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 + exercis
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:	$0.85\times$ exam $+0.15\times$ exercises

What is the course about?

- question: What actions to take to reach the goals?
- general-purpose problem representation and general-purpose algorithms
- application areas:
  - problem-solving (single-agent games like Rubik's cube etc.)
  - high-level planning for intelligent robots
  - autonomous systems: NASA Deep Space One

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

What is the course about?

Algorithms for deterministic planning:

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

What is the course about?

Algorithms for nondeterministic planning:

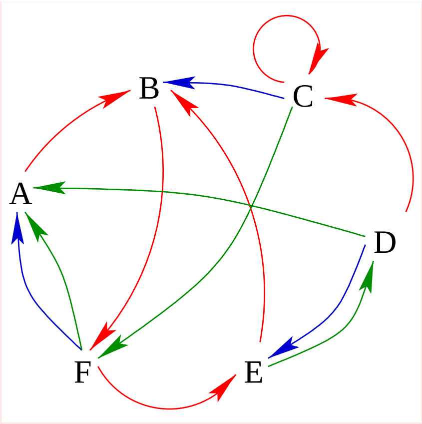
- conditional planning
- iterative algorithms for probabilistic planning (MDPs)
- extension of the techniques to very big state spaces with binary decision diagrams and related data structures.
- partial observability

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 $s' \in S$.



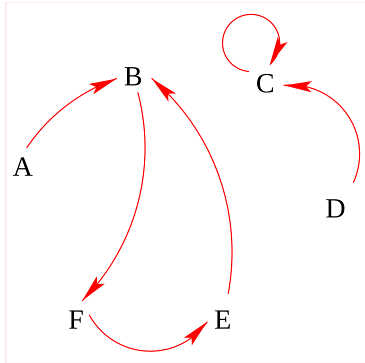
Actions as matrices

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.

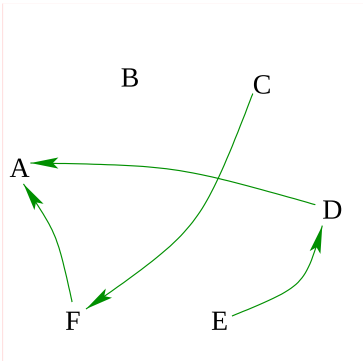
   For deterministic actions there is at most one non-zero element in each row.

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

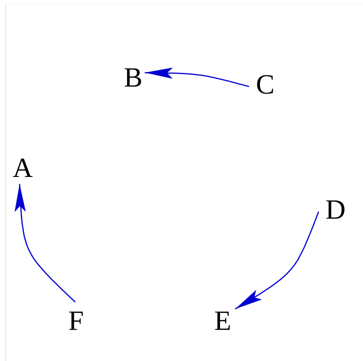
3. The unit matrix $I_{n \times n}$ is the NO-OP action.



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	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	0	1	0

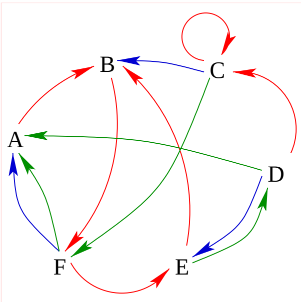


	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	1
D	1	0	0	0	0	0
E	0	0	0	1	0	0
F	1	0	0	0	0	0



	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	1	0	0	0	0
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	1	0	0	0	0	0

Sum matrix $M_R + M_G + M_B$



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	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

We use addition $0 + 0 = 0$ and $b + b' = 1$ if $b = 1$ or $b' = 1$.

Sequential composition as matrix multiplication

$$\left(\begin{array}{cccccc} 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right) \times \left(\begin{array}{cccc|c|c} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right) = \left(\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{array}\right)$$

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.

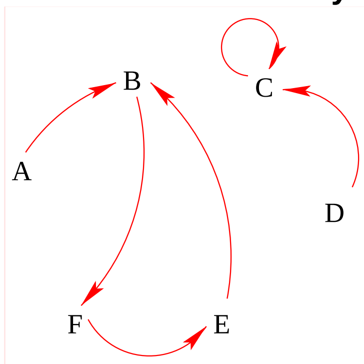
Reachability

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

$$\begin{aligned} R_0 &= I_{n\times n} \\ R_1 &= I_{n\times n} + M \\ R_2 &= I_{n\times n} + M + M^2 \\ R_3 &= I_{n\times n} + M + M^2 + M^3 \\ &\vdots \end{aligned}$$

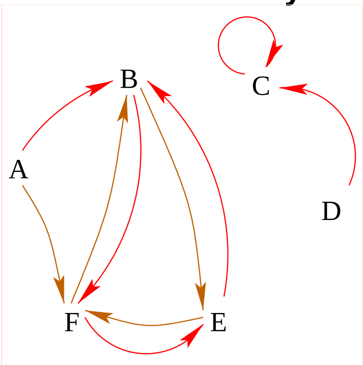
R_i represents reachability by i actions or less. If s' is reachable from s, then it is reachable with $\leq n$ actions: $R_n = R_{n+1}$.

Reachability: example, M_R



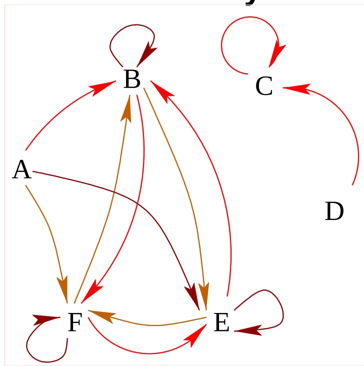
	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

Reachability: example, $M_R + M_R^2$



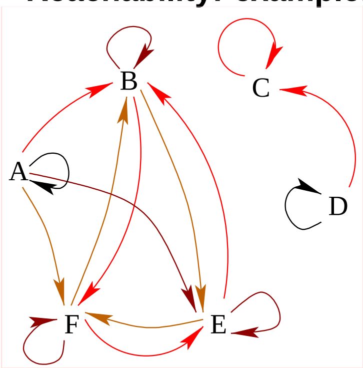
	A	B	C	D	E	F
A	0	1	0	0	0	1
B	0	0	0	0	1	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	1
F	0	1	0	0	1	0

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



	A	B	C	D	E	F
A	0	1	0	0	1	1
B	0	1	0	0	1	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	1	1
F	0	1	0	0	1	1

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



	A	B	C	D	E	F
A	1	1	0	0	1	1
B	0	1	0	0	1	1
C	0	0	1	0	0	0
D	0	0	1	1	0	0
E	0	1	0	0	1	1
F	0	1	0	0	1	1

Reachability: row vectors are sets of states

Row vectors S represent sets.

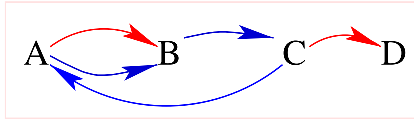
SM is the set of states reachable from S by M.

$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}^T \times \begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 1 & 1 & 0 & 0 & 1 & 1 \\ B & 0 & 1 & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 1 & 1 & 0 & 0 \\ E & 0 & 1 & 0 & 0 & 1 & 1 \\ F & 0 & 1 & 0 & 0 & 1 & 1 \end{array} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}^T$$

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



$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 1 \\
D & 0 & 0 & 0 & 0
\end{array}
+
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 1 & 0 \\
C & 1 & 0 & 0 & 0 \\
D & 0 & 0 & 0 & 0
\end{array}
=
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 1 & 0 \\
C & 1 & 0 & 0 & 1 \\
D & 0 & 0 & 0 & 0
\end{array}
$$

$$
R_0 = \begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 1 & 0 & 0 & 0 \\
B & 0 & 1 & 0 & 0 \\
C & 0 & 0 & 1 & 0 \\
D & 0 & 0 & 0 & 1
\end{array}
\quad
R_1 = \begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 1 & 1 & 0 & 0 \\
B & 0 & 1 & 1 & 0 \\
C & 1 & 0 & 1 & 1 \\
D & 0 & 0 & 0 & 1
\end{array}
$$

$$
R_2 = \begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 1 & 1 & 1 & 0 \\
B & 1 & 1 & 1 & 1 \\
C & 1 & 0 & 1 & 1 \\
D & 0 & 0 & 0 & 1
\end{array}
\quad
R_3 = \begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 1 & 1 & 1 & 1 \\
B & 1 & 1 & 1 & 1 \\
C & 1 & 1 & 1 & 1 \\
D & 0 & 0 & 0 & 1
\end{array}
$$

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 : $\{$ sunny, cloudy, rainy $\}$ = cloudy, HOLIDAY : $\{$ T, F $\}$ = F
- Any n-valued state variable can be replaced by $\lceil \log_2 n \rceil$ 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 ϕ is a propositional formula, then so is $\neg\phi$.
3. If ϕ and ϕ' are propositional formulae, then so is $\phi \vee \phi'$.
4. If ϕ and ϕ' are propositional formulae, then so is $\phi \wedge \phi'$.
5. The symbols $\perp$ and $\top$ are propositional formulae.

The implication $\phi \rightarrow \phi'$ is an abbreviation for $\neg\phi \vee \phi'$.

The equivalence $\phi \leftrightarrow \phi'$ is an abbreviation for $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

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 $v \not\models \phi$
3. $v \models \phi \vee \phi'$ if and only if $v \models \phi$ or $v \models \phi'$
4. $v \models \phi \wedge \phi'$ if and only if $v \models \phi$ and $v \models \phi'$
5. $v \models \top$
6. $v \not\models \bot$

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

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

A propositional formula ϕ is *a logical consequence* of a propositional formula ϕ', 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*.

operation on sets	operation on formulae
$A \cup B$	$A \vee B$
$A \cap B$	$A \wedge B$
$A \backslash B$	$A \wedge \neg B$
question about sets of states	question about formulae
$A \subseteq B$?	$A \models B$?
$A \subset B$?	$A \models B$ and $B \not\models A$?
$A = B$?	$A \models B$ and $B \models A$?
$\bot$	the empty set
$\top$	the universal set

Operators

Actions are represented as *operators* $\langle c, e \rangle$.

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. $c \rhd e$ is an effect if c is a formula over P and e is an effect.

Operators: effects

$c \rhd e$ means that change e takes place if c is true in the current state.

EXAMPLE: Increment 3-bit numbers $p_2 p_1 p_0$.

$$\begin{array}{rcl}(\neg p_0 & \rhd & p_0) \wedge \\ ((\neg p_1 \wedge p_0) & \rhd & (p_1 \wedge \neg p_0)) \wedge \\ ((\neg p_2 \wedge p_1 \wedge p_0) & \rhd & (p_2 \wedge \neg p_1 \wedge \neg p_0))\end{array}$$

Example: operators for blocks world

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

⟨A-CLEAR∧A-ON-TABLE∧B-CLEAR, A-ON-B∧¬A-ON-TABLE∧¬B-CLEAR⟩
⟨A-CLEAR∧A-ON-TABLE∧C-CLEAR, A-ON-C∧¬A-ON-TABLE∧¬C-CLEAR⟩
⋮
⟨A-CLEAR ∧ (A-ON-B ∨ A-ON-C), A-ON-TABLE ∧ ¬A-ON-B ∧ ¬A-ON-C⟩
⟨B-CLEAR ∧ (B-ON-A ∨ B-ON-C), B-ON-TABLE ∧ ¬B-ON-A ∧ ¬B-ON-C⟩
⋮

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. $[c' \rhd e]_s = [e]_s$ if $s \models c'$ and $[c' \rhd e]_s = \emptyset$ otherwise.

Operators: the successor state of a state

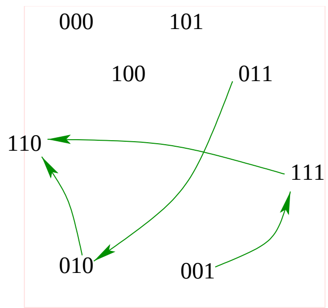
The successor $\text{app}_o(s)$ of s with respect to operator $o = \langle c, e \rangle$ is obtained from s by making literals $[e]_s$ true.

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

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

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

Operators: example



state variables A, B, C

$$\langle (B \wedge C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge C), \\ ((B \wedge C) \rhd \neg C) \\ \wedge (\neg B \rhd (A \wedge B)) \\ \wedge (\neg C \rhd A) \rangle$$

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 $\langle \text{in}(x, y_1), \text{in}(x, y_2) \wedge \neg\text{in}(x, y_1) \rangle$ with

x, t_1 and t_2 taking values $\begin{array}{l} x \in \{\text{car1}, \text{car2}\} \\ y_1 \in \{\text{Freiburg}, \text{Strassburg}\}, \\ y_2 \in \{\text{Freiburg}, \text{Strassburg}\}, y_1 \neq y_2 \end{array}$

corresponds to a set of operators:

$$\begin{array}{ll} \{ & \langle \text{in}(\text{car1}, \text{Freiburg}), \text{in}(\text{car1}, \text{Strassburg}) \wedge \neg\text{in}(\text{car1}, \text{Freiburg}) \rangle, \\ & \langle \text{in}(\text{car1}, \text{Strassburg}), \text{in}(\text{car1}, \text{Freiburg}) \wedge \neg\text{in}(\text{car1}, \text{Strassburg}) \rangle, \\ & \langle \text{in}(\text{car2}, \text{Freiburg}), \text{in}(\text{car2}, \text{Strassburg}) \wedge \neg\text{in}(\text{car2}, \text{Freiburg}) \rangle, \\ & \langle \text{in}(\text{car2}, \text{Strassburg}), \text{in}(\text{car2}, \text{Freiburg}) \wedge \neg\text{in}(\text{car2}, \text{Strassburg}) \rangle \} \end{array}$$

Schematic operators: quantification

existential quantification: finite disjunctions (not for effects)

universal quantification: finite conjunctions

EXAMPLE:

$\exists x \in \{A, B, C\}in(x,$Freiburg$)$ is a short-hand for

in$(A,$Freiburg$) \vee$ in$(B,$Freiburg$) \vee$ in$(C,$Freiburg$)$.

Example: blocks world in PDDL

```
(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))
           (not (clear ?y))
           (on ?x ?y)))
```



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

```
  (: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)))
)
```