Chapter 4

Nondeterministic planning

4.1 Nondeterministic operators

In this section we will present a basic translation of nondeterministic operators into the propositional logic and a regression operation for nondeterministic operators. In the next sections we will discuss a general framework for computing with nondeterministic operators and their transition relations which are represented as propositional formulae. This framework provides techniques for computing both regression and progression for sets of states that are represented as formulae.

4.1.1 Regression for nondeterministic operators

Regression for deterministic operators is given in Definition 3.5. It can be easily generalized to a subclass of nondeterministic operators.

Definition 4.1 (Regression for nondeterministic operators) *Let ϕ be a propositional formula and $o = \langle c, e_1 | \cdots | e_n \rangle$ an operator where $e_1, \ldots, e_n$ are deterministic. Define*

$$\mathit{regr}_o^{nd}(\phi) = \mathit{regr}_{\langle c,e_1\rangle}(\phi) \wedge \cdots \wedge \mathit{regr}_{\langle c,e_n\rangle}(\phi).$$

Theorem 4.2 *Let ϕ be a formula over A, o an operator over A, and S the set of all states over A. Then $\{s \in S | s \models \mathit{regr}_o^{nd}(\phi)\} = \mathit{spreimg}_o(\{s \in S | s \models \phi\})$.*

Proof: Let $o = \langle c, (e_1 | \cdots | e_n) \rangle$.

$\{s \in S | s \models \mathit{regr}_o^{nd}(\phi)\}$
$= \{s \in S | s \models \mathit{regr}_{\langle c,e_1\rangle}(\phi) \wedge \cdots \wedge \mathit{regr}_{\langle c,e_n\rangle}(\phi)\}$
$= \{s \in S | s \models \mathit{regr}_{\langle c,e_1\rangle}(\phi), \ldots, s \models \mathit{regr}_{\langle c,e_n\rangle}(\phi)\}$
$= \{s \in S | \mathit{app}_{\langle c,e_1\rangle}(s) \models \phi, \ldots, \mathit{app}_{\langle c,e_n\rangle}(s) \models \phi\}$ T3.7
$= \{s \in S | s' \models \phi$ for all $s' \in \mathit{img}_o(s)$ and there is $s' \models \phi$ with $sos'\}$
$= \mathit{spreimg}_o(\{s \in S | s \models \phi\})$

The second last equality is because $\mathit{img}_o(s) = \{\mathit{app}_{\langle c,e_1\rangle}(s), \ldots, \mathit{app}_{\langle c,e_n\rangle}(s)\}$. □

Example 4.3 Let $o = \langle d, (b | \neg c) \rangle$. Then

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

■

4.1.2 Translation of nondeterministic operators into propositional logic

In Section 3.6.2 we gave a translation of deterministic operators into the propositional logic. In this section we extend this translation to nondeterministic operators.

We define for effects e the sets *changes*(e) of state variables that are possibly changed by e, or in other words, the set of state variables occurring in an atomic effect in e.

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

We make the following assumption to simplify the translation.

Assumption 4.4 *Let $a \in A$ be a state variable. Let $e_1 \wedge \cdots \wedge e_n$ occur in the effect of an operator. If $e_1, \ldots, e_n$ are not all deterministic, then a or $\neg a$ may occur as an atomic effect in at most one of $e_1, \ldots, e_n$.*

This assumption rules out effects like $(a|b) \wedge (\neg a|c)$ that may make a simultaneously true and false. It also rules out effects like $((d \rhd a)|b) \wedge ((\neg d \rhd \neg a)|c)$ that are well-defined and could be translated into the propositional logic. However, the additional complexity outweighs the benefit of allowing them. Effects can often easily be transformed by the equivalences in Table 2.3 to satisfy Assumption 4.4: $((d \rhd a)|b) \wedge ((\neg d \rhd \neg a)|c)$ is equivalent to $((d \rhd a) \wedge (\neg d \rhd \neg a))|((d \rhd a) \wedge c)|(b \wedge (\neg d \rhd \neg a))|(b \wedge c)$.

The problem in the translation that does not show up with deterministic operators is that for nondeterministic choices $e_1|\cdots|e_n$ the formula for each e_i has to express the changes for exactly the same set of state variables. This set B is given as a parameter to the translation function. The set B has to include all state variables possibly changed by the effect.

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

The first part of the translation $\tau_B^{nd}(e)$ for deterministic e is the translation of deterministic effects we presented in Section 3.6.2 restricted to state variables in B. The other two parts cover all nondeterministic effects in normal form. In the translation of $e_1 \wedge \cdots \wedge e_n$ all state variables that are not changed are handled in the translation of e_1. Assumption 4.4 guarantees that for each $\tau_B^{nd}(e)$ all state variables changed by e are in B.

Example 4.5 We translate the effect

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

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

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

■

For expressing a state in terms of A' instead of A, or vice versa, we need to map a valuation of A to a corresponding valuation of A', or vice versa. for this purpose we define $s[A'/A] = \{\langle a', s(a)\rangle | a \in A\}$.

Definition 4.6 *Let A be a set of state variables. Let $o = \langle c, e\rangle$ be an operator over A in normal form. Define $\tau^{nd}_A(o) = c \wedge \tau^{nd}_A(e)$.*

Lemma 4.7 *Let o be an operator over a set A of state variables. Then*

$$\{v | v \textit{ is a valuation of } A \cup A', v \models \tau^{nd}_A(o)\} = \{s \cup s'[A'/A] | s, s' \in S, s' \in \textit{img}_o(s)\}.$$

Proof: We show that there is a one-to-one match between valuations satisfying $\tau^{nd}_A(o)$ and pairs of states and their successor states.

For the proof from right to left assume that s and s' are states such that $s' \in \textit{img}_o(s)$. Hence there is $E \in [e]_s$ such that s' is obtained from s by making literals in E true. Let $v = s \cup s'[A'/A]$. We show that $v \models \tau^{nd}_A(o)$. Let $o = \langle c, e\rangle$. Since $\textit{img}_o(s)$ is non-empty, $s \models c$. It remains to show that $v \models \tau^{nd}_A(e)$.

Induction hypothesis: Let e be any effect over a set B of state variables, and s and s' states such for some $E \in [e]_s$ $s' \models E$ and $s(a) = s'(a)$ for every $a \in B$ such that $\{a, \neg a\} \cap E = \emptyset$. Then $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

Base case: e is a deterministic effect. There is only one $E \in [e]_s$. A proof similar to that of Lemma 3.42 shows that $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

Inductive case 1, $e = e_1 \wedge \cdots \wedge e_n$: By definition $\tau^{nd}_B(e_1 \wedge \cdots \wedge e_n) = \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_1) \wedge \tau^{nd}_{B_2}(e_2) \wedge \cdots \wedge \tau^{nd}_{B_n}(e_n)$ for $B_i = \textit{changes}(e_i), i \in \{2, \ldots, n\}$. Let E be any member of $[e]_s$ and s' a state such that $s' \models E$ and $s(a) = s'(a)$ for every $a \in B$ such that $\{a, \neg a\} \cap E = \emptyset$. By definition of $[e]_s$ we have $E = E_1 \cup \cdots \cup E_n$ for some $E_i \in [e_i]_s$ for every $i \in \{1, \ldots, n\}$. The assumptions of the induction hypothesis hold for every e_i and $B_i, i \in \{2, \ldots, n\}$:

1. $s' \models E_i$ because $E_i \subseteq E$.
2. By Assumption 4.4 $s(a) = s'(a)$ for every $a \in B_i$ such that $\{a, \neg a\} \cap E_i = \emptyset$.

Similarly for e_1 and $B\setminus(B_2 \cup \cdots \cup B_n)$. Hence $s \cup s'[A'/A] \models \tau^{nd}_{B_i}(e_i)$ for all $i \in \{2, \ldots, n\}$ and $s \cup s'[A'/A] \models \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_i)$, and therefore $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

Inductive case 2, $e = e_1 | \cdots | e_n$: By definition $\tau^{nd}_B(e_1 | \cdots | e_n) = \tau^{nd}_B(e_1) \vee \cdots \vee \tau^{nd}_B(e_n)$. By definition $[e_1 | \cdots | e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$. Hence $E \in [e_i]_s$ for some $i \in \{1, \ldots, n\}$. Hence the assumptions of the induction hypothesis hold for at least one $e_i, i \in \{1, \ldots, n\}$ and we get $s \cup s'[A'/A] \models \tau^{nd}_B(e_i)$. As $\tau^{nd}_B(e_i)$ is one of the disjuncts of $\tau^{nd}_B(e)$ finally $s \cup s'[A'/A] \models \tau^{nd}_B(e)$.

For the proof from left to right assume that $v \models \tau^{nd}_B(e)$ for $v = s \cup s'[A'/A]$. We prove by structural induction that the changes from s to s' correspond to $[e]_s$.

Induction hypothesis: Let e be any effect, B a set of state variables that includes those occurring in e, and s and s' states such that $v \models \tau^{nd}_B(e)$ where $v = s \cup s'[A'/A]$. Then there is $E \in [e]_s$ such that $s \models E$ and $s(a) = s'(a)$ for all $a \in B$ such that $\{a, \neg a\} \cap E = \emptyset$.

Base case: e is a deterministic effect. There is only one $E \in [e]_s$. A proof similar to that of Lemma 3.42 shows that the changes between s and s' for $a \in B$ correspond to E.

Inductive case 1, $e = e_1 \wedge \cdots \wedge e_n$: By definition $[e]_s = \{E_1 \cup \cdots \cup E_n | E_1 \in [e_1]_s, \ldots, E_n \in [e_n]_s\}$, and by Assumption 4.4 sets of the state variables occurring in $e_1, \ldots, e_n$ are disjoint. By definition $\tau^{nd}_B(e_1 \wedge \cdots \wedge e_n) = \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_1) \wedge \tau^{nd}_{B_2}(e_2) \wedge \cdots \wedge \tau^{nd}_{B_n}(e_n)$ for $B_i = \mathit{changes}(e_i), i \in \{2, \ldots, n\}$. The induction hypothesis for e and all $a \in B$ is directly by the induction hypothesis for all $a \in B = (B\setminus(B_2 \cup \cdots \cup B_n)) \cup B_2 \cup \cdots \cup B_n$ because $v \models \tau^{nd}_{B\setminus(B_2\cup\cdots\cup B_n)}(e_1) \wedge \tau^{nd}_{B_2}(e_2) \wedge \cdots \wedge \tau^{nd}_{B_n}(e_n)$.

Inductive case 2, $e = e_1|\cdots|e_n$: By definition $[e_1|\cdots|e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$. By definition $\tau^{nd}_B(e_1|\cdots|e_n) = \tau^{nd}_B(e_1) \vee \cdots \vee \tau^{nd}_B(e_n)$. Because $v \models \tau^{nd}_B(e_1|\cdots|e_n)$, $v \models \tau^{nd}_B(e_i)$ for some $i \in \{1, \ldots, n\}$. By the induction hypothesis there is $E \in [e_i]_s$ with the given property. We get the induction hypothesis for e because $[e_i]_s \subseteq [e]_s$ and hence also $E \in [e]_s$.

Therefore s' is obtained from s by making some literals in $E \in [e]_s$ true and retaining the values of state variables not mentioned in E, and $s' \in \mathit{img}_o(s)$. □

4.2 Computing with transition relations as formulae

As discussed in Section 2.3, formulae are a representation of sets of states. In this section we show how operations on transition relations have a counterpart as operations on formulae that represent transition relations.

Most implementations of the techniques in this section are based on binary decision diagrams (BDDs) [Bryant, 1992], a representation (essentially a normal form) of propositional formulae with useful computational properties, but the techniques are applicable to other representations of propositional formulae as well.

4.2.1 Existential and universal abstraction

The most important operations performed on transition relations represented as propositional formulae are based on *existential abstraction* and *universal abstraction*.

Definition 4.8 Existential abstraction *of a formula ϕ with respect to an atomic proposition a is the formula*

$$\exists a.\phi = \phi[\top/a] \vee \phi[\bot/a].$$

Universal abstraction is defined analogously by using conjunction instead of disjunction.

Definition 4.9 Universal abstraction *of a formula ϕ with respect to an atomic proposition a is the formula*

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

Existential and universal abstraction of ϕ with respect to a *set of atomic propositions* is defined in the obvious way: for $B = \{b_1, \ldots, b_n\}$ such that B is a subset of the propositional variables

occurring in ϕ define

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

In the resulting formulae there are no occurrences of variables in B.

Let ϕ be a formula over A. Then $\exists A.\phi$ is a formula that consists of the constants $\top$ and $\bot$ and the logical connectives only. The truth-value of this formula is independent of the valuation of A, that is, its value is the same for all valuations.

The following lemma expresses the important properties of existential and universal abstraction. When we write $v \cup v'$ for a pair of valuations we view valuations v as binary relations, that is, sets of pairs such that $\{(a,b),(a,c)\} \notin v$ for any a, b and c such that $b \neq c$.

Lemma 4.10 *Let ϕ be a formula over $A \cup A'$ and v' a valuation of A'. Then*

1. *$v' \models \exists A.\phi$ if and only if $(v \cup v') \models \phi$ for at least one valuation v of A, and*
2. *$v' \models \forall A.\phi$ if and only if $(v \cup v') \models \phi$ for all valuations v of A.*

Proof: We prove the statements by induction on the cardinality of A. We only give the proof for $\exists$. The proof for $\forall$ is analogous to that for $\exists$.

Base case $|A| = 0$: There is only one valuation $v = \emptyset$ of the empty set $A = \emptyset$. When there is nothing to abstract we have $\exists\emptyset.\phi = \phi$. Hence trivially $v' \models \exists\emptyset.\phi$ if and only if $(v \cup \emptyset) \models \phi$.

Inductive case $|A| \geq 1$: Take any $a \in A$. $v' \models \exists A.\phi$ if and only if $v' \models \exists A\backslash\{a\}.(\phi[\top/a] \vee \phi[\bot/a])$ by the definition of $\exists a.\phi$. By the induction hypothesis $v' \models \exists A\backslash\{a\}.(\phi[\top/a] \vee \phi[\bot/a])$ if and only if $(v_0 \cup v') \models \phi[\top/a] \vee \phi[\bot/a]$ for at least one valuation v_0 of $A\backslash\{a\}$. Since the formula $\phi[\top/a] \vee \phi[\bot/a]$ represents both possible valuations of a in ϕ, the last statement is equivalent to $(v \cup v') \models \phi$ for at least one valuation v of A. □

4.2.2 Images and preimages as formula manipulation

Let $A = \{a_1, \ldots, a_n\}$, $A' = \{a'_1, \ldots, a'_n\}$ and $A'' = \{a''_1, \ldots, a''_n\}$. Let ϕ_1 be a formula over $A \cup A'$ and ϕ_2 be a formula over $A' \cup A''$. The formulae can be viewed as representations of $2^n \times 2^n$ matrices or as transition relations over a state space of size 2^n.

The product matrix of ϕ_1 and ϕ_2 is represented by a the following formula over $A \cup A''$.

$$\exists A'.\phi_1 \wedge \phi_2$$

Example 4.11 Let $\phi_1 = a \leftrightarrow \neg a'$ and $\phi_2 = a' \leftrightarrow a''$ represent two actions, reversing the truth-value of a and doing nothing. The sequential composition of these actions is

$$\begin{aligned}\exists a'.\phi_1 \wedge \phi_2 &= ((a \leftrightarrow \neg\top) \wedge (\top \leftrightarrow a'')) \vee ((a \leftrightarrow \neg\bot) \wedge (\bot \leftrightarrow a'')) \\ &\equiv ((a \leftrightarrow \bot) \wedge (\top \leftrightarrow a'')) \vee ((a \leftrightarrow \top) \wedge (\bot \leftrightarrow a'')) \\ &\equiv a \leftrightarrow \neg a''.\end{aligned}$$

■

This idea can be used for computing the images, preimages and strong preimages of operators and sets of states in terms of formula manipulation by existential and universal abstraction. Table

matrices	formulas	sets of states
vector $V_{1\times n}$	formula over A	set of states
matrix $M_{n\times n}$	formula over $A \cup A'$	transition relation
$V_{1\times n} + V'_{1\times n}$	$\phi_1 \vee \phi_2$	set union
	$\phi_1 \wedge \phi_2$	set intersection
$M_{n\times n} \times N_{n\times n}$	$\exists A'.(\tau_A^{nd}(o) \wedge \tau_A^{nd}(o')[A''/A', A'/A])[A'/A'']$	sequential composition $o \circ o'$
$V_{1\times n} \times M_{n\times n}$	$(\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A']$	$\mathit{img}_o(T)$
$M_{n\times n} \times V_{n\times 1}$	$\exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A])$	$\mathit{preimg}_o(T)$
	$\forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o)$	$\mathit{spreimg}_o(T)$

Table 4.1: Correspondence between matrix operations, Boolean operations and set-theoretic/relational operations. Above $T = \{s \in S | s \models \phi\}$, M is the matrix corresponding to $\tau_A^{nd}(o)$ and N is the matrix corresponding to o'.

4.1 outlines a number of connections between operations on vectors and matrices, on propositional formulae, and on sets and relations. For transition relations we use valuations of $A \cup A'$ for representing pairs for states and for states we use valuations of A.

Lemma 4.12 *Let ϕ be a formula over A and v a valuation of A. Then $v \models \phi$ if and only if $v[A'/A] \models \phi[A'/A]$, and $(\phi[A'/A])[A/A'] = \phi$.*

Definition 4.13 *Let o be an operator and ϕ a formula. Define*

$$\begin{aligned} \mathit{img}_o(\phi) &= (\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A'] \\ \mathit{preimg}_o(\phi) &= \exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A]) \\ \mathit{spreimg}_o(\phi) &= \forall A'.(\tau_A^{nd}(o) \rightarrow \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o). \end{aligned}$$

Theorem 4.14 *Let $T = \{s \in S | s \models \phi\}$. Then $\{s \in S | s \models \mathit{img}_o(\phi)\} = \{s \in S | s \models (\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A']\} = \mathit{img}_o(T)$.*

Proof: $s' \models (\exists A.(\phi \wedge \tau_A^{nd}(o)))[A/A']$ □
iff $s'[A'/A] \models \exists A.(\phi \wedge \tau_A^{nd}(o))$ L4.12
iff there is valuation s of A such that $(s \cup s'[A'/A]) \models \phi \wedge \tau_A^{nd}(o)$ L4.10
iff there is valuation s of A such that $s \models \phi$ and $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$
iff there is $s \in T$ such that $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$
iff there is $s \in T$ such that $s' \in \mathit{img}_o(s)$ L4.7
iff $s' \in \mathit{img}_o(T)$.

Theorem 4.15 *Let $T = \{s \in S | s \models \phi\}$. Then $\{s \in S | s \models \mathit{preimg}_o(\phi)\} = \{s \in S | s \models \exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A])\} = \mathit{preimg}_o(T)$.*

Proof: $s \models \exists A'.(\tau_A^{nd}(o) \wedge \phi[A'/A])$
iff there is $s'_0 : A' \to \{0,1\}$ such that $(s \cup s'_0) \models \tau_A^{nd}(o) \wedge \phi[A'/A]$
iff there is $s'_0 : A' \to \{0,1\}$ such that $s'_0 \models \phi[A'/A]$ and $(s \cup s'_0) \models \tau_A^{nd}(o)$ L4.10
iff there is $s' : A \to \{0,1\}$ such that $s' \models \phi$ and $(s \cup s'_0) \models \tau_A^{nd}(o)$ L4.12
iff there is $s' \in T$ such that $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$
iff there is $s' \in T$ such that $s' \in \mathit{img}_o(s)$ L4.7
iff there is $s' \in T$ such that $s \in \mathit{preimg}_o(s')$ (5) of L2.2
iff $s \in \mathit{preimg}_o(T)$.
Above we define $s' = s'_0[A/A']$ (and hence $s'_0 = s'[A'/A]$.) □

Theorem 4.16 *Let* $T = \{s \in S | s \models \phi\}$. *Then* $\{s \in S | s \models \mathit{spreimg}_o(\phi)\} = \{s \in S | s \models \forall A'.(\tau_A^{nd}(o) \to \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o)\} = \mathit{spreimg}_o(T)$.

Proof:
$s \models \forall A'.(\tau_A^{nd}(o) \to \phi[A'/A]) \wedge \exists A'.\tau_A^{nd}(o)$
iff $s \models \forall A'.(\tau_A^{nd}(o) \to \phi[A'/A])$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $(s \cup s'_0) \models \tau_A^{nd}(o) \to \phi[A'/A]$ for all $s'_0 : A' \to \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L4.10
iff $(s \cup s'_0) \not\models \tau_A^{nd}(o)$ or $s'_0 \models \phi[A'/A]$ for all $s'_0 : A' \to \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $(s \cup s'[A'/A]) \not\models \tau_A^{nd}(o)$ or $s' \models \phi$ for all $s' : A \to \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L4.12
iff $s' \notin \mathit{img}_o(s)$ or $s' \models \phi$ for all $s' : A \to \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$ L4.7
iff $s' \in \mathit{img}_o(s)$ implies $s' \models \phi$ for all $s' : A \to \{0,1\}$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $\mathit{img}_o(s) \subseteq T$ and $s \models \exists A'.\tau_A^{nd}(o)$
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' : A \to \{0,1\}$ with $(s \cup s'[A'/A]) \models \tau_A^{nd}(o)$ L4.10
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' : A \to \{0,1\}$ with $s' \in \mathit{img}_o(s)$ L4.7
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' \in T$ with $s' \in \mathit{img}_o(s)$
iff $\mathit{img}_o(s) \subseteq T$ and there is $s' \in T$ with sos'
iff $s \in \mathit{spreimg}_o(T)$.
Above we define $s' = s'_0[A/A']$ (and hence $s'_0 = s'[A'/A]$.) □

Corollary 4.17 *Let* $o = \langle c, (e_1 | \cdots | e_n) \rangle$ *be an operator such that all* e_i *are deterministic. The formula* $\mathit{spreimg}_o(\phi)$ *is logically equivalent to* $\mathit{regr}_o^{nd}(\phi)$ *as given in Definition 4.1.*

Proof: By Theorems 4.2 and 4.16 $\{s \in S | s \models \mathit{regr}_o(\phi)\} = \mathit{spreimg}_o(\{s \in S | s \models \phi\}) = \{s \in S | s \models \mathit{spreimg}_o(\phi)\}$. □

Example 4.18 Let $o = \langle c, a \wedge (a \rhd b) \rangle$. Then

$$\mathit{regr}_o^{nd}(a \wedge b) = c \wedge (\top \wedge (b \vee a)) \equiv c \wedge (b \vee a).$$

The transition relation of o is represented by

$$\tau_A^{nd}(o) = c \wedge a' \wedge ((b \vee a) \leftrightarrow b') \wedge (c \leftrightarrow c').$$

The preimage of $a \wedge b$ with respect to o is represented by

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

■

Hence regression for nondeterministic operators (Definition 4.1) can be viewed as a specialized method for computing preimages of sets of states represented as formulae.

Many algorithms include the computation of the union of images or preimages with respect to all operators, for example $\bigcup_{o \in O} \mathit{img}_o(T)$, or in terms of formulae, $\bigvee_{o \in O} \mathit{img}_o(\phi)$ where $T = \{s \in S | s \models \phi\}$. A technique used by many implementations of such algorithms is the following. Instead of computing the images or preimages one operator at a time, construct a combined transition relation for all operators. For an illustration of the technique, consider $\mathit{img}_{o_1}(\phi) \vee \mathit{img}_{o_2}(\phi)$ that represents the union of state sets represented by $\mathit{img}_{o_1}(\phi)$ and $\mathit{img}_{o_2}(\phi)$. By definition

$$\mathit{img}_{o_1}(\phi) \vee \mathit{img}_{o_2}(\phi) = (\exists A.(\phi \wedge \tau_A^{nd}(o_1)))[A/A'] \vee (\exists A.(\phi \wedge \tau_A^{nd}(o_2)))[A/A'].$$

Since substitution commutes with disjunction we have

$$\mathit{img}_{o_1}(\phi) \vee \mathit{img}_{o_2}(\phi) = (\exists A.(\phi \wedge \tau_A^{nd}(o_1))) \vee (\exists A.(\phi \wedge \tau_A^{nd}(o_2)))[A/A'].$$

Since existential abstraction commutes with disjunction we have

$$\mathit{img}_{o_1}(\phi) \vee \mathit{img}_{o_2}(\phi) = (\exists A.((\phi \wedge \tau_A^{nd}(o_1)) \vee (\phi \wedge \tau_A^{nd}(o_2))))[A/A'].$$

By logical equivalence finally

$$\mathit{img}_{o_1}(\phi) \vee \mathit{img}_{o_2}(\phi) = (\exists A.(\phi \wedge (\tau_A^{nd}(o_1) \vee \tau_A^{nd}(o_2))))[A/A'].$$

Hence an alternative way of computing the union of images $\bigvee_{o \in O} \mathit{img}_o(\phi)$ is to first form the disjunction $\bigvee_{o \in O} \tau_A^{nd}(o)$ and then conjoin the formula with ϕ and only once existentially abstract the propositional variables in A. This may reduce the amount of computation because existential abstraction is in general expensive and it may be possible to simplify the formulae $\bigvee_{o \in O} \tau_A^{nd}(o)$ before existential abstraction.

The definitions of $\mathit{preimg}_o(\phi)$ and $\mathit{spreimg}_o(\phi)$ allow using $\bigvee_{o \in O} \tau_A^{nd}(o)$ in the same way.

Notice that defining progression for arbitrary formulae (sets of states) seems to require the explicit use of existential abstraction with potential exponential increase in formula size. A simple syntactic definition of progression similar to that of regression does not seem to be possible because the value of a state variable in a given state cannot be stated in terms of the values of the state variables in the successor state. This is because of the asymmetry of deterministic actions: the current state and an operator determine the successor state uniquely but the successor state and the operator do not determine the current state uniquely. In other words, the changes that take place are a function of the current state, but not a function of the successor state. Taking an action erases the information that determines which changes take place between two states. This information is visible in the predecessor state but not in the successor state.

4.3 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.

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.

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.

4.3.1 Memoryless plans

We use two definitions of plans. The simpler definition, formalized as mappings from states to operators, is applicable to fully observable planning problems only. The general definition

has a sufficient generality for all kinds of planning problems, and includes the sequential plans considered for deterministic planning as a special case.

Definition 4.19 *Let $\langle A, I, O, G, V\rangle$ be a succinct transition system. Let S be the set of states (all Boolean valuations of A). Then a* memoryless plan *is a partial function $\pi : S \to O$.*

To be able to execute a memoryless plan the current state must always be known, otherwise the correct operator in general cannot be correctly chosen. Hence we always assume full observability when using a memoryless plan. In the context of Markov decision processes (see Section 5.5) memoryless plans are also known as *policies* or *history dependent policies*.

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.

Definition 4.20 (Execution graph of a memoryless plan) *Let π be a memoryless plan for a succinct transition system $\langle A, I, O, G, V\rangle$. Then the* execution graph *of π and the transition system is the graph $\langle S, E\rangle$ where*

1. *$E \subseteq S \times S$ and*
2. *$(s, s') \in E$ if $s' \in img_o(s)$.*

The states s such that $s \models I$ are *initial nodes* of the execution graph, and the states s such that $s \models G$ are *goal nodes* of the execution graph. We have introduced the nodes of an execution graph as a notion that is separate from the states in the transition system because for the more general notion of plans we define next these two notions do not coincide.

4.3.2 Conditional plans

Plans determine what actions are executed. We formalize plans as a form of directed graphs. Each node is assigned an operator and information on zero or more successor nodes.

Definition 4.21 *Let $\Pi = \langle A, I, O, G, V\rangle$ be a succinct transition system. A* plan *for Π is a triple $\langle N, b, l\rangle$ where*

1. *N is a finite set of nodes,*
2. *$b \subseteq \mathcal{L} \times N$ maps initial states to starting nodes, and*
3. *$l : N \to O \times 2^{\mathcal{L}\times N}$ is a function that assigns each node n an operator and a set of pairs $\langle \phi, n'\rangle$ where ϕ is a formula over the observable state variables V and $n' \in N$ is a successor node.*

   Nodes n with $l(n) = \langle o, \emptyset\rangle$ for some $o \in O$ are terminal nodes.

Ignoring the operators and branch formulae in a plan π we can construct a graph $G(\pi) = \langle N, E\rangle$ with $E \subseteq N \times N$ such that $\langle n, n'\rangle \in E$ iff $\langle \phi, n'\rangle \in B$ for $l(n) = \langle o, B\rangle$ and some ϕ. A plan π is acyclic if there is no non-trivial path starting and ending at the same node in $G(\pi)$.

Plan execution starts from a node $n \in N$ and state s such that $\langle \phi, n\rangle \in b$ and $s \models I \wedge \phi$. Execution in node n with $l(n) = \langle o, B\rangle$ proceeds by executing the operator o and then testing for

each $\langle \phi, n' \rangle \in l(n)$ whether ϕ is true in all possible current states, and if it is, continuing execution from plan node n'. At most one ϕ may be true for this to be well-defined. Plan execution ends when none of the branch labels matches the current state. In a terminal node plan execution necessarily ends.

Definition 4.22 (Execution graph of a conditional plan) *Let $\langle A, I, O, G, V \rangle$ be a succinct transition system and $\pi = \langle N, b, l \rangle$ be a plan. Define* the execution graph *of π as a pair $\langle M, E \rangle$ where*

1. *$M = S \times (N \cup \{\bot\})$, where S is the set of Boolean valuations of A,*
2. *$E \subseteq M \times M$ has an edge from $\langle s, n \rangle \in S \times N$ to $\langle s', n' \rangle \in S \times N$ if and only if $l(n) = \langle o, B \rangle$ and for some $\langle \phi, n' \rangle \in B$*
   - *(a) $s' \in img_o(s)$ and*
   - *(b) $s' \models \phi$.*

   and an edge from $\langle s, n \rangle \in S \times N$ to $\langle s', \bot \rangle$ if and only if
   - *(a) $l(n) = \langle o, B \rangle$,*
   - *(b) $s' \in img_o(s)$, and*
   - *(c) there is no $\langle \phi, n' \rangle \in B$ such that $s' \models \phi$.*

The *initial nodes* of these execution graphs are nodes $\langle s, n \rangle$ such that $s \models I$ and $s \models \phi$ for some $\langle \phi, n \rangle \in b$.

The *goal nodes* of these execution graphs are nodes $\langle s, n \rangle$ such that $s \models G$.

We can translate every memoryless plan to a conditional plan.

Definition 4.23 *Let $\langle A, I, O, G, V \rangle$ be a succinct transition system. Let S be the set of all states on A. Let $\pi : S \to O$ be a memoryless plan. Define $C(\pi) = \langle N, b, l \rangle$ where*

1. *$N = O$,*
2. *$b = \{\langle FMA(\{s \in S | \pi(s) = o\}), o \rangle | o \in O\}$, and*
3. *$l(o) = (o, \{\langle FMA(\{s \in S | \pi(s) = o\}), o' \rangle | o' \in O\})$ for all $o \in O$.*

Above $FMA(T)$ is a formula ϕ such that $T = \{s \in S | s \models \phi\}$.

The memoryless plan π corresponds the conditional plan $C(\pi)$ in the sense that the subgraphs induced by the initial nodes are isomorphic, and this isomorphism preserves both initial and goal nodes.

4.3.3 Decision problems

There are different types of objectives the plans may have to fulfill. The most basic one, considered in much of AI planning research, is the reachability of a goal state. In this case every plan execution has a finite length. Also problems with infinite plan executions can be considered. A plan does not reach a goal and terminate, but is a continuing process that has to repeatedly reach goal states or avoid visiting bad states. Examples of these are different kinds of maintenance tasks: keep a building clean and transport mail from location A to location B.

We consider two objectives defined in terms of finite plan executions. The first objective requires, just like in deterministic planning, that a goal state is reached after a given number of operator executions.

Not all planning problems that have an intuitively plausible solution are solvable under this objective. A problem that is intuitively solvable is tossing a coin until it yields heads. This problem can be in practice always solved after a small number of tosses but because there is no guaranteed upper bound on the number of tosses that are needed, under the first objective it is not solvable. Hence we also consider another objective.

The second objective requires that from every state that can be reached by executing the plan, the plan is either a goal state or a goal state is reachable by executing the plan further.

The third objective we consider is defined in terms of infinite plan executions. The objective requires that all executions of a plan are infinite, and on every execution all states that are visited are goal states. This objective is known as *maintenance* because the transition system has to be kept in one of the goal states.

Other infinite horizon objectives that are defined in terms of expected costs/rewards are used in connection with probabilistic planning, see Section 5.3.

Definition 4.24 (Bounded reachability) *A plan π for $\langle A, I, O, G, V\rangle$ under the* Bounded Reachability *criterion fulfills the following.*

For all initial nodes x in the execution graph, all paths starting from x have a finite length and maximal paths end in a goal node.

Definition 4.25 (Unbounded reachability) *A plan π for $\langle A, I, O, G, V\rangle$ under the* Unbounded Reachability *criterion fulfills the following.*

For all initial nodes x in the execution graph, for every x' to which there is a path from x there is a path from x' of length ≥ 0 to some x'' such that x'' is a goal node without successor nodes.

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 nonzero 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 4.26 (Maintenance) *A plan π for $\langle A, I, O, G, V\rangle$ under the* Maintenance *criterion fulfills the following.*

All nodes x in the execution graph to which there is a path of length ≥ 0 from an initial node of the execution graph are goal nodes and have a successor node.

Example 4.27 Consider the plan $\langle N, b, l\rangle$ for a problem instance with the operators $O = \{o_1, o_2, o_3\}$, where

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

This could be visualized as the program.

1: o_3
CASE
ϕ_1: GOTO 1
ϕ_2: GOTO 2
$\neg(\phi_1 \vee \phi_2)$: GOTO 3
2: o_2
CASE
ϕ_4: GOTO 1
$\neg\phi_4$: GOTO 3
3:

Every plan $\langle N, b, l\rangle$ can be written as such a program. ■

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

4.4 Planning with full observability

Nondeterminism causes several The differences to algorithms for deterministic planning 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.[1] Hence we need plans with loops for representing the sequences of actions of unbounded length required for solving the problem.

Below in Section 4.4.1 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 4.4.2 we present an algorithm that also produces plans with loops. The structure of the algorithm is more complicated. The algorithms can be implemented by using data structures like binary decision diagrams which makes it possible to utilize the regularities in the state space and to solve very big planning problems. Representation of planning problems with these logic-based data structures is explained in Section 4.2.

4.4.1 An algorithm for constructing acyclic plans

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

The basic algorithm is for transition systems as in Definition 2.1 but the techniques in Section 4.2 can be directly applied to obtain a logic-based algorithm for succinct transition systems (Definition 2.11 in Section 2.3) that can be implemented easily by using any publicly available BDD package.

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

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

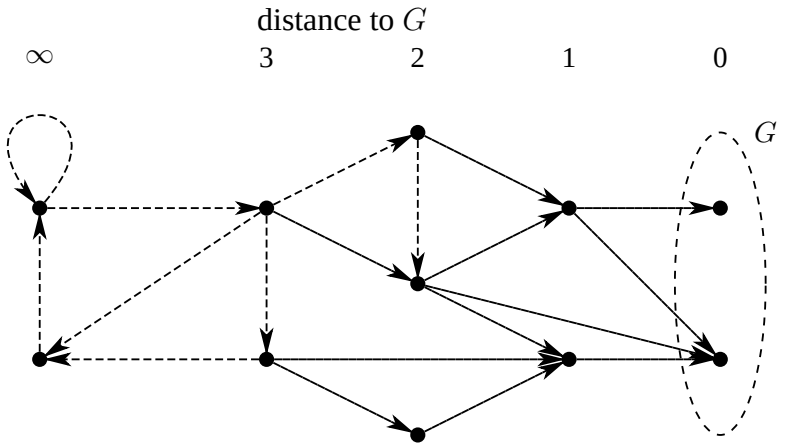


Figure 4.1: Goal distances in a nondeterministic transition system

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

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

Definition 4.28 *Let G be as set of states and O a set of operators, and let $D_0^{bwd}, D_1^{bwd}, \ldots$ be the backward distance sets for G and O. Then* the backward distance *of a state s to G is*

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

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

Example 4.29 We illustrate the distance computation by the diagram in Figure 4.1. The set of states with distance 0 is the set of goal states G. States with distance i are those for which there is an action that always leads to states with distance $i-1$ or smaller. In this example the action depicted by the solid arrow has this property for every state. The dashed arrows depict the second action which for no state is guaranteed to get closer to the goal states. States for which there is no finite upper bound on the number of actions for reaching a goal state have distance ∞. ■

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

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

Lemma 4.30 *Let a state s be in D_j. Then there is a plan that reaches a goal state from s by at most j operator applications.*

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

4.4.2 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 4.31 ■

The problem is those states that do not have a finite strong distance as defined Section 4.4.1. 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* is given in Figure 4.2.

We introduce some terminology. Let S be a set of states, O a set of operators, and $x : S \to O$ a mapping from states to operators. A sequence $s_0, \ldots, s_n$ of states is an *execution* if for all $i \in \{1, \ldots, n\}$ there is $o \in O$ such that $s_i \in \mathit{img}_o(s_{i-1})$, and it is an *execution of* x if $s_i \in \mathit{img}_{x(s_{i-1})}(s_{i-1})$ for all $i \in \{1, \ldots, n\}$.

Lemma 4.32 (Procedure prune) *Let S be a set of states, O a set of operators and $G \subseteq S$ a set of states. Then the procedure call prune(S,O,G) will terminate after a finite number of steps returning a set of states $W \subseteq S$ such that there is function $x : W \to O$ such that*

1. *for every $s \in W$ there is an execution $s_0, s_1, \ldots, s_n$ of x with $n \geq 1$ such that $s = s_0$ and $s_n \in G$,*
2. *$\mathit{img}_{x(s)}(\{s\}) \subseteq W \cup G$ for every $s \in W$, and*
3. *There is no function x satisfying the above properties for states not in W: for every $s \in S \backslash W$ and function $x' : S \to O$ there is an execution $s_0, \ldots, s_n$ of x' such that $s = s_0$ and there is no $m \geq n$ and execution $s_n, s_{n+1}, \ldots, s_m$ such that $s_m \in G$.*

Proof: The proof is by two nested induction proofs that respectively correspond to the repeat-until loops on lines 9 and 13 in the procedure *prune*. If there is no plan that is guaranteed to reach a goal state from a state s, then this is because for any plan after some number of executions steps i it is possible to reach a state from which no sequence actions can reach a goal state. A plan covering

```
1:  procedure prune(S,O,G);
2:  W_{-1} := S;
3:  W_0 := ∅;
4:  repeat
5:    W'_0 := W_0;
6:    W_0 := (W'_0 ∪ ⋃_{o∈O} preimg_o(W'_0 ∪ G)) ∩ S;
7:  until W_0 = W'_0;      (* States from which a goal state can be reached by ≥ 1 operators *)
8:  i := 0;
9:  repeat
10:   i := i + 1;
11:   k := 0;
12:   S_0 := ∅;
13:   repeat
14:     k := k + 1;          (* States from which a state in G is reachable with ≤ k steps. *)
15:     S_k := S_{k-1} ∪ ⋃_{o∈O}(S ∩ preimg_o(S_{k-1} ∪ G) ∩ spreimg_o(W_{i-1} ∪ G));
16:   until S_k = S_{k-1};          (* States that stay within W_{i-1} before reaching G. *)
17:   W_i := S_k;
18: until W_i = W_{i-1};          (* States in W_i stay within W_i before reaching G. *)
19: return W_i;
```

Figure 4.2: Algorithm for detecting a loop that eventually makes progress

all other states exists with an execution reaching a goal state in some h steps. The outer loop and induction go through $i = 0, 1, 2, \ldots$ and the inner loop and induction through $h = 0, 1, 2, \ldots$.

Induction hypothesis: There is function $x : W_i \to O$ such that

1. for every $s \in W_i$ there is an execution $s_0, \ldots, s_n$ of x such that $n \geq 1$, $s = s_0$ and $s_n \in G$,
2. $\mathit{img}_{x(s)}(\{s\}) \subseteq W_{i-1} \cup G$ for every $s \in W_i$, and
3. for all functions $x' : S \to O$ and states $s \in S \backslash W_i$ there is $i' \in \{0, \ldots, i\}$ and an execution $s_0, \ldots, s_{i'}$ of x' such that $s_0 = s$ and there is no $h \geq i'$ and execution $s_{i'}, s_{i'+1}, \ldots, s_h$ such that $s_h \in G$.

Base case $i = 0$:

1. W_0 has been computed to fulfill exactly this property. We denote the value of the variables W_0 in the end of iteration i of the first repeat-until loop by $W_{0,i}$.

   Induction hypothesis:

   (a) There is a function $x : W_{0,j} \to O$ such that there is an execution of x for every $s \in W_{0,j}$ of length $j \geq 1$ reaching a state in G.

   (b) For states not in $W_{0,j}$ there is no x with this property.

   Base case $j = 1$: After the first iteration $W_{0,1} = \bigcup_{o \in O} \mathit{preimg}_o(G)$. Hence for every $s \in W_{0,1}$ assign $x(s) = o$ for any o such that $s \in \mathit{preimg}_o(G)$.

   (a) Now there is an execution of length 1 from any $s \in W_{0,1}$ to a state in G.

(b) For states not in $W_{0,1}$ no one operator may reach a state in G.

Inductive case $j \geq 2$: By induction hypothesis there is a function x with execution of length $j-1 \geq 1$ for reaching a state in G for every state for which such an execution exists. We extend this function to cover states $s \in W_{0,j} \backslash W_{0,j-1}$ as follows: $x(s) = o$ for any o such that $s \in \mathit{preimg}_o(W_{0,j-1} \cup G)$.

(a) For any $s \in W_{0,j}$ there is an execution of x reaching a state in G because for states $s \in W_{0,j-1}$ this is by the induction hypothesis, and for states in $W_{0,j} \backslash W_{0,j-1}$ applying the operator $x(s)$ may reach a state in $W_{0,j-1}$ for which an execution reaching G exists by the induction hypothesis.

(b) Let s be a state such that there is a function $x' : S \rightarrow O$ with an execution that reaches G from s with j steps. Hence there is a state s' for which an execution with x' reaches G from s' with $j-1$ steps. Hence by the induction hypothesis $s \in W_{0,j-1}$ and consequently $s \in \mathit{preimg}_{x'(s)}(W_{0,j-1})$. Therefore for any state not in $W_{0,j}$ there is no such function x'.

2. Because $W_{-1} = S$ trivially $\mathit{img}_{x(s)}(\{s\}) \subseteq W_{-1} \cup G$.

3. States $s \in W_0 \backslash W_{-1}$ are exactly those states from which no operator sequence leads to G by construction of W_0, as shown above.

Inductive case $i \geq 1$: For the inner *repeat-until* loop we prove inductively the following.
Induction hypothesis: There is function $x : S_k \rightarrow O$ such that

1. for every $s \in S_k$ there is an execution $s_0, s_1, \ldots, s_n$ of x such that $n \in \{1, \ldots, k\}$, $s = s_0$ and $s_n \in G$,

2. $\mathit{img}_{x(s)}(\{s\}) \subseteq W_{i-1} \cup G$ for every $s \in S_k$, and

3. for all functions $x' : S \rightarrow O$ and states $s \in S \backslash S_k$ either

   (a) there is $i' \in \{0, \ldots, i\}$ and an execution $s_0, \ldots, s_{i'}$ of x' such that $s_0 = s$ and there is no $h \geq i'$ and execution $s_{i'}, s_{i'+1}, \ldots, s_h$ such that $s_h \in G$, or

   (b) there is no $k' \in \{1, \ldots, k\}$ and an execution $s_0, \ldots, s_{k'}$ of x' such that $s_0 = s$ and $s_{k'} \in G$.

Base case $k = 0$: Because $S_0 = \emptyset$, cases (1) and (2) trivially hold for every $s \in S_0$. It remains to show the third component of the induction hypothesis.

3. For any $s \in S \backslash S_0 = S$ (3b) is satisfied because it requires executions to be longer than $k = 0$.

Inductive case $k \geq 1$: We extend the function $x : S_{k-1} \rightarrow O$ to cover states in $S_k \backslash S_{k-1}$. Let s be any state in S_k. If $s \in S_{k-1}$ then properties (1) and (2) are by the induction hypothesis. Otherwise $s \in S_k \backslash S_{k-1}$. Therefore by definition of S_k, $s \in \mathit{preimg}_o(S_{k-1} \cup G) \cap \mathit{spreimg}_o(W_{i-1} \cup G)$ for some $o \in O$.

1. Because $s \in \mathit{preimg}_o(S_{k-1} \cup G)$ for some $o \in O$, by (4) of Lemma 2.2 either $s \in \mathit{preimg}_o(S_{k-1})$ or $s \in \mathit{preimg}_o(G)$.

   If $s \in \mathit{preimg}_o(G)$ then we set $x(s) = o$. The desired execution just consists of s and a state $s' \in G$.

   If $s \in \mathit{preimg}_o(S_{k-1}) \backslash \mathit{preimg}_o(G)$ then there is a state $s' \in S_{k-1}$ such that $s' \in \mathit{img}_o(\{s\})$. By the induction hypothesis there is an execution of x starting from s' that ends in a goal state. For s such an execution is obtained by prefixing with o, so we define $x(s) = o$.

2. Because $s \in \mathit{spreimg}_o(W_{i-1} \cup G)$, by (2) and (3) of Lemma 2.2 $\mathit{img}_o(\{s\}) \subseteq W_{i-1} \cup G$.

3. Take any $s \in S \backslash S_k$. Now for every operator $o \in O$, either $s \notin \mathit{spreimg}_o(W_{i-1} \cup G)$ or $s \notin \mathit{preimg}_o(S_{k-1} \cup G)$. Consider any function $x' : S \to O$ such that $x'(s) = o$.

   In the first case by the outer induction hypothesis there is $i' \in \{0, \ldots, i-1\}$ and an execution $s_0, \ldots, s_{i'}$ of x' such that $s_0 \in \mathit{img}_o(s)$ and there is no $h \geq i'$ and execution $s_{i'}, s_{i'+1}, \ldots, s_h$ such that $s_h \in G$. Hence executing o first could similarly lead to the state $s_{i'}$ from which no goal could be reached, now requiring i steps.

   In the second case by the inner induction hypothesis for all $s' \in \mathit{img}_o(s)$ there is no execution of length $k-1$ ending in a goal state.

   Because this holds for any $o \in O$, every x' has one of these properties.

This completes the inner induction. To establish the induction step of the outer induction consider the following. The inner repeat-until loops ends when $S_k = S_{k-1}$. This means that $S_z = S_k$ for all $z \geq k$. Hence executions for reaching a goal state for (1) and (3) are allowed to have arbitrarily high length k. The outer induction hypothesis is obtained from the inner induction hypothesis by removing the upper bound and replacing S_k by W_i. By construction $W_i = S_k$.

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

The algorithm in Figure 4.3 uses *prune* to identify states from which a goal state is reachable by some execution and no execution leads to a state outside the set. On line 4 the algorithm tests whether the reachability of a goal state can be guaranteed for the initial states. If not, the algorithm terminates. Starting on line 7 the algorithm computes the *weak backward distances* to G for all states in L. Finally, starting on line 11 the algorithm assigns every state in $L \backslash G$ an operator that may reduce the distance to a goal by one.

4.4.3 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. A *maintanence goal* is a goal that has to hold in all time points. To achieve a maintenance goals a plan has to keep the state of the system in a goal state indefinitely.

Plans that satisfy a maintenance goal have only infinite executions.

Figure 4.4 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

```
 1: procedure FOplancyclic(I,O,G)
 2: S := the set of all states;
 3: L := G∪ prune(S,O,G);
 4: if I ⊈ L then return false;
 5: D_0 := G;                                   (* States with weak backward distance 0 *)
 6: i := 1;
 7: repeat
 8:    D_i := D_{i−1} ∪ ⋃_{o∈O}(preimg_o(D_{i−1}) ∩ spreimg_o(L));
 9:    i := i + 1;
10: until D_i = D_{i−1};
11: for each s ∈ D_i\G do
12:    d := number such that s ∈ D_d\D_{d−1};     (* State has weak backward distance d. *)
13:    assign π(s) := o such that img_o(s) ⊆ L and img_o(s) ∩ D_{d−1} ≠ ∅;
14: end do
```

Figure 4.3: Algorithm for nondeterministic planning with full observability

```
 1: procedure FOplanMAINTENANCE(I,O,G)
 2: i := 0;
 3: G_0 := G;
 4: repeat
 5:    i := i + 1;          (* The subset of G_{i−1} from which G_{i−1} can be always reached. *)
 6:    G_i := ⋃_{o∈O} (spreimg_o(G_{i−1}) ∩ G_{i−1});
 7: until G_i = G_{i−1};
 8: return G_i;
 9: for each s ∈ G_i do
10:    assign π(s) := o such that img_o(s) ⊆ G_i;
11: end do
```

Figure 4.4: Algorithm for nondeterministic planning with full observability and maintenance goals

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 4.33 *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 4.4.*

Then $G' \subseteq G$ and there is a plan π such that $img_{\pi(s)}(s) \subseteq G'$ for every $s \in G'$, and for every $s \in S \backslash G'$ and every plan π' there is $n \geq 1$ and and an execution $s_0, \ldots, s_n$ of π' with $s_0 = s$ such that $s_n \notin G$.

Proof:

Induction hypothesis:

1. $G_i \subseteq G$,
2. there is a plan π such that $img_{\pi(s)}(s) \subseteq G_{i-1}$ for every $s \in G_i$, and
3. for every $s \in S \backslash G_i$ and every plan π' there is $n \in \{1, \ldots, i\}$ and an execution $s_0, \ldots, s_n$ of π' with $s_0 = s$ such that $s_n \notin G$.

Base case $i = 1$:

1. $G_1 \subseteq G_0 = G$ by construction.
2. By construction $G_1 = \bigcup_{o \in O}(spreimg_o(G_0) \cap G_0)$. Hence for every $s \in G_1$ there is $o \in O$ such that $s \in spreimg_o(G_0) \cap G_0 \subseteq spreimg_o(G_0)$. Hence $img_o(s) \subseteq G_0$. Define $\pi(s) = o$. Hence $img_{\pi(s)}(s) \subseteq G_0$ for every $s \in G_1$.
3. Consider any $s \in S \backslash G_1$. For every $o \in O$ $img_o(s) \not\subseteq G_0 = G$ because $s \notin G_1$. Hence for every $s \in S \backslash G_1$ and every plan π' there is an execution s_0, s_1 of π' with $s_0 = s$ such that $s_1 \notin G$.

Inductive case $i \geq 2$:

1. As $G_i \subseteq G_{i-1}$ and by the induction hypothesis $G_{i-1} \subseteq G$, $G_i \subseteq G$.
2. By construction $G_i = \bigcup_{o \in O}(spreimg_o(G_{i-1}) \cap G_{i-1})$. Hence for every $s \in G_i$ there is $o \in O$ such that $s \in spreimg_o(G_{i-1}) \cap G_{i-1} \subseteq spreimg_o(G_{i-1})$. Hence $img_o(s) \subseteq G_{i-1}$. Define $\pi(s) = o$. Hence there is a plan π such that $img_{\pi(s)}(s) \subseteq G_{i-1}$ for every $s \in G_i$.
3. Consider any $s \in S \backslash G_i$. Hence $s \notin spreimg_o(G_{i-1})$ for every $o \in O$. Hence for every $o \in O$ there is $s' \in S \backslash G_{i-1}$ such that $s' \in img_o(s)$. By the induction hypothesis for every plan π' there is $n \in \{1, \ldots, i-1\}$ and an execution $s_0, \ldots, s_n$ of π' with $s_0 = s'$ such that $s_n \notin G$. Hence for every plan π' there is also $n' = n + 1 \in \{1, \ldots, i\}$ and an execution $s, s_0, \ldots, s_n$ of π' with $s_0 = s'$ such that $s_n \notin G$.

Because G_i are finite sets and $G_i \subseteq G_{i-1}$ and every G_{i+1} is a function of G_i, $G_j = G_{j-1}$ for some j and the loop iteration terminates after a finite number of iterations.

Now the claim of the lemma are obtained as follows.

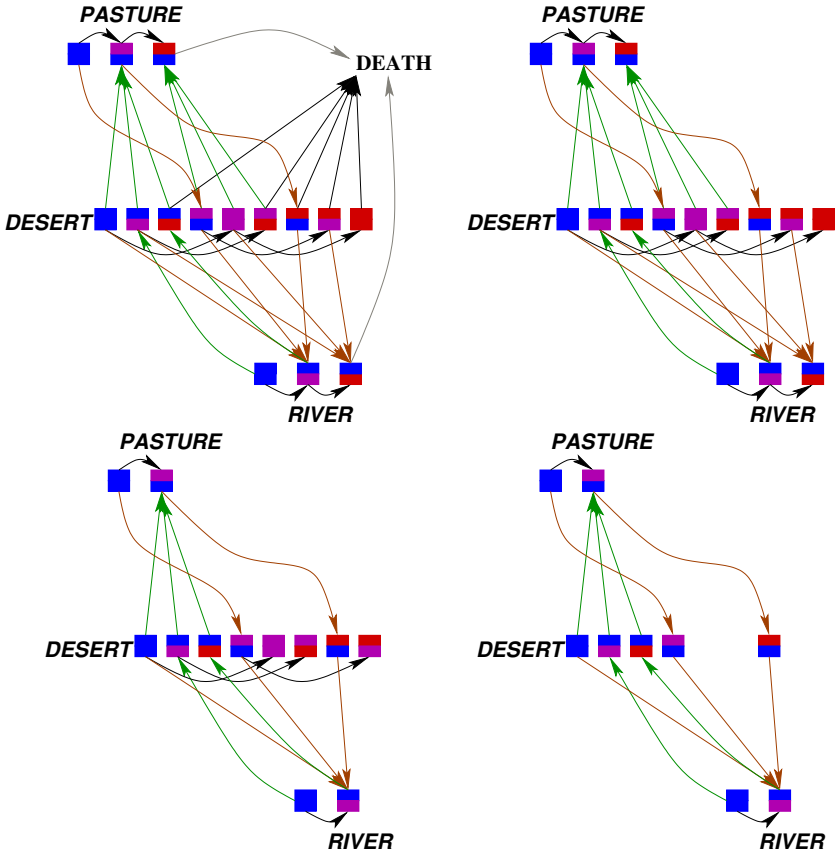


Figure 4.5: Example run of the algorithm for maintenance goals

1. The G' that is returned is $G' = G_j$. By the induction proof $G_j \subseteq G$.

2. By the termination condition of the loop $G_j = G_{j+1} = G'$. Hence by the results of the induction proof there is a plan π such that $\mathit{img}_{\pi(s)}(s) \subseteq G'$ for every $s \in G'$.

3. Because $G' = G_j = G_{j-1}$ and G_j is a function of G_{j-1}, the sets G_k for all $k \geq j$ equal G_j. Hence the constant n for the length of executions leading outside G can be arbitrarily high. By the results of the induction proof for every $s \in S \backslash G'$ and every π' there is $n \geq 0$ and an execution $s_0, \ldots, s_n$ of π' with $s_0 = s$ such that $s_n \notin G$.

□

Example 4.34 Consider the problem depicted in Figure 4.5. 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 point after respectively leaving the pasture or the river. If either one reaches level 3 the animal dies. The hunger and thirst levels are indicated by different colors: the upper halves of the rectanges show thirst level and the lower halves the hunger level. Blue means no hunger or thirst, 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

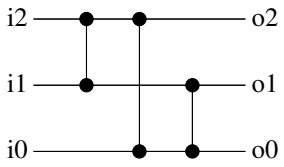


Figure 4.6: A sorting network with three inputs

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.

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. ■

4.5 Planning without observability

4.5.1 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 4.35 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 4.6 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 difference 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 $\models I \rightarrow \mathit{regr}_o(\phi)$ for the belief states. With one initial state this is an easy polynomial time test $I \models \mathit{regr}_o(\phi)$ of whether $\mathit{regr}_o(\phi)$ 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 backward distances in the state space.

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

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. Backward distances yield an admissible distance heuristic for belief states.

Definition 4.36 (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.

4.6 Planning as satisfiability in the propositional logic and QBF

The techniques presented in Sections 3.6 and 3.6.5 can be extended to nondeterministic operators. The notion of parallel application of operators and partially ordered plans can be generalized to nondeterministic operators.

Let T be a set of operators and s a state such that $s \models c$ for every $\langle c, e\rangle \in T$ and $E_1 \cup \cdots \cup E_n$ is consistent for for any $E_i \in [e_i]_s, i \in \{1, \ldots, n\}$ and $T = \{\langle c_1, e_1\rangle, \ldots \langle c_n, e_n\rangle\}$. Then define $\mathit{img}_T(s)$ as the set of states s' that are obtained from s by making $E_1 \cup \cdots \cup E_n$ true in s where $E_i \in [e_i]_s$ for every $i \in \{1, \ldots, n\}$. We also use the notation sTs' for $s' \in \mathit{img}_T(s)$ and $\mathit{img}_T(S) = \bigcup_{s\in S} \mathit{img}_T(s)$.

4.6.1 Advanced translation of nondeterministic operators into propositional logic

In Section 4.1.2 we showed how nondeterministic operators can be translated into formulae in the propositional logic. This translation is not sufficient for reasoning about actions and plans in a setting with more than one agent. This is because the formulae $\tau_A^{nd}(o_1) \vee \cdots \vee \tau_A^{nd}(o_n)$ do not distinguish between the choice of operator in $\{o_1, \ldots, o_n\}$ and the nondeterministic effects (the opponent) of each operator, even though the former is controllable and the latter is not.

In nondeterministic planning in general we have to treat the controllable and uncontrollable choices differently. We cannot do this practically in the propositional logic but by using quantified Boolean formulae (QBF) we can. For the QBF representation of nondeterministic operators we universally quantify over all uncontrollable eventualities (nondeterminism) and existentially quantify over controllable eventualities (the choice of operators).

We need to universally quantify over all the nondeterministic choices because for every choice the remaining operators in the plan must lead to a goal state. This is achieved by associating with every atomic effect a formula that is true if and only if that effect is executed, similarly to functions $\mathit{EPC}_l(e)$ in Definition 3.1, so that for l to become true the universally quantified auxiliary variables that represent nondeterminism have to have values corresponding to an effect that makes l true.

The operators are assumed to be in normal form. For simplicity of presentation we further transform nondeterministic choices $e_1|\cdots|e_n$ so that only binary choices exist. For example $a|b|c|d$ is replaced by $(a|b)|(c|d)$. Each binary choice can be encoded in terms of one auxiliary variable.

The condition for the atomic effect l to be executed when e is executed is $\mathit{EPC}_l^{nd}(e, \sigma)$. The sequence σ of integers is used for deriving unique names for auxiliary variables in $\mathit{EPC}_l^{nd}(e, \sigma)$. The sequences correspond to paths in the tree formed by nested nondeterministic choices and

conjunctions.

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

The translation of nondeterministic operators into the propositional logic is similar to the translation for deterministic operators given in Section 3.6.4.

Nondeterminism is encoded by making the effects conditional on the values of the auxiliary variables x_σ. Different valuations of these auxiliary variables correspond to different nondeterministic effects.

The following frame axioms express the conditions under which state variables $a \in A$ may change from true to false and from false to true. Let $e_1, \ldots, e_n$ be the effects of $o_1, \ldots, o_n$ respectively. Each operator $o \in O$ has a unique index $\Omega(o)$.

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

For $o = \langle c, e\rangle \in O$ there is a formula for describing values of state variables in the predecessor and successor states when the operator is applied.

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

Example 4.37 Consider $o_1 = \langle \neg a, (b|(c \rhd d)) \wedge (a|c)\rangle$ and $o_2 = \langle \neg b, (((d \rhd b)|c)|a)\rangle$. The application of these operators is described by the following formulae.

$$\begin{array}{ll} \neg(a \wedge \neg a') & (\neg a \wedge a') \rightarrow ((o_1 \wedge x_{12}) \vee (o_2 \wedge \neg x_2)) \\ \neg(b \wedge \neg b') & (\neg b \wedge b') \rightarrow ((o_1 \wedge x_{11}) \vee (o_2 \wedge x_2 \wedge x_{21} \wedge d)) \\ \neg(c \wedge \neg c') & (\neg c \wedge c') \rightarrow ((o_1 \wedge \neg x_{12}) \vee (o_2 \wedge x_2 \wedge \neg x_{21})) \\ \neg(d \wedge \neg d') & (\neg d \wedge d') \rightarrow (o_1 \wedge \neg x_{11} \wedge c) \\ o_1 \rightarrow \neg a & \\ (o_1 \wedge x_{12}) \rightarrow a' & (o_1 \wedge x_{11}) \rightarrow b' \\ (o_1 \wedge \neg x_{12}) \rightarrow c' & (o_1 \wedge \neg x_{11} \wedge c) \rightarrow d' \\ o_2 \rightarrow \neg b & \\ (o_2 \wedge \neg x_2) \rightarrow a' & (o_2 \wedge x_2 \wedge x_{21} \wedge d) \rightarrow b' \\ (o_2 \wedge x_2 \wedge \neg x_{21}) \rightarrow c' & \end{array}$$

■

Two operators o and o' may be applied in parallel only if they do not interfere. Hence we use formulae

$$\neg(o \wedge o')$$

for all operators o and o' that interfere and $o \neq o'$.

Let X be the set of auxiliary variables x_σ in all the above formulae. The conjunction of all the above formulae is denoted by

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

We need two auxiliary definitions for proving properties about these formulae and the translation of nondeterministic operators into the propositional logic.

Let $\Xi_\sigma(e)$ be the set of propositional variables $x_{\sigma'}$ in the translation of the effect e with a given σ. This is equal to the set of variables $x_{\sigma'}$ in formulae $EPC_a^{nd}(e,\sigma)$ and $EPC_{\neg a}^{nd}(e,\sigma))$ for all $a \in A$.

Definition 4.38 *Define the set of literals $[e]_s^{\sigma,v}$ which are the active effects of e when e is executed in state s and nondeterministic choices are determined by the valuation v of propositional variables $\Xi_\sigma(e)$ as follows.*

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

Lemma 4.39 *Let s be a state and $\{v_1,\ldots,v_n\}$ all valuations of $X = \Xi_\sigma(e)$. Then $\bigcup_{1\leq i\leq n}[e]_s^{\sigma,v_i} = [e]_s$.*

Lemma 4.40 *Let O and $T \subseteq O$ be sets of operators. Let s and s' be states. Let v_x a valuation of $X = \bigcup_{\langle c,e\rangle\in O}\Xi_{\Omega(\langle c,e\rangle)}(e)$. Let v_o be a valuation of O such that $v_o(o) = 1$ iff $o \in T$.*
Then $s \cup s'[A'/A] \cup v_o \cup v_x \models \mathcal{R}_3(A,A',O,X)$ if and only if

1. *$s \models a$ iff $s' \models a$ for all $a \in A$ such that $\{a,\neg a\} \cap \bigcup_{\langle c,e\rangle\in T}[e]_s^{\Omega(\langle c,e\rangle),v_x} = \emptyset$,*
2. *$s' \models \bigcup_{\langle c,e\rangle\in T}[e]_s^{\Omega(\langle c,e\rangle),v_x}$, and*
3. *$s \models c$ for all $\langle c,e\rangle \in T$.*

The number of auxiliary variables x_σ can be reduced when two operators o and o' interfere. Since they cannot be applied simultaneously the same auxiliary variables can control the nondeterminism in both operators. To share the variables rename the ones occurring in the formulae for one of the operators so that the variables needed for o is a subset of those for o' or vice versa. Having as small a number of auxiliary variables as possible may be important for the efficiency for algorithms evaluating QBF and testing propositional satisfiability.

The formulae $\mathcal{R}_3(A,A',O,X)$ can be used for plan search with algorithms that evaluate QBF (Section 4.6.2) as well as for testing by a satisfiability algorithm whether a conditional plan (with full, partial or no observability) that allows several operators simultaneously indeed is a valid plan.

4.6.2 Finding plans by evaluation of QBF

In deterministic planning in propositional logic (Section 3.6) the problem is to find a sequence of operators so that a goal state is reached when the operators are applied starting in the initial state. When there are several initial states, the operators are nondeterministic and it is not possible to use observations during plan execution for selecting operators, the problem is to find an operator sequence so that a goal state is reached in all possible executions of the operator sequence. There may be several executions because there may be several initial states and the operators may be nondeterministic. Expressing the quantification over all possible executions cannot be concisely

expressed in the propositional logic. This is the reason why quantified Boolean formulae are used instead.

The existence of an n-step partially-ordered plan that reaches a state satisfying G from any state satisfying the formula I can be tested by evaluating the QBF Φ_n^{qpar} defined as

$$\exists V_{plan} \forall V_{nd} \exists V_{exec}$$
$$I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \mathcal{R}_3(A^1, A^2, O^1, X^1) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1}) \wedge G^n).$$

Here $V_{plan} = O^0 \cup \cdots \cup O^{n-1}$, $V_{nd} = A^0 \cup X^0 \cup \cdots \cup X^{n-1}$ and $V_{exec} = A^1 \cup \cdots \cup A^n$. Define $\Phi_n^{qparM} = I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \mathcal{R}_3(A^1, A^2, O^1, X^1) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1}) \wedge G^n)$. The valuation of V_{plan} corresponds to a sequence of sets of operators. For a given valuation of V_{plan} the valuation of V_{nd} determines the execution of these operators. The valuation of V_{exec} is uniquely determined by the valuation of $V_{plan} \cup V_{nd}$.

The algorithms for evaluating QBF that extend the Davis-Putnam procedure traverse an and-or tree in which the and-nodes correspond to universally quantified variables and or-nodes correspond to existentially quantified variables. If the QBF is *true* then these algorithms return a valuation of the outermost existential variables. For a true Φ_n^{qpar} this valuation of V_{plan} corresponds to a plan that can be constructed like the plans in the deterministic case in Section 3.6.5.

Theorem 4.41 *The QBF Φ_n^{qpar} has value* true *if and only if there is a sequence $T_0, \ldots, T_{n-1}$ of sets of operators such that for every $i \in \{0, \ldots, n\}$ and every sequence $s_0, \ldots, s_i$ such that*

1. $s_0 \models I$ *and*
2. $s_0 T_0 s_1 T_1 s_2 \cdots s_{i-1} T_{i-1} s_i$

T_i is applicable in s_i if $i < n$ and $s_i \models G$ if $i = n$.

Proof: We first prove the implication from left to right. Since Φ_n^{qpar} is true there is a valuation v_{plan} of $V_{plan} = O^0 \cup \cdots \cup O^{n-1}$ such that for all valuations v_{nd} of $V_{nd} = A^0 \cup X^0 \cup \cdots \cup X^{n-1}$ there is a valuation v_{exec} of $V_{exec} = A^1 \cup \cdots \cup A^n$ such that $v_{plan} \cup v_{nd} \cup v_{exec} \models I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1}) \wedge G^n)$.

Let $T_0, \ldots, T_{n-1}$ be the sequence of sets of operators such that for all $o \in O$ and $i \in \{0, \ldots, n-1\}$, $o \in T_i$ if and only if $v_{plan}(o^i) = 1$. We prove the right hand side of the theorem by induction on n.

Induction hypothesis: For every $s_0, \ldots, s_i$ such that $s_0 \models I$ and $s_0 T_0 s_1 T_1 s_2 \cdots s_{i-1} T_{i-1} s_i$:

1. T_i is applicable in s_i if $i < n$.
2. $s_i \models G$ if $i = n$.

Base case $i = 0$: Let s_0 be any state sequence such that $s_0 \models I$.

1. If $0 < n$ then we have to show that T_0 is applicable in s_0.

   Let $E = E_1 \cup \cdots \cup E_m$ for all $j \in \{1, \ldots, m\}$ and any $E_j \in [e_j]_{s_0}$, where $e_1, \ldots, e_m$ are respectively the effects of the operators $o_1, \ldots, o_m$ in T_0. Such sets E are the possible active effects of T_0.

   We have to show that E is consistent and the preconditions of operators in T_0 are true in s_0.

   By Lemma 4.39 there is a valuation v of X such that $E = \bigcup_{\langle c,e\rangle \in T_0} [e]_{s_0}^{\Omega(\langle c,e\rangle),v}$.

   Let v_{nd} be any valuation of V_{nd} such that $s_0[A^0/A] \subseteq v_{nd}$ and $v[X^0/X] \subseteq v_{nd}$. Since Φ_n^{qpar} is true there is a valuation of v_{exec} such that $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$.

   Since $v_{nd} \models I^0$ also $v_{plan} \cup v_{nd} \cup v_{exec} \models \mathcal{R}_3(A^0, A^1, O^0, X^0)$. Hence by Lemma 4.40 the preconditions of operators in T_0 are true in s_0 and $s_1 \models E$ where s_1 is the state such that $s_1(a) = v_{exec}(a^1)$ for all $a \in A$. Since E was chosen arbitrarily from the sets of possible sets of active effects of T_0 and it is consistent, T_0 is applicable in s_0.

2. If $n = 0$ then $V_{plan} = V_{exec} = \emptyset$ and $\forall V_{nd}(I^0 \rightarrow G^0)$ is true, and $v_{nd} \models G^0$ for every valuation v_{nd} of V_{nd} such that $v_{nd} \models I^0$.

Inductive case $i \geq 1$: Let $s_0, \ldots, s_i$ be any sequence such that $s_0 \models I$ and $s_0 T_0 s_1 \ldots s_{i-1} T_{i-1} s_i$.

1. If $i < n$ then we have to show that T_i is applicable in s_i.

   Let $E = E_1 \cup \cdots \cup E_m$ for all $j \in \{1, \ldots, m\}$ and any $E_j \in [e_j]_{s_i}$, where $e_1, \ldots, e_m$ are respectively the effects of the operators $o_1, \ldots, o_m$ in T_i. Such sets E are the possible active effects of T_i.

   We have to show that E is consistent and the preconditions of operators in T_i are true in s_i.

By Lemma 4.39 there is a valuation v of X such that $E = \bigcup_{\langle c,e\rangle \in T_i} [e]_{s_i}^{\Omega(\langle c,e\rangle),v}$.

Since by the induction hypothesis $s_j T_j s_{j+1}$ for all $j \in \{0,\ldots,i-1\}$, by Lemma 4.39 for every $j \in \{0,\ldots,i-1\}$ there is a valuation v_j^x of X such that $s_j[A/A^j] \cup s_{j+1}[A'/A^{j+1}] \cup v_o \cup v_j^x \models \mathcal{R}_3(A, A', O, X)$ where v_o assigns every $o \in O$ value 1 iff $o \in T_j$.

Let v_{nd} be any valuation of V_{nd} such that $s_0[A^0/A] \subseteq v_{nd}$ and $v[X^i/X] \subseteq v_{nd}$ and $v_j^x[X^j/X] \subseteq v_{nd}$ for all $j \in \{0,\ldots,i-1\}$.

Since Φ_n^{qpar} is true there is a valuation of v_{exec} such that $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$.

Since $v_{nd} \models I^0$ also $v_{plan} \cup v_{nd} \cup v_{exec} \models \mathcal{R}_3(A^i, A^{i+1}, O^i, X^i)$. Hence by Lemma 4.40 the preconditions of operators in T_i are true in s_i and $s_{i+1} \models E$ where s_{i+1} is a state such that $s_{i+1}(a) = v_{exec}(a^{i+1})$ for all $a \in A$. Since any E is consistent, T_i is applicable in s_i.

2. If $i = n$ we have to show that $s_n \models G$. Like in the proof for the previous case we construct valuations v_{nd} and v_{exec} matching the execution $s_0,\ldots,s_n$ and since $v_{plan} \cup v_{nd} \cup v_{exec} \models I^0 \rightarrow G^n$ we have $s_n \models G$.

Then we prove the implication from right to left. So there is sequence $T_0,\ldots,T_{n-1}$ for which all executions are defined and reach G.

We show that Φ_n^{qpar} is true: there is valuation v_{plan} of $V_{plan} = O^0 \cup \cdots \cup O^{n-1}$ such that for every valuation v_{nd} of $V_{nd} = A^0 \cup X^0 \cup \cdots \cup X^{n-1}$ there is a valuation v_{exec} of $V_{exec} = A^1 \cup \cdots \cup A^n$ such that $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$.

We define the valuation v_{plan} of V_{plan} by $o \in T_i$ iff $v_{plan}(o^i) = 1$ for every $o \in O$ and $i \in \{0,\ldots,n-1\}$.

Take any valuation v_{nd} of V_{nd}. Define the state s_0 by $s_0(a) = 1$ iff $v_{nd}(a^0) = 1$ for every $a \in A$.

If $s_0 \not\models I$ then $v_{nd} \not\models I^0$ and $v_{plan} \cup v_{nd} \cup v_{exec} \models \Phi_n^{qparM}$ for any valuation v_{exec} of V_{exec}.

It remains to consider the case $s_0 \models I$.

Define for every $i \in \{1,\ldots,n\}$ sets E_i and states s_i as follows.

1. Let v_x^i be a valuation of X such that $v_x^i(x) = v_{nd}(x^{i-1})$ for every $x \in X$.

2. Let $E_i = \bigcup_{\langle c,e\rangle \in T_{i-1}} [e]_{s_{i-1}}^{\Omega(\langle c,e\rangle),v_x^i}$.

   We show below that this is the set of literals made true by T_{i-1} in s_{i-1}.

3. Define $s_i(a) = 1$ iff $a \in E_i$ or $s_{i-1}(a) = 1$ and $\neg a \notin E_i$, for every $a \in A$.

Let $v_{exec} = s_1[A^1/A] \cup \cdots \cup s_n[A^n/A]$.

Induction hypothesis: $v_{plan} \cup v_{nd} \cup s_1[A^1/A] \cup \cdots \cup s_i[A^i/A] \models I^0 \wedge \mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{i-1}, A^i, O^{i-1}, X^{i-1})$ and $s_j T_j s_{j+1}$ for all $j \in \{0,\ldots,i-1\}$.

Base case $i = 0$: Trivial because $v_{nd} \models I^0$.

Inductive case $i \geq 1$: Let $v_x \subseteq v_{nd}$ be the valuation of X^{i-1} determined by v_{nd} and let v_o be the valuation of O^{i-1} such that $v_o(o) = v_{plan}(o^{i-1})$ for every $o \in O$. By Lemma 4.40 $v_{plan} \cup v_{nd} \cup s_{i-1}[A^{i-1}/A] \cup s_i[A^i/A] \models \mathcal{R}_3(A^{i-1}, A^i, O^{i-1}, X^{i-1})$. This together with the claim of the induction hypothesis for $i-1$ establishes the first part of the claim of the hypothesis for i. By Lemma 4.39 the set E_i is one of the possible sets of active effects of T_{i-1} in s_{i-1}. Hence $s_{i-1} T_{i-1} s_i$. This finishes the induction proof.

Hence $v_{plan} \cup v_{nd} \cup v_{exec} \models I^0 \wedge \mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{n-1}, A^n, O^{n-1}, X^{n-1})$, and $v_{exec} \models G^n$ because $s_n \models G$ by assumption and $s_n[A^n/A] \subseteq v_{exec}$. □

4.7 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 4.3.1.

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

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

With partial observability, plans do not just contain operators but may also branch. With branching the sequence of operators may depend on the observations, and this makes it possible to reach goals also when no fixed sequence of operators reaches the goals. Like strong preimages in backward search correspond to images, the question arises what does branching correspond to in backward search?

Example 4.42 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 4.7. Starting from the top left, the first diagram depicts the goal belief state. The second diagram depicts the belief states obtained by computing the strong preimage of the goal belief state with respect to the move-A-onto-table action and splitting the set of states to belief states corresponding to the observational classes. The next two diagrams are similarly for strong preimages of move-B-onto-table and move-C-onto-table.

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

The last three diagrams depict the most interesting cases, constructing belief states that subsume two existing belief states in one observational class. The first diagram depicts the construction of the belief state consisting of both states in which A and B are clear and C is under either A or B. This belief state is obtained as the strong preimage of the union of two existing belief states, the one in which all blocks are on the table and the one in which A is on the table and B is on top of C. The action that moves A onto the table yields the belief state because if A is on C all blocks will be on the table and if A is already on the table nothing will happen. Construction of the belief

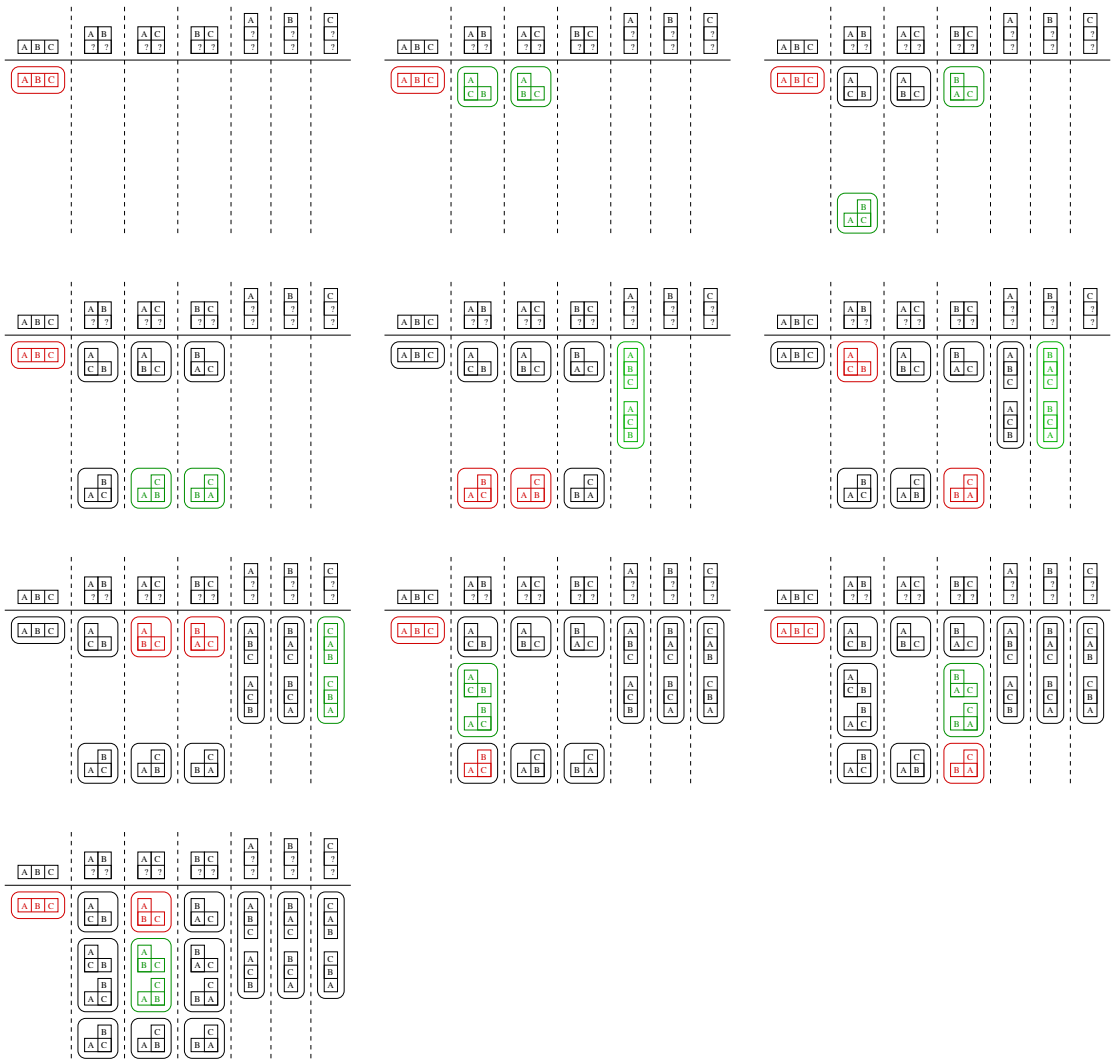


Figure 4.7: Solution of a simple blocks world problem

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 4.8 (order of construction is from the end to the beginning.)

■

We restrict to acyclic plans. Construction of cyclic plans requires looking at more global properties of transition graphs than what is needed for acyclic plans. Taking these local properties into account is difficult because we want to avoid explicit enumeration of the belief states.

4.7.1 Problem representation

Now we introduce the representation for sets of state sets for which a plan for reaching goal states exists.

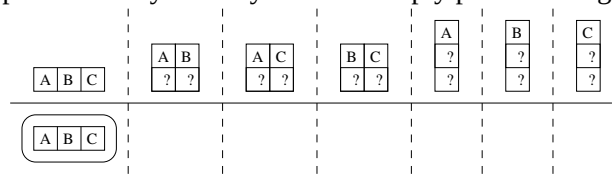
```
16:
  IF clear-A AND clear-B AND clear-C THEN GOTO end
  IF clear-A AND clear-C THEN GOTO 15
  IF clear-B AND clear-C THEN GOTO 13
  IF clear-A AND clear-B THEN GOTO 11
  IF clear-A THEN GOTO 5
  IF clear-B THEN GOTO 7
  IF clear-C THEN GOTO 9
15:
  move-C-onto-table
14:
  IF clear-A AND clear-B 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 4.8: A plan for a partially observable blocks world problem

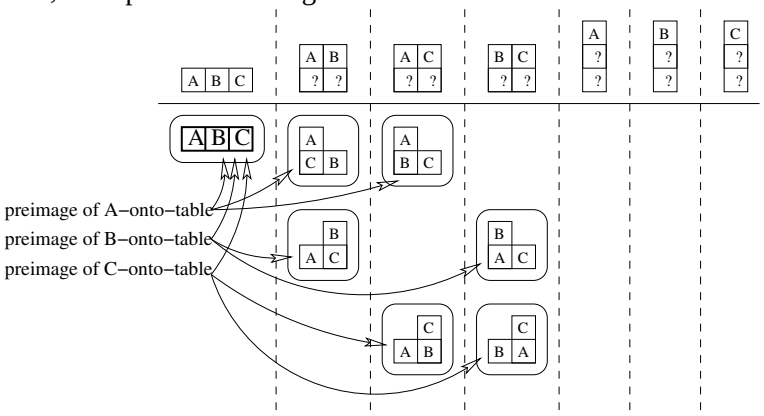
In the following example states are viewed as valuations of state variables, and the observational classes correspond to valuations of those state variables that are observable.

Example 4.43 Consider the blocks world with the state variables *clear(X)* observable, allowing to observe the topmost block of each stack. With three blocks there are 7 observational classes because there are 7 valuations of $\{\text{clear(A)}, \text{clear(B)}, \text{clear(C)}\}$ with at least one block clear.

Consider the problem of trying to reach the state in which all blocks are on the table. For each block there is an action for moving it onto the table from wherever it was before. If a block cannot be moved nothing happens. Initially we only have the empty plan for the goal states.

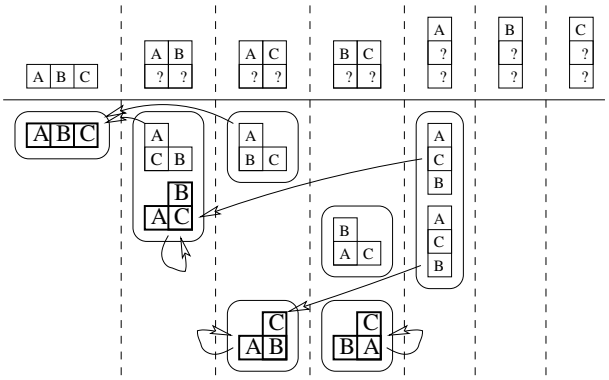


Then we compute the preimages of this set with actions that respectively put the blocks A, B and C onto the table, and split the resulting sets to the different observational classes.



Now for these 7 belief states we have a plan consisting of one or zero actions. But we also have plans for sets of states that are only represented implicitly. These involve branching. For example, we have a plan for the state set consisting of the four states in which respectively all blocks are on the table, A is on C, A is on B, and B is on A. This plan first makes observations and branches, and then executes the plan associated with the belief state obtained in each case. Because 3 observational classes each have 2 belief states, there are 2^3 maximal state sets with a branching plan. From each class only one belief state can be chosen because observations cannot distinguish between belief states in the same class.

We can find more belief states that have plans by computing preimages of existing belief states. Let us choose the belief states in which respectively all blocks are on the table, B is on C, C is on B, and C is on A, and compute their union's preimage with A-onto-table. The preimage intersected with the observational classes yields new belief states: for the class with A and B clear there is a new 2-state belief state covering both previous belief states in the class, and for the class with A clear there is a new 2-state belief state.



Computation of further preimages yields for each observational class a belief state covering all the states in that class, and hence a plan for every belief state. ■

Next we formalize the framework in detail.

Definition 4.44 (Belief space) *Let $P = (C_1, \ldots, C_n)$ be a partition of the set of all states. Then* a belief space *is an n-tuple $\langle G_1, \ldots, G_n \rangle$ where $G_i \subseteq 2^{C_i}$ for all $i \in \{1, \ldots, n\}$ and $B \not\subset B'$ for all $i \in \{1, \ldots, n\}$ and $\{B, B'\} \subseteq G_i$.*

Notice that in each component of a belief space we only have set-inclusion maximal belief states. The simplest belief spaces are obtained from sets B of states as $\mathcal{F}(B) = \langle \{C_1 \cap B\}, \ldots, \{C_n \cap B\} \rangle$. A belief space is extended as follows.

Definition 4.45 (Extension) *Let $P = (C_1, \ldots, C_n)$ be the partition of all states, $G = \langle G_1, \ldots, G_n \rangle$ a belief space, and T a set of states. Define $G \oplus T$ as $\langle G_1 \Cup (T \cap C_1), \ldots, G_n \Cup (T \cap C_n) \rangle$ where the operation $\Cup$ adds the latter set of states to the former set of sets of states and eliminates sets that are not set-inclusion maximal, defined as $U \Cup V = \{R \in U \cup \{V\} | R \not\subset K \text{ for all } K \in U \cup \{V\}\}$.*

A belief space $G = \langle G_1, \ldots, G_n \rangle$ represents the set of sets of states $\text{flat}(G) = \{B_1 \cup \cdots \cup B_n | B_i \in G_i \text{ for all } i \in \{1, \ldots, n\}\}$ and its cardinality is $|G_1| \cdot |G_2| \cdot \ldots \cdot |G_n|$.

4.7.2 Complexity of basic operations

The basic operations on belief spaces needed in planning algorithms are testing the membership of a set of states in a belief space, and finding a set of states whose preimage with respect to an action is not contained in the belief space. Next we analyze the complexity of these operations.

Theorem 4.46 *For belief spaces G and state sets B, testing whether there is $B' \in \text{flat}(G)$ such that $B \subseteq B'$, and computing $G \oplus B$ takes polynomial time.*

Proof: Idea: A linear number of set-inclusion tests suffices. □

Our algorithm for extending belief spaces by computing the preimage of a set of states (Lemma 4.48) uses exhaustive search and runs in worst-case exponential time. This asymptotic worst-case complexity is very likely the best possible because the problem is NP-hard. Our proof for this fact is a reduction from SAT: represent each clause as the set of literals that are not in it, and then a satisfying assignment is a set of literals that is not included in any of the sets, corresponding to the same question about belief spaces.

Theorem 4.47 *Testing if for belief space G there is $R \in$ flat(G) such that preimg$_o(R) \not\subseteq R'$ for all $R' \in$ flat(G) is NP-complete. This holds even for deterministic actions o.*

Proof: Membership is easy: For $G = \langle G_1, \ldots, G_n \rangle$ choose nondeterministically $R_i \in G_i$ for every $i \in \{1, \ldots, n\}$, compute $R =$ *preimg*$_o(R_1 \cup \cdots \cup R_n)$, and verify that $R \cap C_i \not\subseteq B$ for some $i \in \{1, \ldots, n\}$ and all $B \in G_i$. Each of these steps takes only polynomial time.

Let $T = \{c_1, \ldots, c_m\}$ be a set of clauses over propositions $A = \{a_1, \ldots, a_k\}$. We define a belief space based on states $\{a_1, \ldots, a_k, \hat{a}_1, \ldots, \hat{a}_k, z_1, \ldots, z_k, \hat{z}_1, \ldots, \hat{z}_k\}$. The states $\hat{a}$ represent negative literals. Define

$$\begin{aligned} c'_i &= (A \backslash c_i) \cup \{\hat{a} | a \in A, \neg a \notin c_i\} \text{ for } i \in \{1, \ldots, m\}, \\ G &= \langle \{c'_1, \ldots, c'_m\}, \{\{z_1\}, \{\hat{z}_1\}\}, \ldots, \{\{z_k\}, \{\hat{z}_k\}\} \rangle, \\ o &= \{\langle a_i, z_i \rangle | 1 \leq i \leq k\} \cup \{\langle \hat{a}_i, \hat{z}_i \rangle | 1 \leq i \leq k\}. \end{aligned}$$

We claim that T is satisfiable if and only if there is $B \in$ flat(G) such that *preimg*$_o(B) \not\subseteq B'$ for all $B' \in$ flat(G).

Assume T is satisfiable, that is, there is M such that $M \models T$. Define $M' = \{z_i | a_i \in A, M \models a_i\} \cup \{\hat{z}_i | a_i \in A, M \not\models a_i\}$. Now $M' \subseteq B$ for some $B \in$ flat(G) because from each class only one of $\{z_i\}$ or $\{\hat{z}_i\}$ is taken. Let $M'' =$ *preimg*$_o(M') = \{a_i \in A | M \models a_i\} \cup \{\hat{a}_i | a_i \in A, M \not\models a_i\}$. We show that $M'' \not\subseteq B$ for all $B \in$ flat(G). Take any $i \in \{1, \ldots, m\}$. Because $M \models c_i$, there is $a_j \in c_i \cap A$ such that $M \models a_j$ (or $\neg a_j \in c_i$, for which the proof goes similarly.) Now $z_j \in M'$, and therefore $a_j \in M''$. Also, $a_j \notin c'_j$. As there is such an a_j (or $\neg a_j$) for every $i \in \{1, \ldots, m\}$, M'' is not a subset of any c'_i, and hence $M'' \not\subseteq B$ for all $B \in$ flat(G).

Assume there is $B \in$ flat(G) such that $D =$ *preimg*$_o(B) \not\subseteq B'$ for all $B' \in$ flat(G). Now D is a subset of $A \cup \{\hat{a} | a \in A\}$ with at most one of a_i and $\hat{a}_i$ for any $i \in \{1, \ldots, k\}$. Define a model M such that for all $a \in A$, $M \models a$ if and only if $a \in D$. We show that $M \models T$. Take any $i \in \{1, \ldots, m\}$ (corresponding to a clause.) As $D \not\subseteq B$ for all $B \in$ flat(G), $D \not\subseteq c'_i$. Hence there is a_j or $\hat{a}_j$ in $D \backslash c'_i$. Consider the case with a_j ($\hat{a}_j$ goes similarly.) As $a_j \notin c'_i$, $a_j \in c_i$. By definition of M, $M \models a_j$ and hence $M \models c_i$. As this holds for all $i \in \{1, \ldots, m\}$, $M \models T$. □

4.7.3 Algorithms

Based on the problem representation in the preceding section, we devise a planning algorithm that repeatedly identifies new belief states (and associated plans) until a plan covering the initial states is found. The algorithm in Figure 4.10 tests for plan existence; further book-keeping is needed for outputting a plan. The size of the plan is proportional to the number of iterations the algorithm performs, and outputting the plan takes polynomial time in the size of the plan. The algorithm uses the subprocedure *findnew* (Figure 4.9) for extending the belief space (this is the NP-hard subproblem from Theorem 4.47). Our implementation of the subprocedure orders sets $f_1, \ldots, f_m$ by cardinality in a decreasing order: bigger belief states are tried first. We also use a simple pruning technique for deterministic actions o: If *preimg*$_o(f_i) \subseteq$ *preimg*$_o(f_j)$ for some i and j such that $i > j$, then we may ignore f_i.

Lemma 4.48 *Let H be a belief space and o an action. The procedure call findnew(o,$\emptyset$,F, H) returns a set B' of states such that $B' =$ preimg$_o(B)$ for some $B \in$ flat(F) and $B' \not\subseteq B''$ for all $B'' \in$ flat(H), and if no such belief state exists it returns $\emptyset$.*

```
1:  procedure findnew(o,A,F,H);
2:  if F = ⟨⟩ and preimg_o(A) ⊈ B for all B ∈ flat(H)
3:  then return A;
4:  if F = ⟨⟩ then return ∅;
5:  F is ⟨{f_1, ..., f_m}, F_2, ..., F_k⟩ for some k ≥ 1;
6:  for i := 1 to m do
7:     B := findnew(o,A ∪ f_i,⟨F_2, ..., F_k⟩,H);
8:     if B ≠ ∅ then return B;
9:  end;
10: return ∅;
```

Figure 4.9: Algorithm for finding new belief states

```
1:  procedure plan(I,O,G);
2:  H := F(G);
3:  progress := true;
4:  while progress and I ⊈ I' for all I' ∈ flat(H) do
5:     progress := false;
6:     for each o ∈ O do
7:        B := findnew(o,∅,H,H);
8:        if B ≠ ∅ then
9:           begin
10:             H := H ⊕ preimg_o(B);
11:             progress := true;
12:          end;
13:    end;
14: end;
15: if I ⊆ I' for some I' ∈ flat(H) then return true
16: else return false;
```

Figure 4.10: Algorithm for planning with partial observability

Proof: Sketch: The procedure goes through the elements $\langle B_1, \ldots, B_n \rangle$ of $F_1 \times \cdots \times F_n$ and tests whether $\mathit{preimg}_o(B_1 \cup \cdots \cup B_n)$ is in H. The sets $B_1 \cup \cdots \cup B_n$ are the elements of flat(F). The traversal through $F_1 \times \cdots \times F_n$ is by generating a search tree with elements of F_1 as children of the root node, elements of F_2 as children of every child of the root node, and so on, and testing whether the preimage is in H. The second parameter of the procedure represents the state set constructed so far from the belief space, the third parameter is the remaining belief space, and the last parameter is the belief space that is to be extended, that is, the new belief state may not belong to it. □

The correctness proof of the procedure *plan* consists of the following lemma and theorems. The first lemma simply says that extending a belief space H is monotonic in the sense that the members of flat(H) can only become bigger.

Lemma 4.49 *Assume T is any set of states and $B \in$ flat(H). Then there is $B' \in$ flat$(H \oplus T)$ so that $B \subseteq B'$.*

The second lemma says that if we have belief states in different observational classes such that each is included in a belief state of a belief space H, then there is a set in flat(H) that includes all these belief states.

Lemma 4.50 *Let $B_1, \ldots, B_n$ be sets of states so that for every $i \in \{1, \ldots, n\}$ there is $B'_i \in$ flat(H) such that $B_i \subseteq B'_i$, and there is no observational class C such that for some $\{i, j\} \subseteq \{1, \ldots, n\}$ both $i \neq j$ and $B_i \cap C \neq \emptyset$ and $B_j \cap C \neq \emptyset$. Then there is $B' \in$ flat(H) such that $B_1 \cup \cdots \cup B_n \subseteq B'$.*

The proof of the next theorem shows how the algorithm is capable of finding any plan by constructing it bottom up starting from the leaf nodes. The construction is based on first assigning a belief state to each node in the plan, and then showing that the algorithm reaches that belief state from the goal states by repeated computation of preimages.

Theorem 4.51 *Whenever there exists a finite acyclic plan for a problem instance, the algorithm in Figure 4.10 returns* true.

Proof: Assume that there is a plan $\langle N, b, l\rangle$ for a problem instance $\langle S, I, O, G, P\rangle$. We assume that states in S are valuations of a set of state variables. Label all nodes of the plan as follows. Each initial node n_i for $i \in \{1, \ldots, m\}$ with $\{\langle\phi_1, n_1\rangle, \ldots, \langle\phi_m, n_m\rangle\}$ we assign the label $Z(n_i) = \{s \in I | s \models \phi_i\}$.

When all parent nodes $n_1, \ldots, n_m\rangle$ of a node n have a label, we assign a label to n. Let $l(n_i) = \langle o_i, \{\langle\phi_i, n\rangle, \ldots\}\rangle$ for all $i \in \{1, \ldots, m\}$. Then $Z(n) = \bigcup_{i\in\{1,\ldots,n\}}\{s \in \mathit{img}_{o_i}(Z(n_i)) | s \models \phi_i\}$. If the above labeling does not assign anything to a node n, then assign $Z(n) = \emptyset$. Each node is labeled with exactly those states that are possible in that node on some execution.

We show that if plans for $Z(n_1), \ldots, Z(n_k)$ exist, where $n_1, \ldots, n_k$ are children of a node n, then the algorithm determines that a plan for $Z(n)$ exists as well.

Induction hypothesis: for every plan node n such that all paths from it to a terminal node have length i or less, $B = Z(n)$ is a subset of some $B' \in$ 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 4.49 there is G' such that $G \subseteq G'$ and $G' \in$ flat(H) because initially $H = \mathcal{F}(G)$ and thereafter it was repeatedly extended.

Inductive case $i \geq 1$: Let n be a plan node with $l(n) = (o, \{\langle\phi_1, n_1\rangle, \ldots, \langle\phi_k, n_k\rangle\}$.

We show that $Z(n) \subseteq B$ for some $B \in$ flat(H).

By the induction hypothesis $Z(n_i) \subseteq B$ for some $B \in$ flat(H) for all $i \in \{1, \ldots, k\}$.

For all $i \in \{1, \ldots, k\}$ $\{s \in \mathit{img}_o(Z(n)) | s \models \phi_i\} \subseteq Z(n_i)$.

Hence by Lemma 4.50 $B = \bigcup_{i\in\{1,\ldots,k\}}\{s \in \mathit{img}_o(Z(n)) | s \models \phi_i\} \subseteq B'$ for some $B' \in$ flat(H). Assume that there is no such B''. But now by Lemma 4.48 findnew(o,$\emptyset$,H,H) would return B''' such that $\mathit{preimg}_o(B''') \not\subseteq B$ for all $B \in$ flat(H), and the *while* loop could not have exited with H, contrary to our assumption about H. □

Theorem 4.52 *Let $\Pi = \langle S, I, O, G, P\rangle$ be a problem instance. If plan(I,O,G) returns* true*, then Π has a solution plan.*

Proof: Let $H_0, H_1, \ldots$ be the sequence of belief spaces H produced by the algorithm.

Induction hypothesis: For every $B \in H_{i,j}$ for some $j \in \{1,\ldots,n\}$ and $H_i = \langle H_{i,1},\ldots,H_{i,n}\rangle$ a plan reaching G exists.

Base case $i = 0$: Every component of H_0 consists of a subset of G. The empty plan reaches G.

Inductive case $i \geq 1$: H_{i+1} is obtained as $H_i \oplus \mathit{preimg}_o(B)$ where $B = \text{findnew}(o,\emptyset,H_i,H_i)$ and o is an operator.

By Lemma 4.48 $B \in \text{flat}(H_i)$. By the induction hypothesis there are plans π_i for every $B \cap C_i, i \in \{1,\ldots,n\}$. The plan that executes o followed by π_i on observation C_i reaches G from $\mathit{preimg}_o(B)$.

Let $B' \in H_{i+1,j}$ for $H_{i+1} = \langle H_{i+1,1},\ldots,H_{i+1,n}\rangle$ and some $j \in \{1,\ldots,n\}$. We show that for B' there is a plan for reaching G.

If $B' \in H_{i,j}$ then by the induction hypothesis a plan exists.

Otherwise $B' \subseteq \mathit{preimg}_o(B)$ and we can use the plan for $\mathit{preimg}_o(B)$ that first applies o and then continues with a plan associated with one of the belief states in H_i. □

It would be easy to define an algorithm that systematically generates all belief states (plans) breadth-first and therefore plans with optimal execution lengths, but this algorithm would in practice be much slower and plans would be bigger.

Above we have used only one partition of the state space to observational classes. However, it is straightforward to generalize the above definitions and algorithms to the case in which several partitions are used, each for a different set of actions. This means that the possible observations depend on the action that has last been taken.

4.8 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 deterministic 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 planning problems, the fully observable, the unobservable, and the general partially observable planning problem, showing them respectively complete for the complexity classes EXP, EXPSPACE and 2-EXP.

4.8.1 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 4.53 is by simulating polynomial-space alternating Turing 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 interesting thing in the proof is the representation of alternation. Theorem 3.59 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.[2]

[2]Restricting the proof of Theorem 4.53 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.

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 operator. 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 4.53 *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 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.

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 $s \in \Sigma \cup \{|, \Box\}$ and tape cell $i \in \{0, \ldots, p(n)\}$, and
3. h_i for the positions of the R/W head $i \in \{0, \ldots, p(n) + 1\}$.

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. $\Box_i$ for all $i \in \{n+1, \ldots, p(n)\}$.
7. $\neg \Box_i$ for all $i \in \{0, \ldots, n\}$.
8. $|_0$
9. $\neg |_i$ for all $n \in \{1, \ldots, p(n)\}$
10. h_1
11. $\neg h_i$ for all $i \in \{0, 2, 3, 4, \ldots, p(n) + 1\}$

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.[3] 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

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

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

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

That is, the transition is chosen nondeterministically.

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

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 4.53 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 4.54 *The plan existence problem for problem instances with full observability is EXP-hard.*

Proof: This is an easy extension of the proof of Theorem 4.53. 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. □

Theorem 4.55 *The problem of testing the existence of a plan for problem instances with full observability is in EXP.*

Proof: The algorithm in Section 4.4.2 runs in exponential time in the size of the problem instance. □

4.8.2 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 cannot 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 4.56 *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 4.59. 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 $\Sigma \cup \{|, \square\}$.

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

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. □

Theorem 4.57 *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 3.60 but works at the level of belief states. □

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

4.8.3 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 4.55, 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 4.58 *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 4.53 and of the EXPSPACE-hardness proof of Theorem 4.56. 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 4.59 *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 $s \in \Sigma \cup \{|, \Box\}$ 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 $\Sigma \cup \{|, \square\}$.

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

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 $\langle s, q\rangle \in (\Sigma \cup \{|, \square\}) \times Q$ 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, $\alpha = ((h = w) \rhd (\neg s \wedge s')) \wedge s'^* \wedge \neg s^*$ 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

$$\theta = \begin{cases} (h := h - 1) \wedge L \wedge \neg N \wedge \neg R & \text{if } m = L \\ N \wedge \neg L \wedge \neg R & \text{if } m = N \\ (h := h + 1) \wedge R \wedge \neg L \wedge \neg N & \text{if } m = R \end{cases}$$

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

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

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

Proof: We extend the proof of Theorem 4.59 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 4.56 and 4.53.

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 4.59 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 4.59 is replaced by $\langle p, e \wedge t \rangle$ where the test t is the conjunction of the following effects.

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

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 $2^{2^{n+1}-1}$. □

4.8.4 Polynomial size plans

We showed in Section 3.7 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 4.61 *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 NP^{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 $p_1e_1|\cdots|p_ne_n$ 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.

□

Theorem 4.62 *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.

- $A = \{x_1, \ldots, x_n, y_1, \ldots, y_m, s, g\}$
- $I = \neg x_1 \wedge \cdots \wedge \neg x_n \wedge \neg g \wedge s$
- $O = \{\langle s, x_1\rangle, \langle s, x_2\rangle, \ldots, \langle s, x_n\rangle, \langle s, \neg s \wedge (\phi \rhd g)\rangle\}$
- $G = g$

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. □

	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

4.8.5 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.

4.9 Literature

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

There is some early work on conditional planning that mostly restricts to the fully observable case and is based on partial-order planning [Etzioni *et al.*, 1992; Peot and Smith, 1992; Pryor and

Collins, 1996]. We have not discussed these algorithms because they have only been shown to solve very small problem instances.

A variant of the algorithm for constructing plans for nondeterministic planning with full observability in Section 4.4.1 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 4.4 focuses on the computation of distances of states, and plans are synthesized afterwards on the basis of the distances. We believe that our algorithms are conceptually simpler. Cimatti et al. also presented an algorithm for finding *weak plans* that may reach the goals but are not guaranteed to. However, finding weak plans is polynomially equivalent to the deterministic planning problem of Chapter 3.

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

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.