Course outline: Principles of AI Planning

lecturer: Dr. Jussi Rintanen  
email: rintanen@informatik.uni-freiburg.de  
web page: http://www.informatik.uni-freiburg.de/~ki/lehre/ss04/aip/  
time: Mondays 2pm to 4pm, Wednesday 2pm to 3pm + exercises  
lecture hall: SR 00-010/14, Building 101  
textbook: No. Lecture notes available from web page.  
language: English and German  
exam: Wednesday July 21st ??? (to be decided later)  
grade: 0.85 × exam + 0.15 × exercises

What is the course about?

Different variants of planning:
- deterministic vs. nondeterministic actions
- full observability vs. partial observability

objectives:
- plans with success probability 1.0
- plans with maximum expected success probability
- plans with maximum expected rewards

Algorithms for deterministic planning:
- progression, regression
- heuristic search
- translation to propositional logic
- other approaches (e.g. partial-order planning)
- pruning techniques: e.g. symmetry

Jussi Rintanen  
April 21, AI Planning  
1/46

Jussi Rintanen  
April 21, AI Planning  
2/46

Jussi Rintanen  
April 21, AI Planning  
3/46

Jussi Rintanen  
April 21, AI Planning  
4/46
What is the course about?

Algorithms for nondeterministic planning:

- conditional planning
- iterative algorithms for probabilistic planning (MDPs)
- extension of the techniques to very big state spaces with binary decision diagrams and related data structures.
- partial observability

Contents of the first lectures

1. transition systems
2. reachability in transition systems in terms of matrices (basis for BDD-based techniques that are discussed later)
3. representation of states in terms of state variables
4. operators
5. the standard input language for planners PDDL

Transition systems

- Model the dynamics of the world/system/application.
- Are formalized as $\langle S, \{a_1, \ldots, a_n\}\rangle$ where
  - $S$ is a finite set of states,
  - every action $a_i \subseteq S \times S$ is a binary relation on $S$.
- First we restrict to $a_i$ that are (partial) functions from $S$ to $S$: for every $s \in S$, $(s, s') \in a_i$ for at most one $s' \in S$. 

**Actions as matrices**

1. If there are \( n \) states, each action corresponds to a \( n \times n \) matrix:
   - Element at row \( i \) and column \( j \) is 1 if the action maps state \( i \) to state \( j \).
   - For deterministic actions there is at most one non-zero element in each row.

2. Matrix multiplication corresponds to sequential composition:
   - Taking action \( M_1 \) followed by action \( M_2 \) is the product \( M_1M_2 \).
   - (This is also the relational product of the associated relations.)

3. The unit matrix \( I_{n \times n} \) is the NO-OP action.
**Sum matrix** \( M_R + M_G + M_B \)

\[
\begin{array}{cccccc}
A & B & C & D & E & F \\
0 & 1 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 & 1 \\
C & 0 & 1 & 1 & 0 & 0 & 1 \\
D & 1 & 0 & 1 & 0 & 1 & 0 \\
E & 0 & 1 & 0 & 1 & 0 & 0 \\
F & 1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

We use addition \( 0 + 0 = 0 \) and \( b + b' = 1 \) if \( b = 1 \) or \( b' = 1 \).

**Sequential composition as matrix multiplication**

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
\end{pmatrix}
\times
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 \\
\end{pmatrix}
= 
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 \\
\end{pmatrix}
\]

E is reachable from B by two actions, because F is reachable from B by one action and E is reachable from F by one action.

**Reachability**

Let \( M \) be the \( n \times n \) matrix that is the (Boolean) sum of the matrices of the individual actions. Define

\[
R_0 = I_{n \times n},
R_1 = I_{n \times n} + M,
R_2 = I_{n \times n} + M + M^2,
R_3 = I_{n \times n} + M + M^2 + M^3,
\]

\( R_i \) represents reachability by \( i \) actions or less. If \( s' \) is reachable from \( s \), then it is reachable with \( \leq n \) actions: \( R_n = R_{n+1} \).

**Reachability: example, \( M_R \)**

\[
\begin{array}{cccccc}
A & B & C & D & E & F \\
0 & 1 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 & 1 \\
C & 0 & 0 & 1 & 0 & 0 & 0 \\
D & 0 & 0 & 1 & 0 & 0 & 0 \\
E & 0 & 1 & 0 & 0 & 0 & 0 \\
F & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]
Reachability: example, $M_R + M_R^2$

Reachability: example, $M_R + M_R^2 + M_R^3$

Reachability: example, $M_R + M_R^2 + M_R^3 + I_{6\times 6}$

Reachability: row vectors are sets of states

Row vectors $S$ represent sets.

$SM$ is the set of states reachable from $S$ by $M$.
A simple planning algorithm

1. Compute the matrices $R_0, R_1, R_2, \ldots, R_n$.

2. Find the smallest $i$ such that a goal state $s_g$ is reachable from the initial state according to $R_i$.

3. Find an action (the last action of the plan) by which $s_g$ is reached with one step from a state $s_g'$ that is reachable from the initial state according to $R_{i-1}$.

4. Repeatedly proceed backward toward the goal from $s_g'$.

Example

![Graphical representation of a simple planning algorithm]

State variables

- The state of the world is described in terms of a finite set of finite-valued state variables.
- Example: HOUR : $\{0, \ldots, 23\} = 13$, MINUTE : $\{0, \ldots, 59\} = 55$, LOCATION : $\{51, 52, 82, 101, 102\} = 101$, WEATHER : $\{\text{sunny, cloudy, rainy}\} = \text{cloudy}$, HOLIDAY : $\{T, F\} = F$
- Any $n$-valued state variable can be replaced by $\lfloor \log_2 n \rfloor$ Boolean (2-valued) state variables.
- Actions change the values of the state variables.
Example: blocks world with state variables

State variables:
LOCATION-OF-A : \{B, C, TABLE\}
LOCATION-OF-B : \{A, C, TABLE\}
LOCATION-OF-C : \{A, B, TABLE\}

Not all valuations correspond to an intended blocks world state:
e.g. A-ON-B and B-ON-A should not be simultaneously true.

Example: blocks world with Boolean state variables

Boolean state variables:
A-ON-B  A-ON-C  A-ON-TABLE
B-ON-A  B-ON-C  B-ON-TABLE
C-ON-A  C-ON-B  C-ON-TABLE

E.g. A-ON-B and B-ON-A should not be simultaneously true,
and only one state variable of the form x-ON-y for any x, and for
any y except TABLE, should be true at a time.

Logical representations of state spaces

- n state variables with m values induce a state space consisting of \( m^n \) states (\( 2^n \) states for n Boolean state variables).
- A language for talking about sets of states (valuations of state variables) is the propositional logic.
- Logical operators correspond to set-theoretical operators.
- Logical relations on formulae correspond to relations between sets.

Propositional logic

Let \( P \) be a set of atomic propositions (\( \sim \) state variables.)

1. For all \( p \in P \), \( p \) is a propositional formula.
2. If \( \phi \) is a propositional formula, then so is \( \sim \phi \).
3. If \( \phi \) and \( \phi' \) are propositional formulae, then so is \( \phi \lor \phi' \).
4. If \( \phi \) and \( \phi' \) are propositional formulae, then so is \( \phi \land \phi' \).
5. The symbols \( \bot \) and \( \top \) are propositional formulae.

The implication \( \phi \rightarrow \phi' \) is an abbreviation for \( \sim \phi \lor \phi' \).
The equivalence \( \phi \leftrightarrow \phi' \) is an abbreviation for \( (\phi \rightarrow \phi') \land (\phi' \rightarrow \phi) \).
A valuation of $P$ is a function $v : P \rightarrow \{0, 1\}$. Define

1. $v \models p$ if and only if $v(p) = 1$, for $p \in P$.
2. $v \models \neg \phi$ if and only if $v \nvdash \phi$.
3. $v \models \phi \lor \phi'$ if and only if $v \models \phi$ or $v \models \phi'$.
4. $v \models \phi \land \phi'$ if and only if $v \models \phi$ and $v \models \phi'$.
5. $v \models T$.
6. $v \nvdash \bot$.

A propositional formula $\phi$ is *satisfiable* if there is at least one valuation $v$ so that $v \models \phi$. Otherwise it is *unsatisfiable*.

A propositional formula $\phi$ is *valid* or a *tautology* if $v \models \phi$ for all valuations $v$. We write this as $\models \phi$.

A propositional formula $\phi$ is a *logical consequence* of a propositional formula $\phi'$, written $\phi' \models \phi$, if $v \models \phi$ for all valuations $v$ such that $v \models \phi'$.

A propositional formula that is a proposition $p$ or a negated proposition $\neg p$ for some $p \in P$ is a *literal*.

A formula that is a disjunction of literals is a *clause*.

---

<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 of states</th>
<th>question about formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \subseteq B$?</td>
<td>$A \models B$?</td>
</tr>
<tr>
<td>$A \subset B$?</td>
<td>$A \models B$ and $B \nvdash A$?</td>
</tr>
<tr>
<td>$A = B$?</td>
<td>$A \models B$ and $B \models A$?</td>
</tr>
<tr>
<td>$\bot$</td>
<td>the empty set</td>
</tr>
<tr>
<td>$T$</td>
<td>the universal set</td>
</tr>
</tbody>
</table>

Operators

Actions are represented as *operators* $(c, e)$.

$c$ (the *precondition*) is a propositional formula over $P$ describing the set of states in which the action can be taken. *(States in which an arrow starts.)*

$e$ (the *effect*) describes the successor states of states in which the action can be taken. *(Where do the arrows go.)*

The description is procedural: how are the values of the state variable changed?
Operators: effects

Atomic effects are of the form $p := r$ for $p \in P$. For Boolean state variables we always write $p$ for $p := 1$ and $\neg p$ for $p := 0$.

Effects are then recursively defined as follows.

1. $p$ and $\neg p$ for state variables $p \in P$ are effects.
2. $e_1 \wedge \cdots \wedge e_n$ is an effect if $e_1, \ldots, e_n$ are effects (the special case with $n = 0$ is the empty conjunction $\top$.)
3. $c \triangleright e$ is an effect if $c$ is a formula over $P$ and $e$ is an effect.

Example: operators for blocks world

For convenience we use also state variables $A$-CLEAR, $B$-CLEAR, and $C$-CLEAR.

\[
\begin{align*}
\langle A$-CLEAR$ \wedge A$-ON-TABLE$ \wedge B$-CLEAR$ & \rangle, \langle A$-ON-B$ \wedge \neg A$-ON-TABLE$ \wedge \neg B$-CLEAR$ \rangle, \\
\langle A$-CLEAR$ \wedge A$-ON-TABLE$ \wedge C$-CLEAR$ & \rangle, \langle A$-ON-C$ \wedge \neg A$-ON-TABLE$ \wedge \neg C$-CLEAR$ \rangle, \\
\langle A$-CLEAR$ \wedge (A$-ON-B$ \vee A$-ON-C$) & \rangle, \langle A$-ON-TABLE$ \wedge \neg A$-ON-B$ \wedge \neg A$-ON-C$ \rangle, \\
\langle B$-CLEAR$ \wedge (B$-ON-A$ \vee B$-ON-C$) & \rangle, \langle B$-ON-TABLE$ \wedge \neg B$-ON-A$ \wedge \neg B$-ON-C$ \rangle.
\end{align*}
\]

Operators: changes caused by the operator

Operator $\langle c, e \rangle$ is applicable in a state $s$ iff $s \models c$.

Assign each effect $e$ a set $[e]_s$ of literals $p$ and $\neg p$ for $p \in P$.

1. $[p]_s = \{p\}$ and $[\neg p]_s = \{\neg p\}$ for $p \in P$.
2. $[e_1 \wedge \cdots \wedge e_n]_s = [e_1]_s \cup \ldots \cup [e_n]_s$.
3. $[c' \triangleright e]_s = [e]_s$ if $s \models c'$ and $[c' \triangleright e]_s = \emptyset$ otherwise.
Operators: the successor state of a state

The successor $\text{app}_o(s)$ of $s$ with respect to operator $o = (c, e)$ is obtained from $s$ by making literals $[e]_s$ true.

EXAMPLE: Consider the operator $(a, e)$ where $e = \neg a \land (\neg c \supset \neg b)$ and a state $s$ such that $s \models a \land b \land c$.

The operator is applicable because $s \models a$.

Now $[e]_s = \{\neg a\}$ and $\text{app}_{(a, e)}(s) \models \neg a \land b \land c$.

Schematic operators

- Description of state variables and operators in terms of a given set of objects.
- Analogy: propositional logic vs. predicate logic
- Planners take input as schematic operators, and translate them to (ground) operators. This is called grounding.

Schematic operators: example

Schematic operator $(\text{in}(x, y_1), \text{in}(x, y_2) \land \neg \text{in}(x, y_1))$ with

$x \in \{\text{car1, car2}\}$

$x, t_1$ and $t_2$ taking values

$y_1 \in \{\text{Freiburg, Strassburg}\}$,

$y_2 \in \{\text{Freiburg, Strassburg}\}, y_1 \neq y_2$

corresponds to a set of operators:

\{ $(\text{in}(\text{car1, Freiburg}), \text{in}(\text{car1, Strassburg}) \land \neg \text{in}(\text{car1, Freiburg}))$, $(\text{in}(\text{car1, Strassburg}), \text{in}(\text{car1, Freiburg}) \land \neg \text{in}(\text{car1, Strassburg}))$, $(\text{in}(\text{car2, Freiburg}), \text{in}(\text{car2, Strassburg}) \land \neg \text{in}(\text{car2, Freiburg}))$, $(\text{in}(\text{car2, Strassburg}), \text{in}(\text{car2, Freiburg}) \land \neg \text{in}(\text{car2, Strassburg}))$ \}
Schematic operators: quantification

existential quantification: finite disjunctions (not for effects)

universal quantification: finite conjunctions

EXAMPLE:
\[ \exists x \in \{A, B, C\} \text{in}(x, \text{Freiburg}) \text{ is a short-hand for } \text{in}(A, \text{Freiburg}) \lor \text{in}(B, \text{Freiburg}) \lor \text{in}(C, \text{Freiburg}). \]

Example: blocks world in PDDL

```lisp
(define (domain BLOCKS)
  (:requirements :adl :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
               (ontable ?x - block)
               (clear ?x - block))

  (:action fromtable
    :parameters (?x - block ?y - block)
    :precondition (and (not (= ?x ?y))
                    (clear ?x)
                    (ontable ?x)
                    (clear ?y))
    :effect (and (not (ontable ?x))
              (clear ?y)
              (ontable ?x)))

  (:action totable
    :parameters (?x - block ?y - block)
    :precondition (and (clear ?x) (on ?x ?y))
    :effect (and (not (on ?x ?y))
              (clear ?y)
              (ontable ?x)))
```

Jussi Rintanen
April 21, AI Planning 41/46

Jussi Rintanen
April 21, AI Planning 42/46

Jussi Rintanen
April 21, AI Planning 43/46

Jussi Rintanen
April 21, AI Planning 44/46
(:action move
  :parameters (?x - block
                 ?y - block
                 ?z - block)
  :precondition (and (clear ?x) (clear ?z)
                  (on ?x ?y) (not (= ?x ?z)))
  :effect
    (and (not (clear ?z))
         (clear ?y)
         (not (on ?x ?y))
         (on ?x ?z)))

(define (problem blocks-10-0)
  (:domain blocks)
  (:objects d a h g b j e i f c - block)
  (:init (clear c) (clear f)
          (ontable i) (ontable f)
          (on c e) (on e j) (on j b) (on b g)
          (on g h) (on h a) (on a d) (on d i))
  (:goal (and (on d c) (on c f) (on f j) (on j e)
              (on e h) (on h b) (on b a) (on a g)
              (on g i)))
)