Lecture 6: Planning in propositional logic

- Actions as propositional formulae.
- Translations of operators into propositional formulae.
- Planning as satisfiability testing.
Actions as propositional formulae

\[ P = \{p_1, \ldots, p_n\} = \text{state variables in the current state} \]

\[ P' = \{p'_1, \ldots, p'_n\} = \text{state variables in the successor state} \]

A formula \( \phi \) over \( P \cup P' \) can be viewed as representing an action, because it can be viewed as a relation over sets of states.

For \( n \) state variables a formula (over \( 2n \) variables) represents an adjacency matrix of size \( 2^n \times 2^n \).

E.g. for \( n = 20 \), matrix size is \( 2^{1048576} \times 2^{1048576} \sim 10^{12} \) elements
Actions as propositional formulae: example

Formula \((p_1 \leftrightarrow p'_2) \land (p_2 \leftrightarrow p'_3) \land (p_3 \leftrightarrow p'_1)\) represents matrix

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Translating operators into formulae

- Any operator can be translated into a propositional formula.
- Translation is polynomial time, formula has polynomial size.
- Use in planning algorithms. Two main approaches are
  1. Translate problem instance into a formula $\phi$, find a satisfying assignment $v$, read the plan from the assignment $v$.
     = Planning as Satisfiability
  2. Use formulae as a data structure for representing sets of states, algorithm manipulates these data structures.
     e.g. BDD-based planning algorithms, (regression)
Translating operators into formulae

1. First transform operator \( o = \langle z, e \rangle \) into normal form.

2. If an atomic effect \( l \) does not occur in \( e \), add \( \bot \triangleright l \) to \( e \).

3. \( \tau_o \) is the conjunction of \( z \) and

   \[
   ((c \lor (p \land \neg c)) \leftrightarrow p') \land (\neg c \lor \neg c)
   \]

   for every state variable \( p \in P \), with \( c \triangleright p \) and \( \overline{c} \triangleright \neg p \) in \( e \).
Translating operators into formulae: example

Consider operator ⟨A ∨ B, ((B ∨ C) ▷ A) ∧ (¬C ▷ ¬A) ∧ (A ▷ B)⟩.

The corresponding propositional formula is

\[(A ∨ B) \land (((B ∨ C) ∨ (A ∧ ¬¬C)) \leftrightarrow A') \land (¬(B ∨ C) ∨ ¬¬C') \land ((A ∨ (B ∧ ¬⊥)) \leftrightarrow B') \land (¬A ∨ ¬⊥) \land ((⊥ ∨ (C ∧ ¬⊥)) \leftrightarrow C') \land (¬⊥ ∨ ¬⊥) \equiv (A ∨ B) \land (((B ∨ C) ∨ (A ∧ C)) \leftrightarrow A') \land (¬(B ∨ C) ∨ C) \land ((A ∨ B) \leftrightarrow B') \land (C \leftrightarrow C')\]
Planning as satisfiability

1. Encode operator sequences of length 0, 1, 2, ... as formulae $\phi_0, \phi_1, \phi_2, \ldots$
2. Test satisfiability of $\phi_0, \phi_1, \phi_2, \ldots$
3. Satisfiable formula corresponds to a plan.
4. Has been used with the Davis-Putnam procedure and local search algorithms for satisfiability.

This is applied in microprocessor verification / intelligent debugging: Intel, ... (Hot topic in model-checking in CAV!!! Called Bounded Model-Checking.)
Planning as satisfiability: encoding 1

Let $\langle P, I, O, G \rangle$ be a problem instance.

Let $\mathcal{R}_1(B^0, B^1)$ denote $\bigvee_{o \in O} \tau_o$ with $P = \{p_1, \ldots, p_n\}$ and $P' = \{p'_1, \ldots, p'_n\}$ replaced by $B^0 = \{b^0_1, \ldots, b^0_n\}$ and $B^1 = \{b^1_1, \ldots, b^1_n\}$.

Plans of length $n$ are encoded as

$$\iota^0 \land \mathcal{R}_1(P^0, P^1) \land \mathcal{R}_1(P^1, P^2) \land \cdots \land \mathcal{R}_1(P^{n-1}, P^n) \land G^n$$

Here $\iota^0 = \bigwedge \{p^0 | p \in P, I(p) = 1\} \cup \{\neg p^0 | p \in P, I(p) = 0\}$ and $G^n$ is $G$ with propositions $p$ replaced by $p^n$. 
Planning as satisfiability: encoding 1, example

\[ I \models A \land B, \quad G = (A \land \neg B) \lor (\neg A \land B), \]
\[ o_1 = \langle \top, (A \triangleright \neg A) \land (\neg A \triangleright A) \rangle, \quad o_2 = \langle \top, (B \triangleright \neg B) \land (\neg B \triangleright B) \rangle, \]
plan length 3

\[
\begin{align*}
(A^0 \land B^0) \\
\land ( ((A^0 \leftrightarrow A^1) \land (B^0 \leftrightarrow \neg B^1)) \lor ((A^0 \leftrightarrow \neg A^1) \land (B^0 \leftrightarrow B^1)) ) \\
\land ( ((A^1 \leftrightarrow A^2) \land (B^1 \leftrightarrow \neg B^2)) \lor ((A^1 \leftrightarrow \neg A^2) \land (B^1 \leftrightarrow B^2)) ) \\
\land ( ((A^2 \leftrightarrow A^3) \land (B^2 \leftrightarrow \neg B^3)) \lor ((A^2 \leftrightarrow \neg A^3) \land (B^2 \leftrightarrow B^3)) ) \\
\land ( (A^3 \land \neg B^3) \lor (\neg A^3 \land B^3) )
\end{align*}
\]
Planning as satisfiability: encoding 1, example

One valuation that satisfies the formula:

<table>
<thead>
<tr>
<th>time $i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A^i$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B^i$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1. There are several valuations/plans
2. Also plans of length 1 exists (just ignore time points 2 and 3!)
3. Plans of length 2 do not exist!
Planning as satisfiability: encoding 1, problems

- Many satisfiability algorithms (D-P) require transforming $\bigvee_{o \in O} \tau_o$ to CNF: this can explode the size of the formula.

  Fix: define $\mathcal{R}_1(P, P')$ as $(\bigwedge_{o \in O}(o \rightarrow \tau_o)) \land \bigvee_{o \in O} o$.

- By using propositional variables for every operator, further improvements are possible:

  Allow parallel/simultaneous application of operators: if operators do not interfere, apply them simultaneously.

  Turn off $n$ lamps: all lamps simultaneously, $n!$ not a problem.
Parallel application of operators: interference

$o$ and $o'$ interfere if $\text{app}_o(\text{app}_{o'}(s)) \neq \text{app}_{o'}(\text{app}_o(s))$ for some $s$.

- Testing interference is NP-hard (see lecture notes for proof!)

- Poly-time sufficient condition: $\langle p, e \rangle$ and $\langle p', e' \rangle$ interfere if
  - a state variable in $e$ occurs in $p'$,
  - a state variable in $e'$ occurs in $p$,
  - a state variable in $e$ occurs in $e'$,

In the rest of this lecture we use this definition of interference.
Parallel application of operators: interference, example

first enables the second  \( \langle A, B \rangle \) and \( \langle B, C \rangle \)
first disables the second  \( \langle A, \neg B \rangle \) and \( \langle B, C \rangle \)
effects are contradictory  \( \langle A, \neg B \rangle \) and \( \langle C, B \rangle \)
Planning as satisfiability: encoding 2, explanatory frame axioms in $\mathcal{R}_2(P, P')$

Let $p \in P$ be one of the state variables.

Let $c_i$ be the condition that operator $o_i \in O$ makes $p$ true.
Let $\overline{c_i}$ be the condition that operator $o_i \in O$ makes $p$ false.

(operator $o_i$ in normal form; $c_i \triangleright p$ and $\overline{c_i} \triangleright \neg p$ in the effect; or $p/\neg p$ does not occur in the effect and then $c_i = \bot/\overline{c_i} = \bot$.)

\[
\neg p \land p' \rightarrow \left( (o_1 \land c_1) \lor \cdots \lor (o_n \land c_n) \right)
\]
\[
p \land \neg p' \rightarrow \left( (o_1 \land \overline{c_1}) \lor \cdots \lor (o_n \land \overline{c_n}) \right)
\]
Planning as satisfiability: $\mathcal{R}_2(P, P')$, effect axioms

\[ o_i = \langle z, (c_1 \triangleright p_1) \wedge (\overline{c_1} \triangleright \neg p_1) \wedge \cdots \wedge (c_n \triangleright p_n) \wedge (\overline{c_n} \triangleright \neg p_n) \rangle \in O \]

may affect the state variables as follows.

\[
\begin{align*}
(o_i \land c_1) & \rightarrow p'_1 \\
(o_i \land \overline{c_1}) & \rightarrow \neg p'_1 \\
& \vdots \\
(o_i \land c_n) & \rightarrow p'_n \\
(o_i \land \overline{c_n}) & \rightarrow \neg p'_n \\
o_i & \rightarrow z
\end{align*}
\]
Planning as satisfiability: $\mathcal{R}_2(P, P')$, interference

For every $o_i, o_j \in O$, if $o_i$ and $o_j$ interfere and $i \neq j$, then

$$\neg(o_i \land o_j)$$

is included in $\mathcal{R}_2(P, P')$. 
Planning as satisfiability: $\mathcal{R}_2(P, P')$, example

$$o_1 = \langle \neg LAMP_1, LAMP_1 \rangle, \quad o_2 = \langle \neg LAMP_2, LAMP_2 \rangle$$

$$\neg LAMP_1 \land LAMP_1' \rightarrow ((o_1 \land \top) \lor (o_2 \land \bot))$$

$$(LAMP_1 \land \neg LAMP_1') \rightarrow ((o_1 \land \bot) \lor (o_2 \land \bot))$$

$$\neg LAMP_2 \land LAMP_2' \rightarrow ((o_1 \land \bot) \lor (o_2 \land \top))$$

$$LAMP_2 \land \neg LAMP_2' \rightarrow ((o_1 \land \bot) \lor (o_2 \land \bot))$$

$$o_1 \rightarrow LAMP_1'$$

$$o_1 \rightarrow \neg LAMP_1$$

$$o_2 \rightarrow LAMP_2'$$

$$o_2 \rightarrow \neg LAMP_2$$
Planning as satisfiability: encoding 2

Plans of length $n$ are encoded exactly like with $\mathcal{R}_1(P, P')$:

$$\iota^0 \land \mathcal{R}_2(P^0, P^1) \land \mathcal{R}_2(P^1, P^2) \land \cdots \land \mathcal{R}_2(P^{n-1}, P^n) \land G^n$$

Reading the plan from a satisfying assignment $\nu$:

$o_i$ is applied at time point $t$ if and only if $\nu(o_i^t) = 1$. 
Planning as satisfiability: invariants

- We can extend both $R_1(P, P')$ and $R_2(P, P')$ by adding invariants $\phi$ as formulae over $P'$.

- Invariants do not affect the number of valuations, but they help in inferring the truth-values of the propositions earlier, and thereby speed up satisfiability testing.
Planning as satisfiability: What inferences are made by DP?

The Davis-Putnam procedure:

- The most efficient systematic algorithm for satisfiability testing.

- Unit resolution in each node of the search tree: from $p$ and $\neg p \lor p_1 \lor \cdots \lor p_n$ infer $p_1 \lor \cdots \lor p_n$.

- Branching on propositions without value: one subtree with $p$ and one with $\neg p$. 
Planning as satisfiability: example

In the initial state the following are true: clear(C), on(C,B), on(B,A), ontable(A), clear(E), on(E,D), ontable(D) (So there are initially two stacks, CBA and ED.)

The goal: on(A,B) \land on(B,C) \land on(C,D) \land on(D,E)

The Davis-Putnam procedure solves the problem quickly:

- Formulae for plan lengths 1 to 4 shown unsatisfiable by unit resolution.

- Formula for plan length 5 is satisfiable: 3 search tree nodes.
Planning as satisfiability: example

v0.9 13/08/1997 19:32:47
30 propositions 100 operators
Length 1
Length 2
Length 3
Length 4
Length 5
branch on ¬clear(b)[1] depth 0
branch on clear(a)[3] depth 1
Found a plan.
  0 totable(e,d)
  1 totable(c,b) fromtable(d,e)
  2 totable(b,a) fromtable(c,d)
  3 fromtable(b,c)
  4 fromtable(a,b)
Branches 2 last 2 failed 0; time 0.0
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>clear(a)</td>
<td>FF</td>
<td>FFF</td>
<td>TT</td>
</tr>
<tr>
<td>clear(b)</td>
<td>F</td>
<td>F</td>
<td>FF</td>
</tr>
<tr>
<td>clear(c)</td>
<td>TT</td>
<td>FF</td>
<td>TTTFF</td>
</tr>
<tr>
<td>clear(d)</td>
<td>FTTFFF</td>
<td>FTTFFF</td>
<td>FTTFFF</td>
</tr>
<tr>
<td>clear(e)</td>
<td>TTTFFF</td>
<td>TTTFFF</td>
<td>TTTFFF</td>
</tr>
<tr>
<td>on(a,b)</td>
<td>FFF</td>
<td>T</td>
<td>FFFFFT</td>
</tr>
<tr>
<td>on(a,c)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(a,d)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(a,e)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(b,a)</td>
<td>TT</td>
<td>FF</td>
<td>TTTFF</td>
</tr>
<tr>
<td>on(b,c)</td>
<td>FF</td>
<td>TT</td>
<td>FFFTT</td>
</tr>
<tr>
<td>on(b,d)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(b,e)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(c,a)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(c,b)</td>
<td>T</td>
<td>FFF</td>
<td>TT</td>
</tr>
<tr>
<td>on(c,d)</td>
<td>FFFTTT</td>
<td>FFFTTT</td>
<td>FFFTTT</td>
</tr>
<tr>
<td>Event</td>
<td>Condition</td>
<td>Condition</td>
<td>Condition</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>on(c,e)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(d,a)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(d,b)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(d,c)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(d,e)</td>
<td>FFTTTT</td>
<td>FFTTTT</td>
<td>FFTTTT</td>
</tr>
<tr>
<td>on(e,a)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(e,b)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(e,c)</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>on(e,d)</td>
<td>TFFFFFFF</td>
<td>TFFFFFFF</td>
<td>TFFFFFFF</td>
</tr>
<tr>
<td>ontable(a)</td>
<td>TTT F</td>
<td>TTTTTTTF</td>
<td>TTTTTTTF</td>
</tr>
<tr>
<td>ontable(b)</td>
<td>FF FF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>ontable(c)</td>
<td>F FFF</td>
<td>FF FFF</td>
<td>FF FFF</td>
</tr>
<tr>
<td>ontable(d)</td>
<td>TFFFFFFF</td>
<td>TFFFFFFF</td>
<td>TFFFFFFF</td>
</tr>
<tr>
<td>ontable(e)</td>
<td>FTTTTTTT</td>
<td>FTTTTTT</td>
<td>FTTTTTT</td>
</tr>
</tbody>
</table>
01234
fromtable(a,b) ....T
fromtable(b,c) ...T.
fromtable(c,d) ..T..
fromtable(d,e) .T...
totable(b,a) ..T..
totable(c,b) .T...
totable(e,d) T....
**Formulae as a representation of sets of states**

<table>
<thead>
<tr>
<th>operation on sets</th>
<th>operation on formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \cup B$</td>
<td>$A \lor B$</td>
</tr>
<tr>
<td>$A \cap B$</td>
<td>$A \land B$</td>
</tr>
<tr>
<td>$A \setminus B$</td>
<td>$A \land \neg B$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>question about sets</th>
<th>question about formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \subseteq B?$</td>
<td>$\models A \rightarrow B?$</td>
</tr>
<tr>
<td>$A \subset B?$</td>
<td>$\models A \rightarrow B$ and $\not\models B \rightarrow A$?</td>
</tr>
<tr>
<td>$A = B?$</td>
<td>$\models A \leftrightarrow B?$</td>
</tr>
</tbody>
</table>
Formulae as a representation of sets of states

• Many algorithms for deterministic/nondeterministic planning use formulae as the main data structure.

• Type of formulae that are used has a strong impact on the efficiency.
  1. arbitrary propositional formulae
  2. Boolean circuits (with sharing of subformulae)
  3. formulae in normal form like DNF, CNF, DNNF
  4. binary decision diagrams
## Normal forms for propositional formulae

<table>
<thead>
<tr>
<th></th>
<th>$\lor$</th>
<th>$\land$</th>
<th>$\neg$</th>
<th>$\phi \in \text{TAUT?}$</th>
<th>$\phi \in \text{SAT?}$</th>
<th>$\phi \equiv \phi'$?</th>
</tr>
</thead>
<tbody>
<tr>
<td>circuits</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>co-NP-hard</td>
<td>NP-hard</td>
<td>co-NP-hard</td>
</tr>
<tr>
<td>formulae</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>co-NP-hard</td>
<td>NP-hard</td>
<td>co-NP-hard</td>
</tr>
<tr>
<td>DNF</td>
<td>P</td>
<td>exp</td>
<td>exp</td>
<td>co-NP-hard</td>
<td>P</td>
<td>co-NP-hard</td>
</tr>
<tr>
<td>CNF</td>
<td>exp</td>
<td>P</td>
<td>exp</td>
<td>P</td>
<td>NP-hard</td>
<td>co-NP-hard</td>
</tr>
<tr>
<td>DNNF</td>
<td>P</td>
<td>exp</td>
<td>exp</td>
<td>co-NP-hard</td>
<td>P</td>
<td>co-NP-hard</td>
</tr>
<tr>
<td>BDD</td>
<td>exp</td>
<td>exp</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>