Planning with partial observability

- Combines the difficulties needed in the unobservable and fully observable cases.
- Sequential plans (like with unobservability) or state→action plans (like with full observability) do not suffice.
- In principle solvable by reduction to nondeterministic fully observable planning in the *belief space*. But this is impractical because of 2^n belief states for n states.

A simple forward search algorithm

1. Initialize the plan with node b and assign $BS(b) := I$.
2. Make b a branch node with $l(b) = \{\langle C_1, n_1\rangle, \ldots, \langle C_m, n_m\rangle\}$ where $n_1, \ldots, n_m$ are new nodes. Assign $BS(n_i) := I \cap C_i$ for all $i \in \{1, \ldots, m\}$.
3. Choose a node n with $BS(n) \not\subseteq G$ and with $l(n) = \emptyset$.

   If there is no such node, plan is complete.
4. Nondeterministically choose $o \in O$ that is applicable in B.

5. Create node n'. Assign $l(n) := \langle o, n'\rangle$ and $BS(n') :=$ *img*$_o(B)$.
6. Make n' a branch node with $l(n') = \{\langle C_1, n_1\rangle, \ldots, \langle C_m, n_m\rangle\}$ where $n_1, \ldots, n_m$ are new nodes. Assign $BS(n_i) :=$ *img*$_o(B) \cap C_i$ for every $i \in \{1, \ldots, m\}$.
7. Go to step 3.

Nondeterministic choice in step 4 is implemented as search.

Prevention of infinite plans: no node n' following n may fulfill $BS(n) \subseteq BS(n')$.

Why is forward search not good?

- Conflict between plan size and branching:
  1. If you branch (split the set of states), you quickly get a plan with an astronomic size.
  2. If you do not branch, you risk not finding a plan.

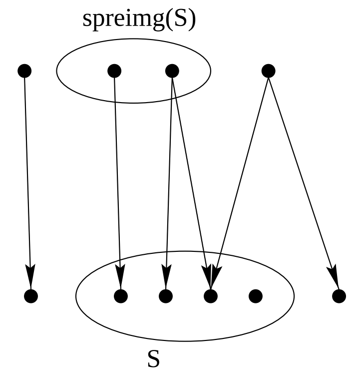
  Trying out all possible ways to branch is not feasible.

  No solutions to this problem has been presented.
- Efficient algorithms for planning with full observability use backward search (dynamic programming.)

Backward search algorithms

- Flavor similar to the backward algorithms for fully observable problems.
- Backward steps with operator applications: strong preimages.
- Backward steps with branching: we present a new construction for doing this.

Regression/preimages

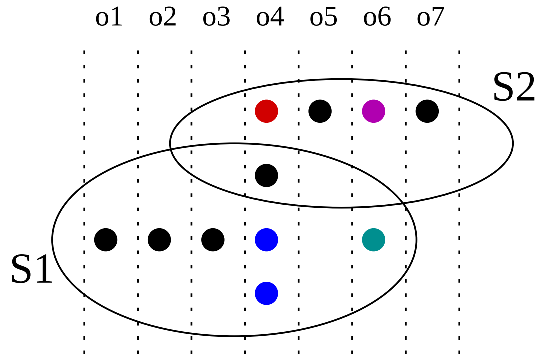


Branching in backward search

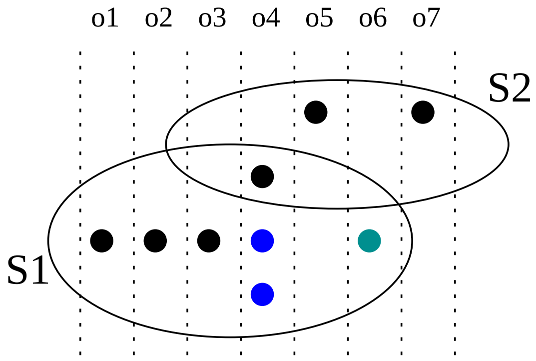
- Let the observational classes be $C_1, \dots, C_n$.
- Let $S_1, S_2, \dots, S_n$ be sets of states with plans so that for all i, j such that $i \neq j$ there is no $C \in \{C_1, \dots, C_n\}$, such that $S_i \cap C \neq \emptyset$ and $S_j \cap C \neq \emptyset$.

  Now they can be combined to $S = S_1 \cup \dots \cup S_n$ that has a plan starting with a branch.
- Where do such sets S_i come from?

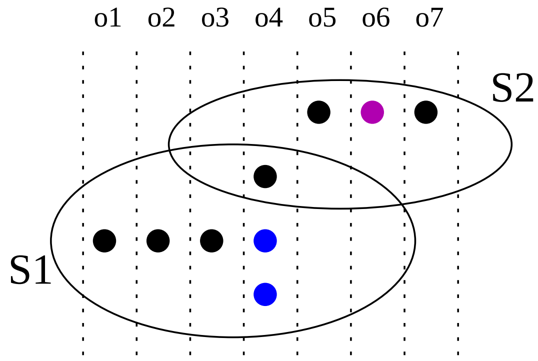
Branching



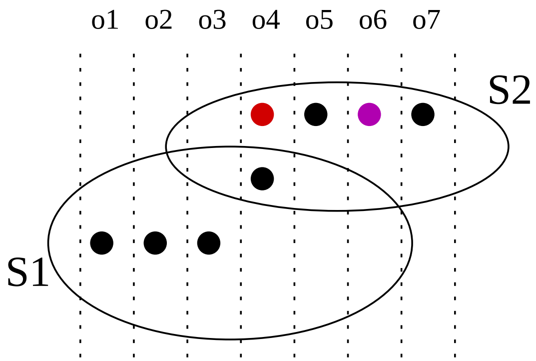
Combination 11



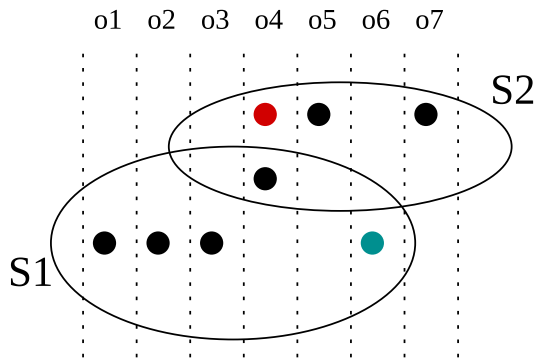
Combination 12



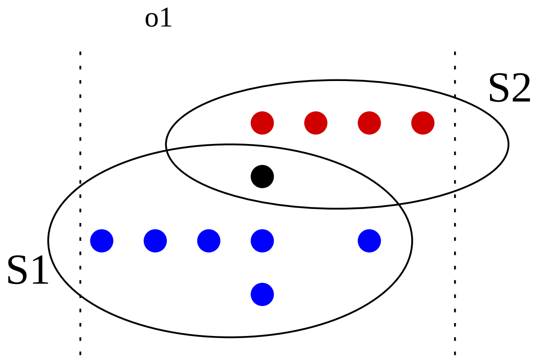
Combination 22



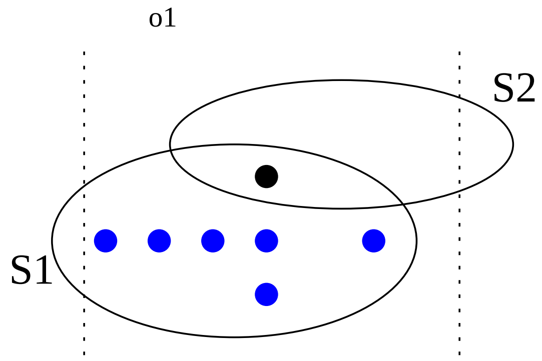
Combination 21



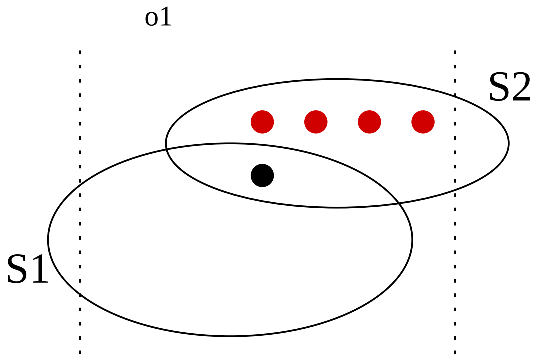
No observability ⇒ No branching



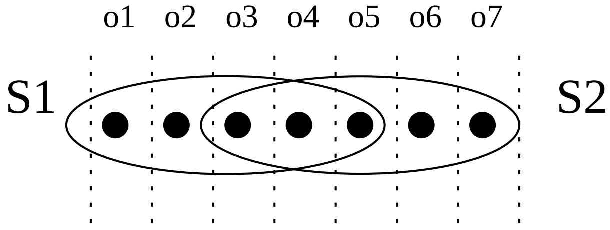
Combination 1



Combination 2



Combination with full observability



Algorithm outline

- Pick from each observational class one belief state.
- Compute the strong preimage of their union w.r.t. operator o.
- Split the resulting set of states to belief states for different observational classes.
- Objective: obtain new belief states, preferably closer to I.

Algorithm idea: construction of plans

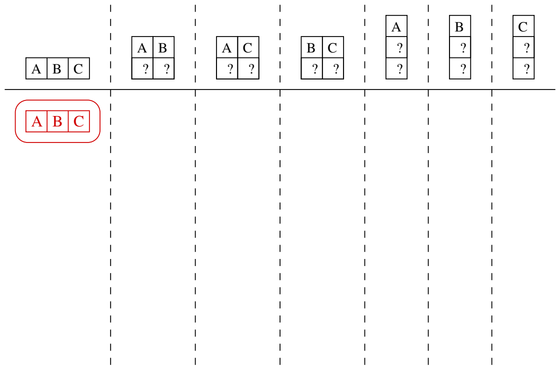
If plans for belief states $Z_1, \dots, Z_n$, respectively corresponding to observational classes $C_1, \dots, C_n$, were $\pi_1, \dots, \pi_n$, the plan for a new belief state is

1. Apply o.
2. If new current state is in C_i for $i \in \{1, \dots, n\}$, continue with π_i.

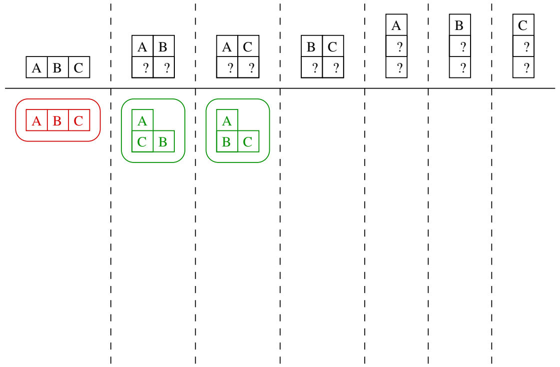
Example: backward search in the belief space

- 3 blocks A, B and C
- Goal: all blocks are on the table
- Only the variables *clear(X)* are observable.
- A block can be moved onto the table if the block is clear.
- 8 observational classes corresponding to the 8 valuations of $\{$*clear(A), clear(B), clear(C)*$\}$ (one of the valuations does not correspond to a blocks world state.)

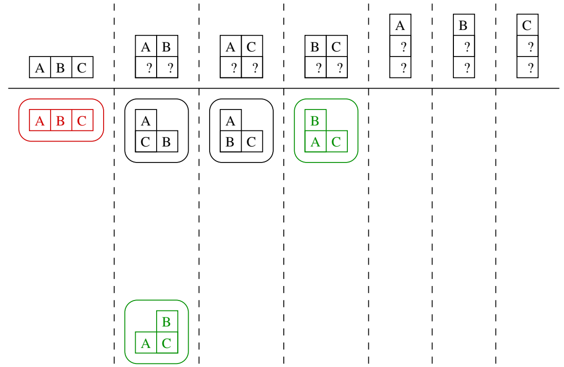
Example: goal belief state



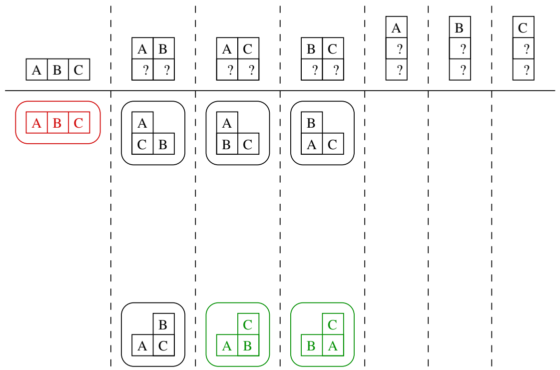
Example: backup step with A-onto-table



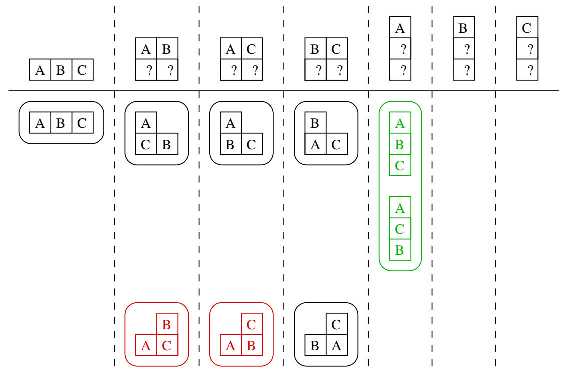
Example: backup step with B-onto-table



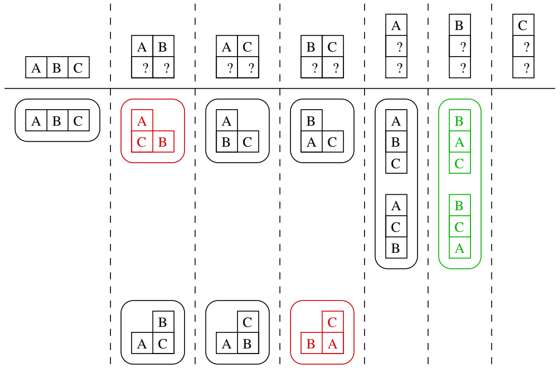
Example: backup step with C-onto-table



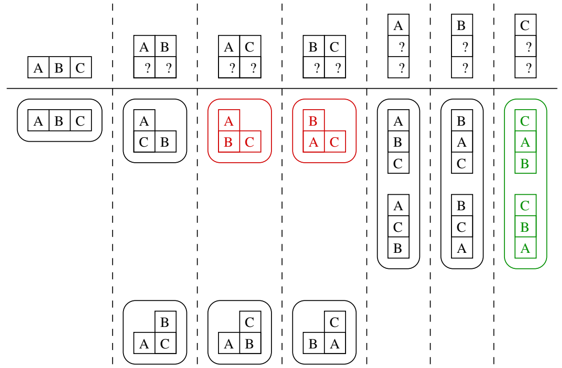
Example: backup step with A-onto-table



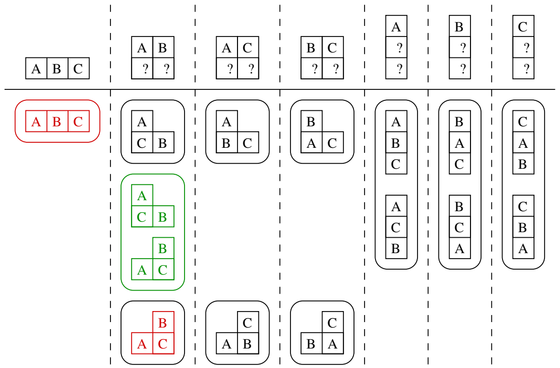
Example: backup step with B-onto-table



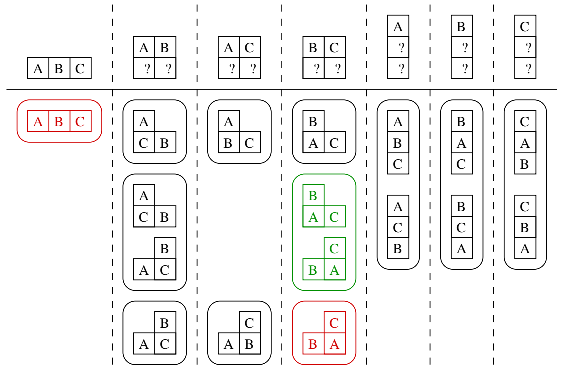
Example: backup step with C-onto-table



Example: backup step with A-onto-table



Example: backup step with B-onto-table



Example: backup step with C-onto-table