Observability and sensing (June 20, 2005)

Observability

- Robot can see and hear only the immediate surroundings.
- Poker player cannot see the opponents' cards.
- Formalization: only a subset of the state variables are observable.

Problem definition

A succinct transition system is a 5-tuple $\langle A, I, O, G, V \rangle$ consisting of

- a set A of state variables,
- a propositional formula I over A,
- a set O of operators,
- a propositional formula G over A, and
- a set $V \subseteq A$ of observable state variables.

Restrictions on observability

Let $\langle A, I, O, G, V \rangle$ be a problem instance in conditional planning.

1. If $A = V$, the problem instance is fully observable.
2. If $V = \emptyset$, the problem instance is unobservable.
3. If there are no restrictions on V then the problem instance is partially observable.

Observational classes

- When variables in $V = \{a_1, \ldots, a_m\}$ are observable and others are not then it is not possible to distinguish between states s and s' such that $s(a) = s'(a)$ for all $a \in V$.
- Let S be the set of all states (valuations of A). Observability partitions S to classes $S_1, S_2, \ldots, S_n$ of observationally indistinguishable states so that
  1. $S = S_1 \cup S_2 \cup \cdots \cup S_n$,
  2. $S_i \cap S_j = \emptyset$ for any $\{i, j\} \subset \{1, \ldots, n\}$ such that $i \neq j$.

Observational classes: cardinality

Full observability: $S = \{s_1, \ldots, s_n\}$ is partitioned to singleton classes $S_1 = \{s_1\}, S_2 = \{s_2\}, \ldots, S_n = \{s_n\}$.

Unobservability: The partition has only one class $S_1 = S$ consisting of all the n states.

Partial observability: The number of partitions and the cardinalities of S_i may be anywhere between 1 and n.

Belief states and the belief space

- Current state is not in general known during plan execution. Instead, the a set of possible current states is known.
- A set of possible current states is a belief state.
- The set of all belief states is the belief space.
- If there are n states and none of them can be observationally distinguished from another, then there are $2^n - 1$ belief states.

Planning without observability

- First we consider the second special case of planning with partial observability: planning without observability.
- Plans are sequences of actions because observations are not possible, actions cannot depend on the nondeterministic events, and hence the same actions have to be taken no matter what happens.
- Techniques needed for planning without observability can often be generalized to the general partially observable case.

The belief space

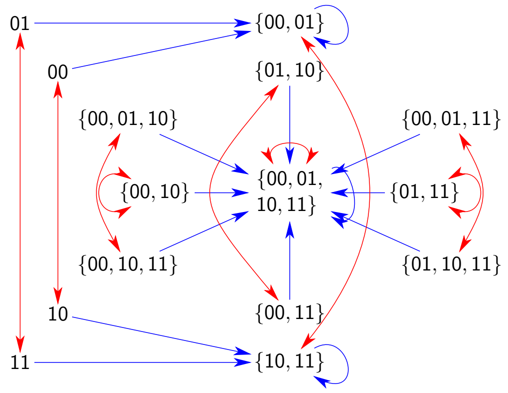
1. Let B be a belief state (a set of states).
2. Operator o is executable in B if it is executable in every $s \in B$.
3. When o is executed, possible next states are $T = img_o(B)$.
4. Observations correspond to one of the observational class, S_j.
5. New belief state is $B' = img_o(B) \cap S_j$.

The belief space
Example

Example (Next slide)

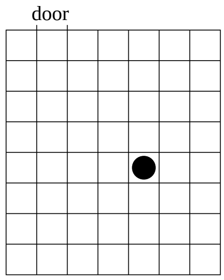
Belief space generated by states over two Boolean state variables.
$n = 2$ state variables, $2^n = 4$ states, $2^{2^n} = 16$ belief states
red action: complement the value of the first state variable
blue action: assign a random value to the second state variable

The belief space
Example

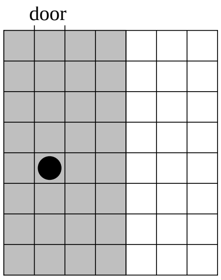


The belief space
Example

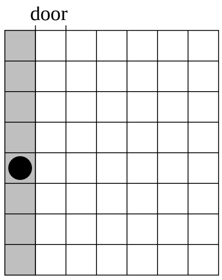
- A robot without any sensors, anywhere in a room of size 7×8.
- Actions: go North, South, East, West
- Plan for getting out: $6 \times$ West, $7 \times$ North, $1 \times$ East, $1 \times$ North
- On the next slides we depict one possible location of the robot (•) and the change in the belief state at every execution step.



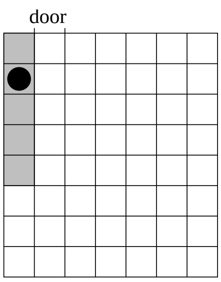
Example: after WWW



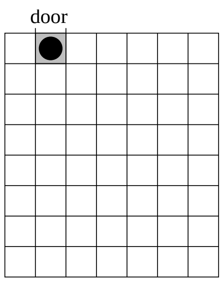
Example: after WWWWWW



Example: after WWWWWWNNN

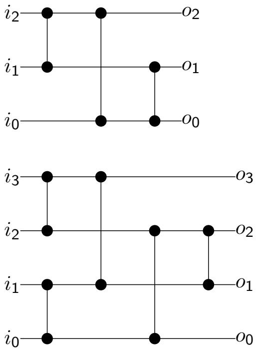


Example: after WWWWWWNNNNNNNE



The belief space
Sorting networks

Sorting networks consist of comparator-swapper elements that compare the values of two inputs and output them sorted: if first input is bigger than the second, then they are swapped, otherwise the outputs are the inputs.
A sorting network for n inputs should sort any input sequence.



The belief space
Sorting networks

Theorem
If a sorting network correctly sorts any sequence of binary digits 0 and 1, then it correctly sorts any input sequence.

3-input sorting networks can be formalized as a succinct transition system $\langle A, I, O, G, V \rangle$ where

$$
\begin{aligned}
A &= \{a_0, a_1, a_2\} \\
I &= \top \\
O &= \{o_{01}, o_{02}, o_{12}\} \\
G &= (a_0 \rightarrow a_1) \wedge (a_1 \rightarrow a_2) \\
o_{01} &= \langle \top, (a_0 \wedge \neg a_1) \rhd (\neg a_0 \wedge a_1) \rangle \\
o_{02} &= \langle \top, (a_0 \wedge \neg a_2) \rhd (\neg a_0 \wedge a_2) \rangle \\
o_{12} &= \langle \top, (a_1 \wedge \neg a_2) \rhd (\neg a_1 \wedge a_2) \rangle
\end{aligned}
$$

The belief space
Sorting networks

A plan for the 3-input sorting network is o_{12}, o_{02}, o_{01}.
The initial states are $000, 001, 010, 011, 100, 101, 110, 111$.
The goal states are $000, 001, 011, 111$
The belief state evolves as follows.

$000, 001, 010, 011, 100, 101, 110, 111$	initially
$000, 001, 010, 011, 100, 101, 110, 111$	after o_{12}
$000, 001, 010, 011, 100, 101, 110, 111$	after o_{02}
$000, 001, 010, 011, 100, 101, 110, 111$	after o_{01}

Algorithms for unobservable problems

1. Find an operator sequence $o_1, \ldots, o_n$ that reaches a state satisfying G starting from any state satisfying I.
2. o_1 must be applicable in all states $B_0 = \{s \in S | s \models I\}$ satisfying I.
   o_2 must be applicable in all states in $B_1 = img_{o_1}(B_0)$.
   o_i must be applicable in all states in $B_i = img_{o_i}(B_{i-1})$ for all $i \in \{1, \ldots, n\}$.
   Terminal states must be goal states: $B_n \subseteq \{s \in S | s \models G\}$.

Algorithms for unobservable problems

- Algorithms for deterministic planning can be lifted to the level of belief states.
- We can do forward search in the belief space with *img*$_o(B)$, backward search with *spreimg*$_o(B)$.
- We have already introduced implementation techniques that allow representing belief states B as formulae ϕ and computing images and preimages respectively as *img*$_o(\phi)$ and *spreimg*$_o(\phi)$.
- Size of belief space is exponentially bigger than the size of the corresponding state space.
  For n states there are 2^n belief states.

Algorithms for unobservable problems
Heuristic search

progression/regression + heuristic search (A$*$, IDA$*$, simulated annealing, ...)
Heuristics:

- heuristic 1: backward distances (for forward search)
- heuristic 2: cardinality of belief state (for both forward and backward search)

Algorithms for unobservable problems
Distance heuristics

Use backward distances of states as a heuristic:

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

A lower bound on plan length for belief state B is j if $B \subseteq D_j$ and $B \not\subseteq D_{j-1}$ for $j \geq 1$.
This is an admissible heuristic (does not overestimate the distance).

Algorithms for unobservable problems
Cardinality heuristics

- Backward search: Prefer operators that increase the size of the belief state, i.e. find a plan suffix that reaches a goal state from more starting states.
- Forward search: Prefer operators that decrease the size of the belief state, i.e. reduce the uncertainty about the current state and make reaching goals easier.
  For the room navigation example this heuristic works very well until the size of the belief state is 1.
- This heuristic is **not admissible**.
- Computing the cardinality of a belief state from its BDD representation takes polynomial time. (Propositional logic in general: problem is NP-hard.)

Algorithms for unobservable problems
Quantified Boolean formulas

Translation into quantified Boolean formulae (QBF)
Why not translation into propositional logic?

- We need to say that there is a plan such that ...
  This is like the satisfiability problem in CPC: there is a valuation...
- We need to say that for all executions ...
  This is like the validity problem in CPC: for all valuations...
- Consequence: the problem does not seem to be in NP nor in co-NP.

Quantified Boolean formulae
Definition

- If ϕ is a propositional formula and σ is a sequence of $\exists p$ and $\forall p$, one for every $p \in A$, then $\sigma\phi$ is a QBF.
- A formula $\exists x\phi$ is true if and only if $\phi[\top/x] \vee \phi[\bot/x]$ is true.
  (Equivalently, $\phi[\top/x]$ is true or $\phi[\bot/x]$ is true.)
- A formula $\forall x\phi$ is true if and only if $\phi[\top/x] \wedge \phi[\bot/x]$ is true.
  (Equivalently, $\phi[\top/x]$ is true and $\phi[\bot/x]$ is true.)

The most important algorithms for evaluating QBF are based on AND/OR tree search, $\forall$-variables correspond to AND-nodes and $\exists$-variables to OR-nodes.

Quantified Boolean formulae
Definition

The evaluation problem of QBF generalizes both the satisfiability and validity/tautology problems of the propositional logic. The latter are respectively NP-complete and co-NP-complete whereas the former is PSPACE-complete.

Example

The formulae $\forall x\exists y(x \leftrightarrow y)$ and $\exists x\exists y(x \wedge y)$ are true.
The formulae $\exists x\forall y(x \leftrightarrow y)$ and $\forall x\forall y(x \vee y)$ are false.

Algorithms for unobservable problems
Quantified Boolean formulas

There is a sequence of operators so that
for all initial states and nondeterministic choices
there is an execution that reaches a goal state.

$$\begin{array}{l}
\exists o_1^0 \cdots o_m^0 \cdots o_1^{t-1} \cdots o_n^{t-1} \\
\forall a_1^0 \cdots a_n^0 x_1^0 \cdots x_k^0 \cdots x_1^{t-1} \cdots x_k^{t-1} \\
\exists a_1^1 \cdots a_n^1 \cdots a_1^t \cdots a_n^t \\
I^0 \rightarrow (\mathcal{R}_3(A^0, A^1, O^0, X^0) \wedge \cdots \wedge \mathcal{R}_3(A^{t-1}, A^t, O^{t-1}, X^{t-1}) \wedge G^t)
\end{array}$$

Nondeterministic operators in CPC

- We replace nondeterministic choice by dependence of the effects on values of "hidden" state variables x.
- Nondeterministic effect $e_1|e_2|\cdots|e_n$ roughly corresponds to a number of conditional effects:

  $$(\phi_1 \rhd e_1) \wedge (\phi_2 \rhd e_2) \wedge \cdots \wedge (\phi_n \rhd e_n).$$

  Formulae ϕ_i refer to valuations of a some unknown "hidden" state variables $x_1, \ldots, x_m$ (different at every time point).
  For n choices we have $m = \lceil \log_2 n \rceil$ variables x_j.

Nondeterministic operators in CPC

- The translation $\tau_A^{nd}(o)$ of individual operators and the formulae $\tau_a^{nd}(o) \vee \cdots \vee \tau_a^{nd}(o)$ do not allow to distinguish between controllable and uncontrollable choices.
  Choice of operator is controllable, but the choice between nondeterministic alternatives is not.
- We give a new translation that distinguishes between controllability and uncontrollability.
- This translation also allows parallel operator application.

Nondeterministic operators in CPC

- We consider binary nondeterminism only so that every nondeterministic choice corresponds to the values of one propositional variable.
  Effects $a|b|c|d$ can always be equivalently represented as $(a|b)|(c|d)$.
- For n nondeterministic choices we need $\lceil \log_2 n \rceil$ auxiliary variables.
- For $(a|b)|(c|d)$ the variable x_1 chooses between $a|b$ and $c|d$.
  After $a|b$ or $c|d$ has been chosen, the respective choices between a and b, and c and d are represented by a second variable x_{11}.

Nondeterministic operators in CPC

Let e be an effect and σ a sequence of integers. Sequences σ identify nondeterministic choice inside an operator.
Define $EPC_l^{nd}(e, \sigma)$ as follows.

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

Nondeterministic operators in CPC
Example

Example

$$
\begin{aligned}
EPC^{nd}_a((a|b)|(c|d),1) &= (x_1 \wedge EPC^{nd}_a((a|b),1)) \\
&\quad \vee(\neg x_1 \wedge EPC^{nd}_a((c|d),1)) \\
&\equiv (x_1 \wedge EPC^{nd}_a((a|b),1)) \\
&\equiv (x_1 \wedge ((x_{11} \wedge EPC^{nd}_a(a,1)))) \\
&\qquad \vee(\neg x_{11} \wedge EPC^{nd}_a(b,1)) \\
&\equiv x_1 \wedge x_{11} \\
EPC^{nd}_b((a|b)|(c|d),1) &= (x_1 \wedge EPC^{nd}_b((a|b),1)) \\
&\quad \vee(\neg x_1 \wedge EPC^{nd}_b((c|d),1)) \\
&\equiv (x_1 \wedge EPC^{nd}_b((a|b),1)) \\
&\equiv (x_1 \wedge ((x_{11} \wedge EPC^{nd}_b(a,1)))) \\
&\qquad \vee(\neg x_{11} \wedge EPC^{nd}_b(b,1)) \\
&\equiv x_1 \wedge \neg x_{11}
\end{aligned}
$$

Nondeterministic operators in CPC
Example

Frame axioms

Let $e_1, \dots, e_n$ be the effects of $o_1, \dots, o_n$ respectively.

$$
\begin{aligned}
(a \wedge \neg a') &\rightarrow ((o_1 \wedge EPC^{nd}_{\neg a}(e_1,1)) \vee \cdots \vee (o_n \wedge EPC^{nd}_{\neg a}(e_n,n))) \\
(\neg a \wedge a') &\rightarrow ((o_1 \wedge EPC^{nd}_{a}(e_1,1)) \vee \cdots \vee (o_n \wedge EPC^{nd}_{a}(e_n,n)))
\end{aligned}
$$

Precondition and effect axioms

Let i be the index of operator $o = \langle c, e\rangle \in O$. The formula that describes this operator are

$$
\begin{aligned}
&(o \rightarrow c) \wedge \\
&\textstyle\bigwedge_{a\in A}(o \wedge EPC^{nd}_a(e,i) \rightarrow a') \wedge \\
&\textstyle\bigwedge_{a\in A}(o \wedge EPC^{nd}_{\neg a}(e,i) \rightarrow \neg a').
\end{aligned}
$$

Nondeterministic operators in CPC
Example

Consider the operators

$$
\begin{aligned}
o_1 &= \langle \neg a, (\overbrace{b}^{x_{11}} | \overbrace{(c \rhd d)}^{\neg x_{11}}) \wedge (\overbrace{a}^{x_{12}} | \overbrace{c}^{\neg x_{12}})\rangle \\
o_2 &= \langle \neg b, \overbrace{(\overbrace{(d \rhd b)}^{x_{21}} | \overbrace{c}^{\neg x_{21}})}^{x_2} | \overbrace{a}^{\neg x_2}\rangle
\end{aligned}
$$

var	made true by o_1 if	made true by o_2 if
a	x_{12}	$\neg x_2$
b	x_{11}	$x_2 \wedge x_{21} \wedge d$
c	$\neg x_{12}$	$x_2 \wedge \neg x_{21}$
d	$\neg x_{11} \wedge c$	

Nondeterministic operators in CPC
Example

Now $\mathcal{R}_3(\{a,b,c,d\},\{a',b',c',d'\},\{o_1,o_2\},\{x_{11},x_{12},x_2,x_{21}\})$ is the conjunction of the following formulae.

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