Principles of AI Planning

Strong nondeterministic planning with full observability

Malte Helmert Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

January 17th, 2007

Strong planning with full observability

We will first consider one of the simplest cases of nondeterministic planning by restricting attention to:

- fully observable planning tasks and
- strong plans.

In this lesson, planning task always means fully observable nondeterministic planning task.

Memoryless strategies
Definition

As noted previously, in the fully observable case, we can use simpler notions of strategies and plans.

Definition

Let S be the set of states of a planning task $\mathcal{T}$.
A memoryless strategy for $\mathcal{T}$ is a partial function $\pi : S \rightarrow O$ such that $\pi(s)$ is applicable wherever $\pi(s)$ is defined.

Execution of a memoryless strategy

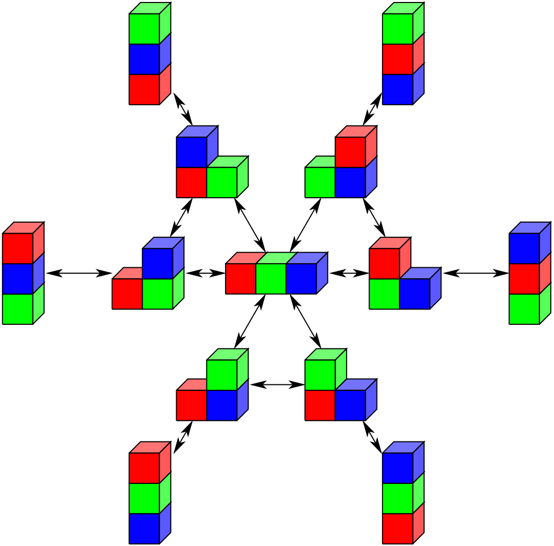
1. Determine the current state s (full observability!).
2. If $\pi(s)$ is not defined then terminate execution. (If s is a goal state, then $\pi(s)$ should not be defined so that the execution terminates.)
3. Execute action $\pi(s)$.
4. Repeat from first step.

Memoryless plans

- Memoryless strategies can be straightforwardly translated to strategies as introduced in the previous lesson.
- We do not discuss this.
- Following the definitions from the previous lesson, we can introduce concepts such as weak memoryless plans, strong memoryless plans etc.

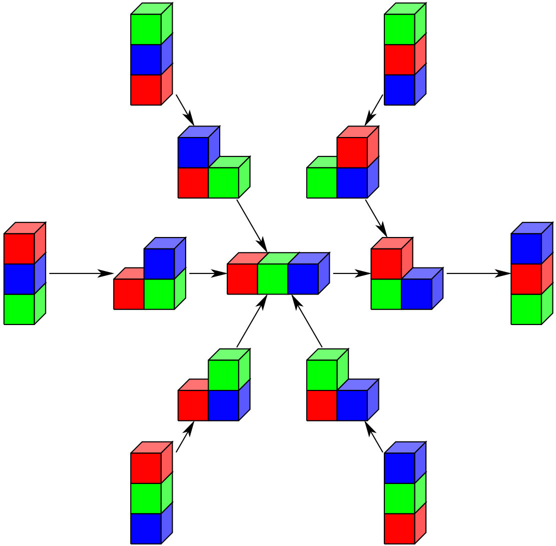
Memoryless plans

Transition system of a blocks world task



Memoryless plans

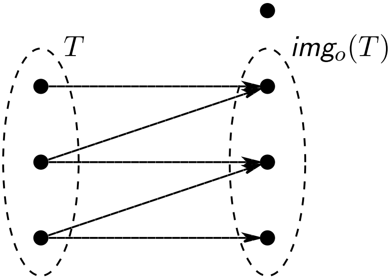
Memoryless plan (deterministic operators, uncertain initial state)



Images

Image

The image of a set T of states with respect to an operator o is the set of those states that can be reached by executing o in a state in T.



Images
Formal definition

Definition (Image of a state)

$img_o(s) = \{s' \in S | sos'\}$

Definition (Image of a set of states)

$img_o(T) = \bigcup_{s \in T} img_o(s)$

- Observe that $img_o(T) = app_o(T)$, where T is a belief state. We avoid the term "belief state" in this lesson because the intuition behind this term is wrong for fully observable planning – here, we consider sets of states together for algorithmic or efficiency reasons, not because they cannot be distinguished.

Images
Formal definition

Definition (Image of a state)

$\mathit{img}_o(s) = \{s' \in S | s o s'\}$

Definition (Image of a set of states)

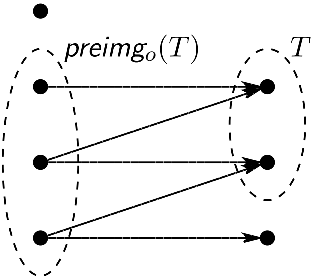
$\mathit{img}_o(T) = \bigcup_{s \in T} \mathit{img}_o(s)$

- Observe that $\mathit{img}_o(T) = \mathit{app}_o(T)$, where T is a belief state. We avoid the term "belief state" in this lesson because the intuition behind this term is wrong for fully observable planning – here, we consider sets of states together for algorithmic or efficiency reasons, not because they cannot be distinguished.

Weak preimages

Weak preimage

The weak preimage of a set T of states with respect to an operator o is the set of those states from which a state in T can be reached by executing o.



Weak preimages
Formal definition

Definition (Weak preimage of a state)

$\textit{preimg}_o(s') = \{s \in S | sos'\}$

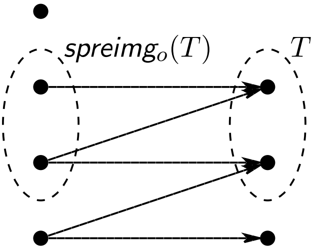
Definition (Weak preimage of a set of states)

$\textit{preimg}_o(T) = \bigcup_{s \in T} \textit{preimg}_o(s)$.

Strong preimages

Strong preimage

The strong preimage of a set T of states with respect to an operator o is the set of those states from which a state in T is always reached when executing o.



Strong preimages
Formal definition

Definition (Strong preimage of a set of states)

$$\mathit{spreimg}_o(T) = \{s \in S \mid \exists s' \in T : sos', \mathit{img}_o(s) \subseteq T\}$$

Algorithms for fully observable problems

1. Heuristic search (forward)
   Strong planning can be viewed as AND-OR search.
   OR nodes: Choice between operators
   AND nodes: Nondeterministically reached state
   Heuristic AND-OR search algorithms:
   AO*, B*, Proof Number Search, ...
2. Dynamic programming (backward)
   Compute operator/distance/value for a state based on the operators/distances/values of its all successor states.
   1. 0 actions needed for goal states.
   2. If states with i actions to goals are known, states with $\leq i + 1$ actions to goals can be easily identified.

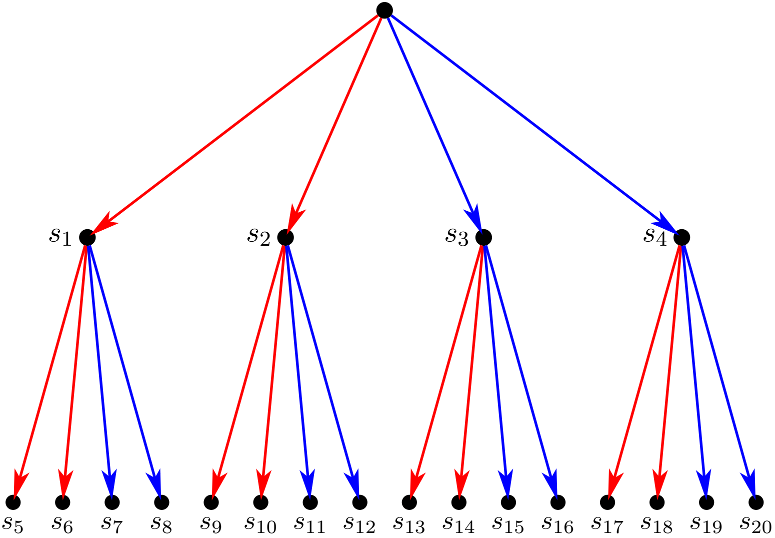
   Automatic reuse of already found plan suffixes.

Algorithms for fully observable problems

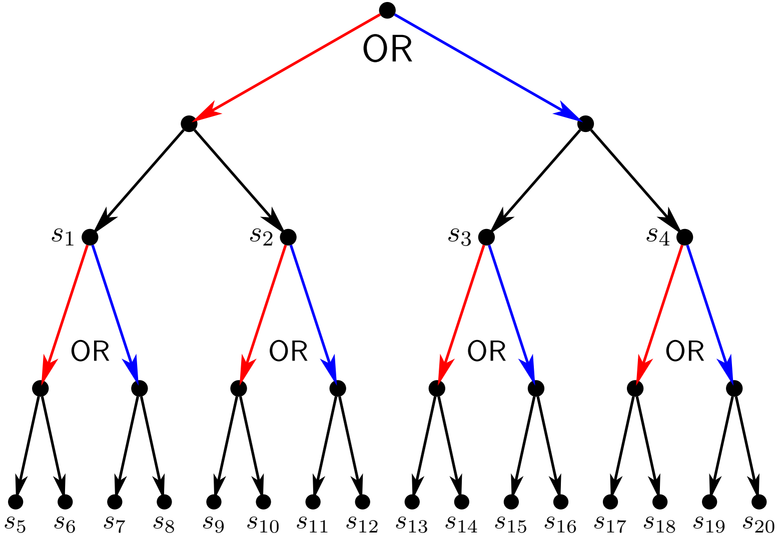
1. Heuristic search (forward)
   Strong planning can be viewed as AND-OR search.
   OR nodes: Choice between operators
   AND nodes: Nondeterministically reached state
   Heuristic AND-OR search algorithms:
   AO*, B*, Proof Number Search, ...
2. Dynamic programming (backward)
   Compute operator/distance/value for a state based on the operators/distances/values of its all successor states.
   1. 0 actions needed for goal states.
   2. If states with i actions to goals are known, states with $\leq i+1$ actions to goals can be easily identified.

   Automatic reuse of already found plan suffixes.

AND-OR search



AND-OR search



Dynamic programming

Planning by dynamic programming

If for all successors of state s with respect to operator o a plan exists, assign operator o to s.

Base case $i = 0$: In goal states there is nothing to do.

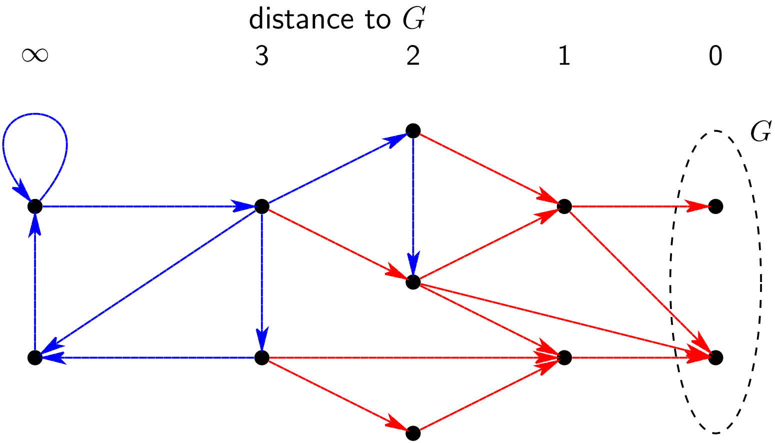
Inductive case $i \geq 1$: If there is $o \in O$ such that for all $s' \in \mathit{img}_o(s)$, the state s' is a goal state or $\pi(s')$ was assigned in an earlier iteration, then assign $\pi(s) = o$.

Backward distances

If s is assigned a value on iteration $i \geq 1$, then the backward distance of s is i.

The dynamic programming algorithm essentially computes the backward distances of states.

Backward distances
Example



Definition

Let G be a set of states and O a set of operators.

The backward distance sets D_i^{bwd} for G and O consist of those states for which there is a guarantee of reaching a state in G with at most i operator applications using operators in O:

$$D_0^{bwd} := G$$

$$D_i^{bwd} := D_{i-1}^{bwd} \cup \bigcup_{o \in O} \mathit{spreimg}_o(D_{i-1}^{bwd}) \text{ for all } i \geq 1$$

Backward distances
Definition

Definition

Let G be a 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 for G and O is

$$\delta_G^{bwd}(s) = \begin{cases} 0 & \text{if } s \in G \\ i & \text{if } s \in D_i^{bwd} \setminus D_{i-1}^{bwd} \text{ for any } i \in \mathbb{N}_1 \\ \infty & \text{otherwise} \end{cases}$$

Strong memoryless plans based on distances

Let $\mathcal{T} = \langle A, I, O, G, V \rangle$ be a planning task with state set S.

Extraction of a strong memoryless plan from distance sets

1. Let $S' \subseteq S$ be those states having a finite backward distance for G and O.
2. Let $s \in S'$ be a state with distance $i = \delta_G^{bwd}(s) \geq 1$.
3. Assign to $\pi(s)$ any operator $o \in O$ such that $img_o(s) \subseteq D_{i-1}^{bwd}$. Hence o decreases the backward distance by at least one.

Then π is a strong plan for $\mathcal{T}$ iff $\{s \in S \mid s \models I\} \subseteq S'$.

Question: What is the worst-case runtime of the algorithm?

Question: What is the best-case runtime of the algorithm if most states have a finite backward distance?

Strong memoryless plans based on distances

Let $\mathcal{T} = \langle A, I, O, G, V \rangle$ be a planning task with state set S.

Extraction of a strong memoryless plan from distance sets

1. Let $S' \subseteq S$ be those states having a finite backward distance for G and O.
2. Let $s \in S'$ be a state with distance $i = \delta_G^{bwd}(s) \geq 1$.
3. Assign to $\pi(s)$ any operator $o \in O$ such that $img_o(s) \subseteq D_{i-1}^{bwd}$. Hence o decreases the backward distance by at least one.

Then π is a strong plan for $\mathcal{T}$ iff $\{s \in S \mid s \models I\} \subseteq S'$.

Question: What is the worst-case runtime of the algorithm?

Question: What is the best-case runtime of the algorithm if most states have a finite backward distance?

Strong memoryless plans based on distances

Let $\mathcal{T} = \langle A, I, O, G, V \rangle$ be a planning task with state set S.

Extraction of a strong memoryless plan from distance sets

1. Let $S' \subseteq S$ be those states having a finite backward distance for G and O.
2. Let $s \in S'$ be a state with distance $i = \delta_G^{bwd}(s) \geq 1$.
3. Assign to $\pi(s)$ any operator $o \in O$ such that $img_o(s) \subseteq D_{i-1}^{bwd}$. Hence o decreases the backward distance by at least one.

Then π is a strong plan for $\mathcal{T}$ iff $\{s \in S \mid s \models I\} \subseteq S'$.

Question: What is the worst-case runtime of the algorithm?

Question: What is the best-case runtime of the algorithm if most states have a finite backward distance?

Making the algorithm a logic-based algorithm

- An algorithm that represents the states explicitly stops being feasible at about 10^8 or 10^9 states.
- For planning with bigger transition systems structural properties of the transition system have to be taken advantage of.
- As before, representing state sets as propositional formulae or BDDs often allows taking advantage of the structural properties: a formula or BDD that represents a set of states or a transition relation that has certain regularities may be very small in comparison to the set or relation.
- In the following, we will present a BDD-based algorithm.

Breadth-first search with progression and state sets

Reminder: Algorithm for the deterministic case

Progression breadth-first search

def bfs-progression(A, I, O, G):

  goal $:=$ *formula-to-set*(G)

  reached $:= \{I\}$

  loop:

    if *reached* $\cap$ *goal* $\neq \emptyset$:

      return solution found

    new-reached $:=$ *reached* $\cup$ *apply*$($*reached*$, O)$

    if *new-reached* $=$ *reached*:

      return no solution exists

    reached $:=$ *new-reached*

$\rightsquigarrow$ This can easily be transformed into a regression algorithm.

Breadth-first search with regression and state sets

Algorithm for the deterministic case

Regression breadth-first search

def bfs-regression(A, I, O, G):
    init $:= I$
    reached $:=$ *formula-to-set*(G)
    loop:
        if *init* $\in$ *reached*:
            return solution found
        new-reached $:=$ *reached* $\cup$ *apply*$^{-1}$(*reached*, O)
        if *new-reached* $=$ *reached*:
            return no solution exists
        reached $:=$ *new-reached*

- This algorithm is very similar to the dynamic programming algorithm for the nondeterministic case!

Breadth-first search with regression and state sets

Algorithm for the nondeterministic case

Regression breadth-first search

def bfs-regression(A, I, O, G):
    init := *formula-to-set*(I)
    reached := *formula-to-set*(G)
    loop:
        if *init* $\subseteq$ *reached*:
            return solution found
        new-reached := *reached* $\cup \bigcup_{o \in O}$ *spreimg*$_o$(*reached*)
        if *new-reached* = *reached*:
            return no solution exists
        reached := *new-reached*

- How do we define *spreimg* with set-theoretic (BDD) operations?

Computing strong preimages

Strong preimages

$$
\begin{aligned}
\mathit{spreimg}_o(T) &= \{s \in S \mid \exists s' \in T : sos', \mathit{img}_o(s) \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad \{s' \in S \mid sos'\} \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad (\forall s' \in S : sos' \rightarrow (s' \in T))\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad (\neg \exists s' \in S : sos' \wedge s' \notin T)\}
\end{aligned}
$$

Computing strong preimages

Strong preimages

$$
\begin{aligned}
\mathit{spreimg}_o(T) &= \{s \in S \mid \exists s' \in T : sos', \mathit{img}_o(s) \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad \{s' \in S \mid sos'\} \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad (\forall s' \in S : sos' \rightarrow (s' \in T))\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad (\neg\exists s' \in S : sos' \wedge s' \notin T)\}
\end{aligned}
$$

Computing strong preimages

Strong preimages

$$
\begin{aligned}
\mathit{spreimg}_o(T) &= \{s \in S \mid \exists s' \in T : sos', \mathit{img}_o(s) \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad \{s' \in S \mid sos'\} \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad (\forall s' \in S : sos' \rightarrow (s' \in T))\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad (\neg \exists s' \in S : sos' \wedge s' \notin T)\}
\end{aligned}
$$

Computing strong preimages

Strong preimages

$$\begin{aligned}
\mathit{spreimg}_o(T) &= \{s \in S \mid \exists s' \in T : sos', \mathit{img}_o(s) \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad\qquad \{s' \in S \mid sos'\} \subseteq T\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad\qquad (\forall s' \in S : sos' \rightarrow (s' \in T))\} \\
&= \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge \\
&\qquad\qquad (\neg \exists s' \in S : sos' \wedge s' \notin T)\}
\end{aligned}$$

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg\exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def rename-A-to-A'(B):
  for each $a \in A$:
    $B := \textit{bdd-rename}(B, a, a')$
  return B

def forget-A'(B):
  for each $a \in A$:
    $B := \textit{bdd-forget}(B, a')$
  return B

Computing strong preimages with BDD operations

$$\mathit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg\exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

  s'-in-T := *rename-A-to-A'*(T)

  s'-not-in-T := *bdd-complement*(*s'-in-T*)

  B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

  B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

  return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg\exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

  s'-in-T := *rename-A-to-A'*(T)

  s'-not-in-T := *bdd-complement*(*s'-in-T*)

  B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

  B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

  return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge (\neg\exists s' \in S : sos' \wedge s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):
  s'-in-T := *rename-A-to-A'*(T)
  s'-not-in-T := *bdd-complement*(*s'-in-T*)
  B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))
  B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))
  return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg \exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

s'-in-T := *rename-A-to-A'*(T)

s'-not-in-T := *bdd-complement*(*s'-in-T*)

B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge (\neg \exists s' \in S : sos' \wedge s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

s'-in-T := *rename-A-to-A'*(T)

s'-not-in-T := *bdd-complement*(*s'-in-T*)

B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\mathit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg\exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

s'-in-T := *rename-A-to-A'*(T)

s'-not-in-T := *bdd-complement*(*s'-in-T*)

B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$spreimg_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge (\neg \exists s' \in S : sos' \wedge s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

s'-in-T := *rename-A-to-A'*(T)

s'-not-in-T := *bdd-complement*(*s'-in-T*)

B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg\exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):
  s'-in-T := *rename-A-to-A'*(T)
  s'-not-in-T := *bdd-complement*(*s'-in-T*)
  B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))
  B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))
  return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \wedge s o s') \wedge (\neg\exists s' \in S : s o s' \wedge s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):
    s'-in-T := *rename-A-to-A'*(T)
    s'-not-in-T := *bdd-complement*(*s'-in-T*)
    B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))
    B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))
    return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \wedge sos') \wedge (\neg \exists s' \in S : sos' \wedge s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):
    s'-in-T := *rename-A-to-A'*(T)
    s'-not-in-T := *bdd-complement*(*s'-in-T*)
    B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))
    B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))
    return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Are we done?

Computing strong preimages with BDD operations

$$\textit{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s' \in T \land sos') \land (\neg\exists s' \in S : sos' \land s' \notin T)\}$$

Strong preimages with BDDs

def strong-preimage(o, T):

s'-in-T := *rename-A-to-A'*(T)

s'-not-in-T := *bdd-complement*(*s'-in-T*)

B_1 := *forget-A'*(*bdd-intersection*(*s'-in-T*, $T_A(o)$))

B_2 := *forget-A'*(*bdd-intersection*($T_A(o)$, *s'-not-in-T*))

return *bdd-intersection*(B_1, *bdd-complement*(B_2))

Are we done?

No, because we have not yet shown how to compute $T_A(o)$ for nondeterministic operators.

Transition formula for nondeterministic operators

The formula $\tau_A(o)$ (on which the BDD/relation $T_A(o)$ is based) must express

- the conditions for applicability of o,
- how o changes state variables, and
- which state variables o does not change.

A significant difficulty lies in the third requirement because different variables may be affected depending on nondeterministic choices.

Normal forms for nondeterministic operators

- In deterministic planning, we translated effects to normal form to express them in propositional logic.
- For nondeterministic effects, there is no (simple) normal form with all the nice properties of deterministic operator normal form:
  - expressiveness (all effects are convertible to normal form)
  - efficient computability
  - simple representation in propositional logic
- We will thus introduce different normal forms which have a subset of these properties.

Unary nondeterminism normal form
Definition

Definition

An effect e is in unary nondeterminism normal form iff

- e is deterministic and in normal form, or
- $e = e_1 \mid \ldots \mid e_n$ where each e_i is deterministic and in normal form.

- What about simple representation, expressiveness and efficient computability?

Unary nondeterminism normal form
Simple representation

Recall: $\tau_A(o)$ for deterministic operators $o = \langle c, e\rangle$

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

For $o = \langle c, e_1 \,|\, \dots \,|\, e_n\rangle$ where each e_i is deterministic:

$$\tau_A(o) = c \wedge \bigvee_{i=1}^{n} \bigwedge_{a \in A} ((\mathit{EPC}_a(e_i) \vee (a \wedge \neg \mathit{EPC}_{\neg a}(e_i))) \leftrightarrow a') \wedge \bigwedge_{i=1}^{n} \bigwedge_{a \in A} \neg(\mathit{EPC}_a(e_i) \wedge \mathit{EPC}_{\neg a}(e_i))$$

Unary nondeterminism normal form

Expressiveness and efficient computability

Unary nondeterminism normal form is expressive.

Every nondeterministic effect can be converted by using the following equivalences to raise nondeterminism to the root of the effect:

$$\begin{aligned} c \rhd (e_1 \,|\, \dots \,|\, e_n) &\equiv (c \rhd e_1) \,|\, \dots \,|(c \rhd e_n) \\ (e_1 \,|\, \dots \,|\, e_n) \wedge e' &\equiv (e_1 \wedge e') \,|\, \dots \,|(e_n \wedge e') \\ (e_1 \,|\, \dots \,|\, e_n) \,|\, e'_1 \dots \,|\, e'_m &\equiv e_1 \,|\, \dots \,|\, e_n \,|\, e'_1 \,|\, \dots \,|\, e'_m \end{aligned}$$

and then converting the deterministic subeffects using the standard algorithm.

However, this is not efficiently computable because there are operators for which an exponential growth of operator size is unavoidable ($\rightsquigarrow$ exercises).

Unary nondeterminism normal form
Expressiveness and efficient computability

Unary nondeterminism normal form is expressive.
Every nondeterministic effect can be converted by using the following equivalences to raise nondeterminism to the root of the effect:

$$\begin{aligned}
c \rhd (e_1 \,|\, \ldots \,|\, e_n) &\equiv (c \rhd e_1) \,|\, \ldots |(c \rhd e_n) \\
(e_1 \,|\, \ldots \,|\, e_n) \wedge e' &\equiv (e_1 \wedge e') \,|\, \ldots |(e_n \wedge e') \\
(e_1 \,|\, \ldots \,|\, e_n) \,|\, e'_1 \ldots \,|\, e'_m &\equiv e_1 \,|\, \ldots \,|\, e_n \,|\, e'_1 \,|\, \ldots \,|\, e'_m
\end{aligned}$$

and then converting the deterministic subeffects using the standard algorithm.

However, this is not efficiently computable because there are operators for which an exponential growth of operator size is unavoidable ($\rightsquigarrow$ exercises).

Unary nondeterminism normal form

Expressiveness and efficient computability

Unary nondeterminism normal form is expressive.

Every nondeterministic effect can be converted by using the following equivalences to raise nondeterminism to the root of the effect:

$$\begin{aligned} c \rhd (e_1 \,|\, \dots \,|\, e_n) &\equiv (c \rhd e_1) \,|\, \dots |(c \rhd e_n) \\ (e_1 \,|\, \dots \,|\, e_n) \wedge e' &\equiv (e_1 \wedge e') \,|\, \dots |(e_n \wedge e') \\ (e_1 \,|\, \dots \,|\, e_n) \,|\, e'_1 \dots \,|\, e'_m &\equiv e_1 \,|\, \dots \,|\, e_n \,|\, e'_1 \,|\, \dots \,|\, e'_m \end{aligned}$$

and then converting the deterministic subeffects using the standard algorithm.

However, this is not efficiently computable because there are operators for which an exponential growth of operator size is unavoidable ($\rightsquigarrow$ exercises).

Unary nondeterminism normal form
Discussion

- Unary nondeterminism normal form is among the simplest possible normal forms. There is only one possible nesting of effect types:
  - atomic effects
  - within conditional effects
  - within conjunctive effects
  - within choice effects
- The price for this simplicity is an exponential blow-up in many cases.
- To avoid this blowup, we will now relax the nesting options somewhat.

Unary conditionality normal form

Definition

Definition

An effect e is in unary conditionality normal form iff for all conditional effects $(c \rhd e')$ occurring within e, the effect e' is atomic.

- Note that conjunctive effects and choice effects may be nested arbitrarily.

Unary conditionality normal form is expressive.

Every nondeterministic effect can be converted by using the following equivalences to push conditional effects towards the leaves of the effect:

$$c \rhd (e_1 | \dots | e_n) \equiv (c \rhd e_1) | \dots | (c \rhd e_n)$$
$$c \rhd (e_1 \wedge \dots \wedge e_n) \equiv (c \rhd e_1) \wedge \dots \wedge (c \rhd e_n)$$
$$c \rhd (c' \rhd e) \equiv (c \wedge c') \rhd e$$

This is also efficiently computable.

However, for this normal form, there does not appear to be a simple representation in propositional logic.

Unary conditionality normal form is expressive.
Every nondeterministic effect can be converted by using the following equivalences to push conditional effects towards the leaves of the effect:

$$
\begin{aligned}
c \rhd (e_1 \,|\, \dots \,|\, e_n) &\equiv (c \rhd e_1) \,|\, \dots \,|(c \rhd e_n) \\
c \rhd (e_1 \wedge \dots \wedge e_n) &\equiv (c \rhd e_1) \wedge \dots \wedge (c \rhd e_n) \\
c \rhd (c' \rhd e) &\equiv (c \wedge c') \rhd e
\end{aligned}
$$

This is also efficiently computable.

However, for this normal form, there does not appear to be a simple representation in propositional logic.

Unary conditionality normal form is expressive.
Every nondeterministic effect can be converted by using the following equivalences to push conditional effects towards the leaves of the effect:

$$\begin{aligned} c \rhd (e_1 \,|\, \dots \,|\, e_n) &\equiv (c \rhd e_1) \,|\, \dots \,|(c \rhd e_n) \\ c \rhd (e_1 \wedge \dots \wedge e_n) &\equiv (c \rhd e_1) \wedge \dots \wedge (c \rhd e_n) \\ c \rhd (c' \rhd e) &\equiv (c \wedge c') \rhd e \end{aligned}$$

This is also efficiently computable.

However, for this normal form, there does not appear to be a simple representation in propositional logic.

Unary conditionality normal form
Discussion

- Unary conditionality normal form allows too complicated nestings of conjunctive and choice effects.
- This makes it difficult to test, for example, whether there are possible choices that will lead to inconsistent effects.
- For this reason, we will now look into a slightly stricter normal form which is a good compromise between our desiderata.

Decomposablue unary conditionality normal form
Scope

Definition

Define the scope of an effect e as

$$\begin{aligned}
\textit{scope}(a) &= \{a\} \\
\textit{scope}(\neg a) &= \{a\} \\
\textit{scope}(c \rhd e) &= \textit{scope}(e) \\
\textit{scope}(e_1 \land \dots \land e_n) &= \textit{scope}(e_1) \cup \dots \cup \textit{scope}(e_n) \\
\textit{scope}(e_1 \,|\, \dots \,|\, e_n) &= \textit{scope}(e_1) \cup \dots \cup \textit{scope}(e_n)
\end{aligned}$$

Decomposable unary conditionality normal form
Definition

Definition

An effect e is in decomposable unary conditionality (DUC) normal form iff it is in unary conditionality normal form and for all conjunctive effects $(e_1 \wedge \cdots \wedge e_n)$ occurring within e, either

- all e_i are deterministic, or
- for all $i \neq j$, *scope*(e_i) and *scope*(e_j) are disjoint.

Example: $(a \,|\, b) \wedge (\neg b \,|\, d)$ is not in DUC normal form because variable b occurs in $(a \,|\, b)$ and $(\neg b \,|\, d)$.

- Consistency of effect application can be tested easily: The effect is guaranteed to be consistent in state s iff this is the case for each deterministic sub-effect.

Decomposable unary conditionality normal form
Definition

Definition

An effect e is in decomposable unary conditionality (DUC) normal form iff it is in unary conditionality normal form and for all conjunctive effects $(e_1 \wedge \cdots \wedge e_n)$ occurring within e, either

- all e_i are deterministic, or
- for all $i \neq j$, *scope*(e_i) and *scope*(e_j) are disjoint.

Example: $(a \,|\, b) \wedge (\neg b \,|\, d)$ is not in DUC normal form because variable b occurs in $(a \,|\, b)$ and $(\neg b \,|\, d)$.

- Consistency of effect application can be tested easily: The effect is guaranteed to be consistent in state s iff this is the case for each deterministic sub-effect.

Decomposable unary conditionality normal form

Definition

Definition

An effect e is in decomposable unary conditionality (DUC) normal form iff it is in unary conditionality normal form and for all conjunctive effects $(e_1 \wedge \cdots \wedge e_n)$ occurring within e, either

- all e_i are deterministic, or
- for all $i \neq j$, *scope*(e_i) and *scope*(e_j) are disjoint.

Example: $(a \,|\, b) \wedge (\neg b \,|\, d)$ is not in DUC normal form because variable b occurs in $(a \,|\, b)$ and $(\neg b \,|\, d)$.

- Consistency of effect application can be tested easily: The effect is guaranteed to be consistent in state s iff this is the case for each deterministic sub-effect.

Decomposable unary conditionality normal form
Properties

- DUC normal form is a special case of unary conditionality normal form and a generalization of unary nondeterminism normal form.
- Because it generalizes unary nondeterminism normal form, it is expressive.
- We do not discuss efficient computability in detail, but only note that in practice, nondeterministic operators can usually be compactly represented in DUC normal form.
- We will now consider the property of simple representation.

Decomposable unary conditionality normal form

Representation in propositional logic

Recall: $\tau_A(o)$ for deterministic operators $o = \langle c, e\rangle$

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

For nondeterministic $o = \langle c, e\rangle$ where e is in DUC normal form, this generalizes to:

$$\tau_A(o) = c \wedge \tau_A^{\mathit{nd}}(e) \wedge \bigwedge_{e' \in E^{\mathit{det}}} \bigwedge_{a \in A} \neg(\mathit{EPC}_a(e') \wedge \mathit{EPC}_{\neg a}(e'))$$

where E^{det} is the set of deterministic sub-effects of e and $\tau_A^{\mathit{nd}}(e)$ is defined on the following slide.

Decomposable unary conditionality normal form
Representation in propositional logic

We make sure that $\tau_A^{nd}(e)$ describes changed and unchanged variables consistently by expressing changes

- for exactly the same variables B within choice effects and
- for disjoint variables B for (nondeterministic) conjunctive effects.

This gives rise to the following recursive definition:

Definition

$$\tau_B^{nd}(e) = \tau_B(e) \text{ for deterministic effects } e$$

$$\tau_B^{nd}(e_1 \mid \ldots \mid e_n) = \tau_B^{nd}(e_1) \vee \cdots \vee \tau_B^{nd}(e_n)$$

$$\tau_B^{nd}(e_1 \wedge \cdots \wedge e_n) = \tau_{scope(e_1)}^{nd}(e_1) \wedge \cdots \wedge \tau_{scope(e_n)}^{nd}(e_n) \wedge \bigwedge_{a \in B \setminus \bigcup_{i=1}^{n} scope(e_i)} (a \leftrightarrow a')$$

Summary
Strong planning with full observability

- We have considered the special case of nondeterministic planning where
  - planning tasks are fully observable and
  - we are interested in strong plans.
- We have introduced important concepts also relevant to other variants of nondeterministic planning such as
  - images and
  - weak and strong preimages.
- We have discussed some basic classes of algorithms:
  - forward search in AND/OR graphs, and
  - backward induction by dynamic programming.
- Finally, we have shown how to make a dynamic programming algorithm more efficient by exploiting logic- or set-based representations such as BDDs.