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}$$

To define the operators, we first define effects corresponding to all possible transitions.

For all $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\langle s', q', m \rangle \in (\Sigma \cup \{|\}) \times Q \times \{L, N, R\}$ define the effect $\tau_{s,q}(s', q', m)$ as $\alpha \wedge \kappa \wedge \theta$ where the effects α, κ and θ are defined as follows.

The effect α describes what happens to the tape symbol under the R/W head. If $s = s'$ then $\alpha = \top$ as nothing on the tape changes. Otherwise, $\alpha = ((h = w) \rhd (\neg s \wedge s'))$ to denote that the new symbol in the watched tape cell is s' and not s.

The effect κ describes the change to the internal state of the TM. Again, either the state changes or does not, so $\kappa = \neg q \wedge q'$ if $q \neq q'$ and $\top$ otherwise. If R/W head movement is to the right we define $\kappa = \neg q \wedge ((h < e(n)) \rhd q')$ if $q \neq q'$ and $(h = e(n)) \rhd \neg q$ otherwise. This prevents reaching an accepting state if the space bound is violated: no further operator applications are possible.

The effect θ describes the movement of the R/W head. Either there is movement to the left, no movement, or movement to the right. Hence

$$\theta \;=\; \begin{cases} h := h - 1 & \text{if } m = L \\ \top & \text{if } m = N \\ h := h + 1 & \text{if } m = R \end{cases}$$

By definition of TMs, movement at the left end of the tape is always to the right.

Now, these effects $\tau_{s,q}(s', q', m)$ which represent possible transitions are used in the operators that simulate the DTM. Let $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\delta(s, q) = \{\langle s', q', m \rangle\}$.

If $g(q) = \exists$, then define the operator

$$o_{s,q} = \langle ((h \neq w) \vee s) \wedge q, \tau_{s,q}(s', q', m) \rangle$$

It is easy to verify that the planning problem simulates the DTM assuming that when operator $o_{s,q}$ is executed the current tape symbol is indeed s. So assume that some $o_{s,q}$ is the first operator that misrepresents the tape contents and that $h = c$ for some tape cell location c. Now there is an execution of the plan so that $w = c$. On this execution the precondition $o_{s,q}$ is not satisfied, and the plan is not executable. Hence a valid plan cannot contain operators that misrepresent the tape contents. $\square$

Theorem 4.46 *The problem of testing the existence of a plan for problem instances with unobservability is in EXPSPACE.*

Proof: Proof is similar to the proof Theorem 3.43 but works at the level of belief states. $\square$

The two theorems together yield the EXPSPACE-completeness of the plan existence problem for conditional planning without observability.

4.5.3 Planning with partial observability

We show that the plan existence problem of the general conditional planning problem with partial observability is 2-EXP-complete. The hardness proof is by a simulation of AEXPSPACE=2-EXP Turing machines. Membership in 2-EXP is obtained directly from the decision procedure discussed earlier: the procedure runs in polynomial time in the size of the enumerated belief space of doubly exponential size.

Showing that the plan existence problem for planning with partial observability is in 2-EXP is straightforward. The easiest way to see this is to view the partially observable planning problem as a nondeterministic fully observable planning problem with belief states viewed as states. An operator maps a belief state to another belief state nondeterministically: compute the image of a belief state with respect to an operator, and choose the subset of its states that correspond to one of the possible observations. Like pointed out in the proof of Theorem 4.44, the algorithms for fully observable planning run in polynomial time in the size of the state space. The state space with the belief states as the states has a doubly exponential size in the size of the problem instance, and hence the algorithm runs in doubly exponential time in the size of the problem instance. This gives us the membership in 2-EXP.

Theorem 4.47 *The plan existence problem for problem instances with partial observability is in 2-EXP.*

The hardness proof is an extension of both the EXP-hardness proof of Theorem 4.42 and of the EXPSPACE-hardness proof of Theorem 4.45. From the first proof we have the simulation of alternating Turing machines, and from the second proof the simulation of Turing machines with an exponentially long tape.

Theorem 4.48 *The problem of testing the existence of an acyclic plan for problem instances with partial observability is 2-EXP-hard.*

Proof: Let $\langle \Sigma, Q, \delta, q_0, g \rangle$ be any alternating Turing machine with an exponential space bound $e(x)$. Let σ be an input string of length n. We denote the ith symbol of σ by σ_i.

The Turing machine may use space $e(n)$, and for encoding numbers from 0 to $e(n) + 1$ corresponding to the tape cells we need $m = \lceil \log_2(e(n) + 2) \rceil$ Boolean state variables.

We construct a problem instance in nondeterministic planning with full observability for simulating the Turing machine. The problem instance has a size that is polynomial in the size of the description of the Turing machine and the input string.

We cannot have a state variable for every tape cell because the reduction from Turing machines to planning would not be polynomial time. It turns out that it is not necessary to encode the whole contents of the tape in the transition system of the planning problem, and that it suffices to keep track of only one tape cell (which we will call the *watched tape cell*) that is randomly chosen in the beginning of every execution of the plan.

The set A of state variables in the problem instance consists of

1. $q \in Q$ for denoting the internal states of the TM,

2. w_i for $i \in \{0, \ldots, m - 1\}$ for the watched tape cell $i \in \{0, \ldots, e(n)\}$,

3. s for every symbol $s \in \Sigma \cup \{|, \square\}$ for the contents of the watched tape cell,

4. s^* for every $s \in \Sigma \cup \{|\}$ for the symbol last written (important for nondeterministic transitions),

5. L, R and N for the last movement of the R/W head (important for nondeterministic transitions), and

6. h_i for $i \in \{0, \ldots, m-1\}$ for the position of the R/W head $i \in \{0, \ldots, e(n)+1\}$.

The observable state variables are L, N and R, $q \in Q$, and s^* for $s \in \Sigma$. These are needed by the plan to decide how to proceed execution after a nondeterministic transition with a $\forall$ state.

The uncertainty in the initial state is about which tape cell is the watched one. Otherwise the formula encodes the initial configuration of the TM, and it is the conjunction of the following formulae.

1. q_0

2. $\neg q$ for all $q \in Q \backslash \{q_0\}$.

3. $\neg s^*$ for all $s \in \Sigma \cup \{|\}$.

4. Formulae for having the contents of the watched tape cell in state variables $\Sigma \cup \{|, \Box\}$.

$$\begin{aligned} | &\leftrightarrow (w = 0) \\ \Box &\leftrightarrow (w > n) \\ s &\leftrightarrow \bigvee\nolimits_{i \in \{1,\ldots,n\}, \sigma_i = s} (w = i) \text{ for all } s \in \Sigma \end{aligned}$$

5. $h = 1$ for the initial position of the R/W head.

So the initial state formula allows any values for state variables w_i and the values of the state variables $s \in \Sigma$ are determined on the basis of the values of w_i. The expressions $w = i$, $w > i$ denote the obvious formulae for testing integer equality and inequality of the numbers encoded by $w_0, w_1, \ldots$. Later we will also use effects $h := h + 1$ and $h := h - 1$ that represent incrementing and decrementing the number encoded by $h_0, h_1, \ldots$.

The goal is the following formula.

$$G = \bigvee \{q \in Q | g(q) = \text{accept}\}$$

Next we define the operators. All the transitions may be nondeterministic, and the important thing is whether the transition is for a $\forall$ state or an $\exists$ state. For a given input symbol and a $\forall$ state, the transition corresponds to one nondeterministic operator, whereas for a given input symbol and an $\exists$ state the transitions corresponds to a set of deterministic operators.

To define the operators, we first define effects corresponding to all possible transitions.

For all $\langle s, q \rangle \in (\Sigma \cup \{|, \Box\}) \times Q$ and $\langle s', q', m \rangle \in (\Sigma \cup \{|\}) \times Q \times \{L, N, R\}$ define the effect $\tau_{s,q}(s', q', m)$ as $\alpha \wedge \kappa \wedge \theta$ where the effects α, κ and θ are defined as follows.

The effect α describes what happens to the tape symbol under the R/W head. If $s = s'$ then $\alpha = \top$ as nothing on the tape changes. Otherwise, $\alpha = ((h = w) \rhd (\neg s \wedge s')) \wedge s'^* \wedge \neg s^*$ to denote that the new symbol in the watched tape cell is s' and not s, and to make it possible for the plan to detect which symbol was written to the tape by the possibly nondeterministic transition.

The effect κ describes the change to the internal state of the TM. Again, either the state changes or does not, so $\kappa = \neg q \wedge q'$ if $q \neq q'$ and $\top$ otherwise. If R/W head movement is to the right we

define $\kappa = \neg q \wedge ((h < e(n)) \rhd q')$ if $q \neq q'$ and $(h = e(n)) \rhd \neg q$ otherwise. This prevents reaching an accepting state if the space bound is violated: no further operator applications are possible.

The effect θ describes the movement of the R/W head. Either there is movement to the left, no movement, or movement to the right. Hence

$$
\theta \;=\; \left\{
\begin{array}{ll}
(h := h - 1) \wedge L \wedge \neg N \wedge \neg R & \text{if } m = L \\
N \wedge \neg L \wedge \neg R & \text{if } m = N \\
(h := h + 1) \wedge R \wedge \neg L \wedge \neg N & \text{if } m = R
\end{array}
\right.
$$

By definition of TMs, movement at the left end of the tape is always to the right.

Now, these effects $\tau_{s,q}(s', q', m)$ which represent possible transitions are used in the operators that simulate the ATM. Operators for existential states $q, g(q) = \exists$ and for universal states $q, g(q) = \forall$ differ. Let $\langle s, q \rangle \in (\Sigma \cup \{|, \square\}) \times Q$ and $\delta(s, q) = \{\langle s_1, q_1, m_1 \rangle, \ldots, \langle s_k, q_k, m_k \rangle\}$.

If $g(q) = \exists$, then define k deterministic operators

$$
\begin{aligned}
o_{s,q,1} &= \langle ((h \neq w) \vee s) \wedge q, \tau_{s,q}(s_1, q_1, m_1) \rangle \\
o_{s,q,2} &= \langle ((h \neq w) \vee s) \wedge q, \tau_{s,q}(s_2, q_2, m_2) \rangle \\
&\;\vdots \\
o_{s,q,k} &= \langle ((h \neq w) \vee s) \wedge q, \tau_{s,q}(s_k, q_k, m_k) \rangle
\end{aligned}
$$

That is, the plan determines which transition is chosen.

If $g(q) = \forall$, then define one nondeterministic operator

$$
\begin{aligned}
o_{s,q} = \langle ((h \neq w) \vee s) \wedge q, \quad &(\tau_{s,q}(s_1, q_1, m_1)| \\
&\tau_{s,q}(s_2, q_2, m_2)| \\
&\;\vdots \\
&\tau_{s,q}(s_k, q_k, m_k)) \rangle).
\end{aligned}
$$

That is, the transition is chosen nondeterministically.

We claim that the problem instance has a plan if and only if the Turing machine accepts without violating the space bound. If the Turing machine violates the space bound, then $h > e(n)$ and an accepting state cannot be reached because no further operator will be applicable.

From an accepting computation tree of an ATM we can construct a plan, and vice versa. Accepting final configurations are mapped to terminal nodes of plans, $\exists$-configurations are mapped to operator nodes in which an operator corresponding to the transition to an accepting successor configuration is applied, and $\forall$-configurations are mapped to operator nodes corresponding to the matching nondeterministic operators followed by a branch node that selects the plan nodes corresponding to the successors of the $\forall$ configuration. The successors of $\forall$ and $\exists$ configurations are recursively mapped to plans.

Construction of computation trees from plans is similar, but involves small technicalities. A plan with DAG form can be turned into a tree by having several copies of the shared subplans. Branches not directly following the nondeterministic operator causing the uncertainty can be moved earlier so that every nondeterministic operator is directly followed by a branch that chooses a successor node for every possible new state, written symbol and last tape movement. With these transformations there is an exact match between plans and computation trees of the ATM, and mapping from plans to ATMs is straightforward like in the opposite direction.

Because alternating Turing machines with an exponential space bound are polynomial time reducible to the nondeterministic planning problem with partial observability, the plan existence problem is AEXPSPACE=2-EXP-hard. □

What remains to be done is the extension of the above theorem to the case with arbitrary (possibly cyclic) plans. For the fully observable case counting the execution length does not pose a problem because we only have to count an exponential number of execution steps, which can be represented by a polynomial number of state variables, but in the partially observable case we need to count a doubly exponential number of execution steps, as the number of belief states to be visited may be doubly exponential. A binary representation of these numbers requires an exponential number of bits, and we cannot use an exponential number of state variables for the purpose, because the reduction to planning would not be polynomial time. However, partial observability together with only a polynomial number of auxiliary state variables can be used to force the plans to count doubly exponentially far.

Theorem 4.49 *The plan existence problem for problem instances with partial observability is 2-EXP-hard.*

Proof: We extend the proof of Theorem 4.48 by a counting scheme that makes cyclic plans ineffective. We show how counting the execution length can be achieved within a problem instance obtained from the alternating Turing machine and the input string in polynomial time.

Instead of representing the exponential number of bits explicitly as state variables, we use a randomizing technique for forcing the plans to count the number of Turing machine transitions. The technique has resemblance to the idea in simulating exponentially long tapes in the proofs of Theorems 4.45 and 4.42.

For a problem instance with n state variables (representing the Turing machine configurations) executions that visit each belief state at most once may have length 2^{2^n}. Representing numbers from 0 to $2^{2^n} - 1$ requires 2^n binary digits. We introduce $n + 1$ new unobservable state variables $d_0, \ldots, d_n$ for representing the index of one of the digits and v_d for the value of that digit, and new state variables $c_0, \ldots, c_n$ through which the plan indicates changes in the counter of Turing machine transitions. There is a set of operators by means of which the plan sets the values of these variables before every transition of the Turing machine is made.

The idea of the construction is the following. Whenever the counter of TM transitions is incremented, one of the 2^n digits in the counter changes from 0 to 1 and all of the less significant digits change from 1 to 0. The plan is forced to communicate the index of the digit that changes from 0 to 1 by the state variables $c_0, \ldots, c_n$. The unobservable state variables $d_0, \ldots, d_n, v_d$ store the index and value of one of the digits (chosen randomly in the beginning of the plan execution), that we call *the watched digit*, and they are used for checking that the reporting of $c_0, \ldots, c_n$ by the plan is truthful. The test for truthful reporting is randomized, but this suffices to invalidate plans that incorrectly report the increments, as a valid plan has to reach the goals on every possible execution. The plan is invalid if reporting is false or when the count can exceed 2^{2^n}. For this reason a plan for the problem instance exists if and only if an acyclic plan exists if and only if the Turing machine accepts the input string.

Next we exactly define how the problem instances defined in the proof of Theorem 4.48 are extended with a counter to prevent unbounded looping.

The initial state description is extended with the conjunct $\neg d_v$ to signify that the watched digit

is initially 0 (all the digits in the counter implicitly represented in the belief state are 0.) The state variables $d_0, \ldots, d_n$ may have any values which means that the watched digit is chosen randomly. The state variables $d_v, d_0, \ldots, d_n$ are all unobservable so that the plan does not know the watched digit (may not depend on it).

There is also a failure flag f that is initially set to false by having $\neg f$ in the initial states formula.

The goal is extended by $\neg f \wedge ((d_0 \wedge \cdots \wedge d_n) \rightarrow \neg d_v)$ to prevent executions that lead to setting f true or that have length $2^{2^{n+1}-1}$ or more. The conjunct $(d_0 \wedge \cdots \wedge d_n) \rightarrow \neg d_v$ is false if the index of the watched digit is $2^{n+1} - 1$ and the digit is true, indicating an execution of length $\geq 2^{2^{n+1}-1}$.

Then we extend the operators simulating the Turing machine transitions, as well as introduce new operators for indicating which digit changes from 0 to 1.

The operators for indicating the changing digit are

$$\begin{array}{ll} \langle \top, c_i \rangle & \text{for all } i \in \{0, \ldots, n\} \\ \langle \top, \neg c_i \rangle & \text{for all } i \in \{0, \ldots, n\} \end{array}$$

The operators for Turing machine transitions are extended with the randomized test that the digit the plan claims to change from 0 to 1 is indeed the one: every operator $\langle p, e \rangle$ defined in the proof of Theorem 4.48 is replaced by $\langle p, e \wedge t \rangle$ where the test t is the conjunction of the following effects.

$$\begin{array}{rcl} ((c = d) \wedge d_v) & \rhd & f \\ (c = d) & \rhd & d_v \\ ((c > d) \wedge \neg d_v) & \rhd & f \\ (c > d) & \rhd & \neg d_v \end{array}$$

Here $c = d$ denotes $(c_0 \leftrightarrow d_0) \wedge \cdots \wedge (c_n \leftrightarrow d_n)$ and $c > d$ encodes the greater-than test for the binary numbers encoded by $c_0, \ldots, c_n$ and $d_0, \ldots, d_n$.

The above effects do the following.

1. When the plan claims that the watched digit changes from 0 to 1 and the value of d_v is 1, fail.

2. When the plan claims that the watched digit changes from 0 to 1, change d_v to 1.

3. When the plan claims that a more significant digit changes from 0 to 1 and the value of d_v is 0, fail.

4. When the plan claims that a more significant digit changes from 0 to 1, set the value of d_v to 0.

That these effects guarantee the invalidity of a plan that relies on unbounded looping is because the failure flag f will be set if the plan lies about the count, or the most significant bit with index $2^{n+1} - 1$ will be set if the count reaches $2^{2^{n+1}-1}$. Attempts of unfair counting are recognized and consequently f is set to true because of the following.

Assume that the binary digit at index i changes from 0 to 1 (and therefore all less significant digits change from 1 to 0) and the plan incorrectly claims that it is the digit j that changes, and this is the first time on that execution that the plan lies (hence the value of d_v is the true value of the watched digit.)

If $j > i$, then i could be the watched digit (and hence $c > d$), and for j to change from 0 to 1 the less significant bit i should be 1, but we would know that it is not because d_v is false. Consequently on this plan execution the failure flag f would be set.

If $j < i$, then j could be the watched digit (and hence $c = d$), and the value of d_v would indicate that the current value of digit j is 1, not 0. Consequently on this plan execution the failure flag f would be set.

So, if the plan does not correctly report the digit that changes from 0 to 1, then the plan is not valid. Hence any valid plan correctly counts the execution length which cannot exceed $2^{2^{n+1}-1}$. $\square$

4.5.4 Polynomial size plans

We showed in Section 3.8 that the plan existence problem of deterministic planning is only NP-complete, in contrast to PSPACE-complete, when a restriction to plans of polynomial length is made. Here we investigate the same question for conditional plans.

Theorem 4.50 *The plan existence problem for conditional planning without observability restricted to polynomial length plans is in Σ_2^p.*

Proof: Let $p(n)$ be any polynomial. We give an $\mathrm{NP}^{\mathrm{NP}}$ algorithm (Turing machine) that solves the problem. Let the problem instance $\langle A, I, O, G, \emptyset \rangle$ have size n.

First guess a sequence of operators $\sigma = o_0, o_1, \ldots, o_k$ for $k < p(n)$. This is nondeterministic polynomial time computation.

Then use an NP-oracle for testing that σ is a solution. The oracle is a nondeterministic polynomial-time Turing machine that accepts if a plan execution does not lead to a goal state or if the plan is not executable (operator precondition not satisfied). The oracle guesses an initial state and for each nondeterministic operator for each step which nondeterministic choices are made, and then in polynomial time tests whether the execution of the operator sequence leads to a goal state.

1. Guess valuation I' that satisfies I.

2. Guess the results of the nondeterministic choices for every operator in the plan: replace every $p_1 e_1 | \cdots | p_n e_n$ by a nondeterministically selected e_i.

3. Compute $s_j = app_{o_j}(app_{o_{j-1}}(\cdots app_{o_2}(app_{o_1}(I'))))$ for $j = 0, j = 1, j = 2, \ldots, j = k$.

4. If $s_j \not\models c_j$ for $o_j = \langle c_j, e_j \rangle$, accept.

5. If $s_k \not\models G$, accept.

6. Otherwise reject.

$\square$

Theorem 4.51 *The plan existence problem for conditional planning without observability restricted to polynomial length plans is Σ_2^p-hard.*

Proof: Truth of QBF of the form $\exists x_1 \cdots x_n \forall y_1 \cdots y_m \phi$ is Σ_2^p-complete. We reduce this problem to the plan existence problem of unobservable planning with polynomial length plans.

- $A = \{x_1, \ldots, x_n, y_1, \ldots, y_m, s, g\}$

	deterministic context-independent	deterministic context-dependent	non-deterministic context-dependent
full observability	PSPACE	PSPACE	EXPTIME
no observability	PSPACE	EXPSPACE	EXPSPACE
partial observability	PSPACE	EXPSPACE	2-EXPTIME

Table 4.2: Computational complexity of plan existence problems

	deterministic context-independent	deterministic context-dependent	non-deterministic context-dependent
full observability	PSPACE	PSPACE	EXPTIME
no observability	PSPACE	PSPACE	EXPSPACE
partial observability	PSPACE	PSPACE	2-EXPTIME

Table 4.3: Computational complexity of plan existence problems with one initial state

- $I = \neg x_1 \wedge \cdots \wedge \neg x_n \wedge \neg g \wedge s$

- $O = \{\langle s, x_1 \rangle, \langle s, x_2 \rangle, \ldots, \langle s, x_n \rangle, \langle s, \neg s \wedge (\phi \rhd g) \rangle\}$

- $G = g$

Out claim is that there is a plan if and only if $\exists x_1 \cdots x_n \forall y_1 \cdots y_m \phi$ is true.

Assume the QBF is true, that is, there is a valuation x for $x_1, \ldots, x_n$ so that $x, y \models \phi$ for any valuation y of $y_1, \ldots, y_m$. Let $X = \{\langle s, x_i \rangle | i \in \{1, \ldots, n\}, x(x_i) = 1\}$. Now the operators X in any order followed by $\langle s, \neg s \wedge (\phi \rhd g) \rangle$ is a plan: whatever values $y_1, \ldots, y_m$ have, ϕ is true after executing the operators X, and hence the last operator makes $G = g$ true.

Assume there is a plan. The plan has one occurrence of $\langle s, \neg s \wedge (\phi \rhd g) \rangle$ and it must be the last operator. Define the valuation x of $x_1, \ldots, x_n$ as follows. Let $x(x_i) = 1$ iff $\langle s, x_i \rangle$ is one of the operators in the plan, for all $i \in \{1, \ldots, n\}$. Because g is reached, $x, y \models \phi$ for any valuation y of $y_1, \ldots, y_m$, and the QBF is therefore true. $\qquad \square$

4.5.5 Summary of the results

The complexities of the plan existence problem under different restrictions on operators and observability are summarized in Tables 4.2 (with an arbitrary number of initial states) and 4.3 (with one initial state). The different columns list the complexities with different restrictions on the operators. In the previous sections we have considered the general problems with arbitrary operators containing conditional effects and nondeterministic choice. These results are summarized in the third column. The second column lists the complexities in the case without nondeterminism (choice |), and the first column without nondeterminism (choice |) and without conditional effects ($\rhd$). These results are not given in this lecture.

4.6 Literature

There is a difficult trade-off between the two extreme approaches, producing a conditional plan covering all situations that might be encountered, and planning only one action ahead. Schoppers

[1987] proposed *universal plans* as a solution to the high complexity of planning. Ginsberg [1989] attacked Schopper's idea. Schopper's proposal was to have memoryless plans that map any given observations to an action. He argued that plans have to be memoryless in order to be able to react to all the unforeseeable situations that might be encountered during plan execution. Ginsberg argued that plans that are able to react to all possible situations are necessarily much too big to be practical. It seems to us that Schopper's insistence on using plans without a memory is not realistic nor necessary, and that most of Ginsberg argumentation on impracticality of universal plans relies on the lack of any memory in the plan execution mechanism. Of course, we agree that a conditional plan that can be executed efficiently can be much bigger than a plan or a planner that has no restrictions on the amount of time consumed in deciding about the action to be taken. Plans without such restrictions could have as high expressivity as Turing machines, for example, and then a conditional plan does not have to be less succinct than the description of a general purpose planning algorithm.

There is some early work on conditional planning that mostly restricts to the fully observable case and is based on partial-order planning [Etzioni *et al.*, 1992; Peot and Smith, 1992; Pryor and Collins, 1996]. We have not discussed these algorithms because they have only been shown to solve very small problem instances.

A variants of the algorithm for constructing plans for nondeterministic planning with full observability in Section 4.3.1 was first presented by Cimatti et al. [2003]. The algorithms by Cimatti et al. construct mappings of states to actions whereas our presentation in Section 4.3 focuses on the computation of distances of states, and plans are synthesized afterwards on the basis of the distances. We believe that our algorithms are conceptually simpler. Cimatti et al. also presented an algorithm for finding *weak plans* that may reach the goals but are not guaranteed to. However, finding weak plans is polynomially equivalent to the deterministic planning problem of Chapter 3 by an easy reduction that replaces each nondeterministic operator by a set of deterministic operators.

The nondeterministic planning problem with unobservability is not very interesting because all robots and intelligent beings can sense their environment in at least some extent. However, there are problems (outside AI) that are equivalent to the unobservable planning problem. Finding homing/reset/synchronization sequences of circuits/automata is an example of such a problem [Pixley *et al.*, 1992]. There are extensions of the distance and cardinality based heuristics for planning without observability not discussed in this lecture [Rintanen, 2004].

Bertoli et al. have presented a forward search algorithm for finding conditional plans in the general partially observable case [Bertoli *et al.*, 2001].

The computational complexity of conditional planning was first investigated by Littman [1997] and Haslum and Jonsson [2000]. They presented proofs for the EXPTIME-completeness of planning with full observability and the EXPSPACE-completeness of planning without observability. The hardness parts of the proofs were reductions respectively from the existence problem of winning strategies for the game G_4 [Stockmeyer and Chandra, 1979] and from the universality problem of regular expressions with exponentiation [Hopcroft and Ullman, 1979]. In this chapter we gave more direct hardness proofs by direct simulation of alternating polynomial space (exponential time) and exponential space Turing machines.