In this chapter, we will consider the simplest case of nondeterministic planning by restricting attention to strong plans.

Recall the definition of strong plans:

**Definition (strong plan)**
Let $S$ be the set of states of a planning task $\Pi$. Then a strong plan for $\Pi$ is a function $\pi : S_\pi \rightarrow O$ for some subset $S_\pi \subseteq S$ such that

- $\pi(s)$ is applicable in $s$ for all $s \in S_\pi$,
- $S_\pi(s_0) \subseteq S_\pi \cup S_\pi$ (\(\pi\) is closed),
- $S_\pi(s') \cap S_\pi \neq \emptyset$ for all $s' \in S_\pi(s_0)$ (\(\pi\) is proper), and
- there is no state $s' \in S_\pi(s_0)$ such that $s'$ is reachable from $s'$ following $\pi$ in a strictly positive number of steps (\(\pi\) is acyclic).
**Strong plans**

**Execution of a strong plan**
1. Determine the current state $s$.
2. If $s$ is a goal state then terminate.
3. Execute action $\pi(s)$.
4. Repeat from first step.

**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 $T$.

$$\text{img}_o(T) = \{ s' \in S \mid s \xrightarrow{o} s' \} = \text{app}_o(T)$$

**Definition (image of a state)**

$$\text{img}_o(s) = \{ s' \in S \mid s \xrightarrow{o} s' \} = \text{app}_o(s)$$

**Definition (image of a set of states)**

$$\text{img}_o(T) = \bigcup_{s \in T} \text{img}_o(s)$$
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 \( o \).

\[
\text{wpreimg}_o(T) = \{ s \in S | s \xrightarrow{o} s' \text{ and } s' \in T \}
\]

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 \( o \).

\[
\text{spreimg}_o(T) = \{ s \in S | \exists s' \in T : s \xrightarrow{o} s' \text{ and } \text{img}_o(s) \subseteq T \}
\]
**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 \( \pi(s) \) is still undefined and there is \( o \in O \) such that for all \( s' \in img_o(s) \), 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

Definition (backward distance sets)
Let $G$ be a set of states and $O$ a set of operators. The backward distance sets $D_i^{\text{bwd}}$ 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 $O$:

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

Strong plans based on distances

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a nondeterministic planning task with state set $S$ and goal states $S^\star$.

Extraction of a strong plan from distance sets

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

Then $\pi$ is a strong plan for $\mathcal{T}$ iff $I \in S'$.

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 an algorithm using a boolean-formula representation (without going into the details of how to implement it using BDDs).
**Making the algorithm a logic-based algorithm**

**Remark:** The following algorithm assumes a propositional representation of the state space as opposed to a finite-domain representation. We have already seen how to translate an FDR encoding into a propositional encoding in Chapter 9 (cf. definition of the “induced propositional planning task”). Therefore, for the rest of the present section, we will assume without loss of generality that all \( v \in V \) are propositional variables with domain \( D_v = \{0, 1\} \).

---

**Breadth-first search with progression and state sets (deterministic case)**

**Progression breadth-first search**

```python
def bfs-progression(V, I, O, γ):
    goal := formula-to-set(γ)
    reached := \{I\}
    loop:
        if reached \cap goal ≠ \emptyset:
            return solution found
        new-reached := reached ∪ \bigcup_{o ∈ O} img_o(reached)
        if new-reached = reached:
            return no solution exists
        reached := new-reached
```

This can easily be transformed into a regression algorithm.

---

**Regression breadth-first search**

```python
def bfs-regression(V, I, O, γ):
    init := I
    reached := formula-to-set(γ)
    loop:
        if init ∈ reached:
            return solution found
        new-reached := \bigcup_{o ∈ O} \text{wpreimg}_o(reached)
        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!
Transition formula for nondeterministic operators

Let $V$ be the set of state variables and $V' := \{v' \mid v \in V\}$ a set of primed copies of the variables in $V$. Intuition:

- Variables in $V$ describe the current state $s$.
- Variables in $V'$ describe the next state $s'$.

We would like to define a formula $\tau_V(o)$ that describes the transitions labeled with $o$ between states $s$ (over $V$) and $s'$ (over $V'$) in terms of $V$ and $V'$.

### $\tau_V(o)$ for deterministic operators $o = \langle \chi, e \rangle$

$$\tau_V(o) = \chi \land \bigwedge_{v \in V} ((EPC_v(e) \lor (v \land \neg EPC_{-v}(e))) \leftrightarrow v')$$

$$\land \bigwedge_{v \in V} \neg (EPC_v(e) \land EPC_{-v}(e))$$

Assume that $e = \bigwedge_{a \in A} a \land \bigwedge_{d \in D} \neg d'$ for $A = \{a_1, \ldots, a_k\}$ and $D = \{d_1, \ldots, d_l\}$ with $A \cap D = \emptyset$. Then this becomes simpler.

### $\tau_V(o)$ for STRIPS operators $o = \langle \chi; \bigwedge_{a \in A} a \land \bigwedge_{d \in D} \neg d' \rangle$

$$\tau_V(o) = \chi \land \bigwedge_{a \in A} a' \land \bigwedge_{d \in D} \neg d' \land \bigwedge_{v \in V \setminus (A \cup D)} (v \leftrightarrow v')$$

Example

Let $V = \{a, b\}$, $V' = \{a', b'\}$, and $o = \langle \neg a, \{a \land \neg b\} \rangle$. Then

$$\tau_V(o) = \neg a \land \left( (a' \land (b \leftrightarrow b')) \lor (a' \land \neg b') \right).$$

Transition formula for nondeterministic operators

The formula $\tau_V(o)$ 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.

Transition formula for nondeterministic operators

For nondeterministic operators $o = \langle \chi; \{e_1, \ldots, e_n\} \rangle$ with corresponding add and delete lists $A_i$ and $D_i$ of $e_i$ such that $A_i \cap D_j = \emptyset$, we get:

$$\tau_V(o) = \chi \land \prod_{i=1}^n \left( \bigwedge_{a \in A_i} a' \land \bigwedge_{d \in D_i} \neg d' \land \bigwedge_{v \in V \setminus (A_i \cup D_i)} (v \leftrightarrow v') \right)$$

Example

Let $V = \{a, b\}$, $V' = \{a', b'\}$, and $o = \langle \neg a, \{a \land \neg b\} \rangle$. Then

$$\tau_V(o) = \neg a \land \left( (a' \land (b \leftrightarrow b')) \lor (a' \land \neg b') \right).$$
Computing strong preimages

**Definition (substitution)**

Let $\varphi, t_1, \ldots, t_n$ be propositional formulas and $v_1, \ldots, v_n$ atomic propositions.

We denote the formula obtained from $\varphi$ by simultaneous replacement of all variables $v_i$ by the corresponding formulas $t_i$, $i = 1, \ldots, n$, by $\varphi[t_1, \ldots, t_n/v_1, \ldots, v_n]$.

**Strong preimages**

\[ \text{spreimg}_\varphi(T) = \{ s \in S \mid (\exists s' \in S : s \xrightarrow{o} s' \land s' \in T) \land \langle \forall s' \in S : s \xrightarrow{o} s' \Rightarrow s' \in T \rangle \} \]

\[ = \{ s \in S \mid (\exists s' \in S : s \xrightarrow{o} s' \land s' \in T) \land (\forall s' \in S : s \xrightarrow{o} s' \Rightarrow s' \in T) \} \]

We can use this regression formula for efficient symbolic regression search. BDDs support all necessary operations (atomic propositions, $\neg$, $\land$, $\lor$, substitution, $\exists$, $\forall$).
Example
Let $V = \{a, b\}$, $V' = \{a', b'\}$, and
$$o = (\neg a, \{a, a \land \neg b\}, \text{ i.e., } 
\tau_V(o) = \neg a \land \left((a' \land (b \leftrightarrow b')) \lor (a' \land \neg b')\right).$$
Moreover, let $\varphi = a$. Then
$$\text{spreimg}_\varphi(o) = \exists a' \exists b'. \left(\neg a \land \left((a' \land (b \leftrightarrow b')) \lor (a' \land \neg b')\right) \land 
\neg \exists a' \exists b'. \left(\neg a \land \left((a' \land (b \leftrightarrow b')) \lor (a' \land \neg b')\right) \land \neg a\right) \equiv \neg a$$

Progression Search
- We saw a generalization of regression search to strong planning.
- However, this search is uninformed (breadth-first search).
- Is there an analogue to A* search for strong planning?
- Yes: AO* search
  - Progression search (like A*)
  - Guided by a heuristic (like A*)
  - Guaranteed optimality (under certain conditions, like A*)
Progression Search

- We describe AO* on a graph representation without intermediate nodes, i.e., as in the first figure.
- There are different variants of AO*, depending on whether the graph that is being searched is an AND/OR tree, an AND/OR DAG, or a general, possibly cyclic, AND/OR graph.
- The graphs we want to search, $\mathcal{T}(\Pi)$, are in general cyclic.
- However, AO* becomes a bit more involved when dealing with cycles, so we only discuss AO* under the assumption of acyclicity and leave the generalization to cyclic state spaces as an exercise.

AO* Search

- The search is over $\mathcal{T}(\Pi)$.
- For ease of presentation, we do not distinguish between states of $\mathcal{T}(\Pi)$ and search nodes.
- Also, for ease of presentation, we do not handle the case that no strong plan exists.

AO* Search

**Definition (solution graph)**

A solution graph for a nondeterministic transition system $\mathcal{T} = \langle S, L, T, S_0, S^* \rangle$ is an acyclic subgraph of $\mathcal{T}$ (viewed as a graph), $\mathcal{T}' = \langle S', L', T' \rangle$, such that

- $S_0 \in S'$,
- for each $s' \in S' \setminus S_*$, there is exactly one label $l \in L$ s.t.
  - $T'$ contains at least one outgoing transition from $s'$ labeled with $l$,
  - $T'$ contains all outgoing transitions from $s'$ labeled with $l$ (and $S'$ contains the states reached via such transitions),
  - $T'$ contains no outgoing transitions from $s'$ labeled with any $\tilde{l} \neq l$, and
- every directed path in $\mathcal{T}'$ terminates at a goal state.
**AO* Search**

**Definition (partial solution graph)**
A partial solution graph for a nondeterministic transition system \( T = \langle S, L, T, s_0, S_\ast \rangle \) is an acyclic subgraph of \( T \) (viewed as a graph), \( T_p = \langle S_p, L, T_p, s_0, S_\ast \rangle \), s.t.
- \( S_0 \subseteq S_p \),
- for each \( s' \in S_p \) that is not an unexpanded leaf node in \( T_p \) there is exactly one outgoing transition from \( s' \) labeled with \( l \), and
- every directed path in \( T_p \) terminates at a goal state or an unexpanded leaf node in \( T_p \).

**AO* Search**

**Procedure ao-star**
```python
def ao-star(T):
    let \( T_e \) initially consist of the initial state \( s_0 \).
    while \( T_p \) has unexpanded non-goal node:
        expand unexpanded non-goal node \( s \) of \( T_p \)
        add new successor states to \( T_e \)
        for all new states \( s' \) added to \( T_e \):
            \( f(s') \leftarrow h(s') \)
        \( Z \leftarrow s \) and its ancestors in \( T_e \) along marked actions.
    while \( Z \) is not empty:
        remove from \( Z \) a state \( s \) w/o descendant in \( Z \).
        \( f(s) \leftarrow \min_o \text{applicable in } s (1 + \max_{s \rightarrow s'} f(s')) \).
        mark the best outgoing action for \( s \) (this may implicitly change \( T_p \)).
    return an optimal solution graph.
```

**Correctness (proof sketch)**
- Solution graphs directly correspond to strong plans.
- Algorithm eventually terminates (finite number of possible node expansions).
- Acyclicity guarantees that extraction of \( T_p \) and dynamic programming back-propagation of \( f \) values always terminates.
- Marking makes sure that existing solutions are eventually marked.
Details

- Pseudocode omits bookkeeping of solved states (can improve performance).
- Choice of unexpanded non-goal node of best partial solution graph is unspecified.
  - Correctness/optimality not affected.
  - One possibility: choose node with lowest cost estimate.
  - Alternative: expand several nodes simultaneously.
- Algorithm can be extended to deal with cycles in the AND/OR graph.

AO* Search

Example

[A diagram showing AO* search process]
Heuristic Evaluation Function

- **Desirable:** informative, domain-independent heuristic to initialize cost estimates.
- Heuristic should estimate (strong) goal distances.
- Heuristic does not necessarily have to be admissible (unless we seek optimal solutions).
- We can adapt many heuristics we already know from classical planning (details omitted).

Summary

- 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:
  - backward induction by dynamic programming, and
  - forward search in AND/OR graphs.