Transition systems
Definition
Example
Matrices
Reachability
Algorithm

Succinct transition systems
State variables
Propositional logic
Operators
Schematic operators

Definition
A transition system is \( \langle S, I, \{a_1, \ldots, a_n\}, G \rangle \) where
- \( S \) is a finite set of states (the state space),
- \( I \subseteq S \) is a finite set of initial states,
- every action \( a_i \subseteq S \times S \) is a binary relation on \( S \),
- \( G \subseteq S \) is a finite set of goal states.

Definition
An action \( a \) is applicable in a state \( s \) if \( sas' \) for at least one state \( s' \).
Transition systems

Deterministic transition systems

A transition system is deterministic if there is only one initial state and all actions are deterministic. Hence all future states of the world are completely predictable.

Definition

A deterministic transition system is \((S, I, O, G)\) where

- \(S\) is a finite set of states (the state space),
- \(I \in S\) is a state,
- actions \(a \in O\) (with \(a \subseteq S \times S\)) are partial functions,
- \(G \subseteq S\) is a finite set of goal states.

Successor state wrt. an action

Given a state \(s\) and an action \(A\) so that \(a\) is applicable in \(s\), the successor state of \(s\) with respect to \(a\) is \(s'\) such that \(sas'\), denoted by \(s' = app_a(s)\).

Blocks world

The rules of the game

Location on the table does not matter.

Location on a block does not matter.

Blocks world

The transition graph for three blocks
**Blocks world**

**Properties**

<table>
<thead>
<tr>
<th>blocks</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>73</td>
</tr>
<tr>
<td>5</td>
<td>501</td>
</tr>
<tr>
<td>6</td>
<td>4051</td>
</tr>
<tr>
<td>7</td>
<td>37633</td>
</tr>
<tr>
<td>8</td>
<td>459653</td>
</tr>
<tr>
<td>9</td>
<td>58941091</td>
</tr>
</tbody>
</table>

1. Finding a solution is polynomial time in the number of blocks (move everything onto the table and then construct the goal configuration).
2. Finding a shortest solution is NP-complete (for a compact description of the problem).

---

**Deterministic planning: plans**

**Definition**

A plan for \((S, I, O, G)\) is a sequence \(\pi = o_1, \ldots, o_n\) of operators such that \(o_1, \ldots, o_n \in O\) and \(s_0, \ldots, s_n\) is a sequence of states (the execution of \(\pi\)) so that

1. \(s_0 = I\),
2. \(s_i = app(o_i(s_{i-1}))\) for every \(i \in \{1, \ldots, n\}\), and
3. \(s_n \in G\).

This can be equivalently expressed as

\[ app(o_n(app(o_{n-1}(\cdots(app(o_1(I))\cdots))) \in G \]

---

**Transition relations as matrices**

1. If there are \(n\) states, each action (a binary relation) corresponds to an \(n \times n\) matrix: The element at row \(i\) and column \(j\) is 1 if the action maps state \(i\) to state \(j\), and 0 otherwise. 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 also corresponds to the **join** of the relations.)
3. The unit matrix \(I_{n\times n}\) is the NO-OP action: every state is mapped to itself.

---

**Example**

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

Transition systems
Matrices

Sum matrix $M_R + M_G + M_B$
Representing one-step reachability by any of the component actions

Sequential composition as matrix multiplication

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

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.
Transition systems

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 $\leq n - 1$ actions: $R_{n-1} = R_n$.

### Reachability: example, $M_R$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Reachability: example, \( M_R + M_R^2 + M_R^3 + I_{6 \times 6} \)

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

Row vectors \( S \) represent sets. \( SM \) is the set of states reachable from \( S \) by \( M \).

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

A simple planning algorithm

- We next present a simple planning algorithm based on computing distances in the transition graph.
- The algorithm finds shortest paths less efficiently than Dijkstra’s algorithm; we present the algorithm because we later will use it as a basis of an algorithm that is applicable to much bigger state spaces than Dijkstra’s algorithm directly.
A simple planning algorithm

1. Compute the matrices $R_0$, $R_1$, $R_2$, \ldots, $R_n$ representing reachability with 0, 1, 2, \ldots, $n$ steps with all actions.
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. Repeat the last step, now viewing $s_g'$ as the goal state with distance $i - 1$.

Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0 1 0 0</td>
<td>A</td>
<td>1 1 0 0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0 0 1 0</td>
<td>B</td>
<td>0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0 0 0 1</td>
<td>C</td>
<td>1 0 1 1</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0 0 0 0</td>
<td>D</td>
<td>0 0 0 1</td>
<td></td>
</tr>
</tbody>
</table>

$s_0 = A$

$R_0 = A$

$R_1 = \begin{array}{cccc}
A & 1 & 1 & 0 \\
B & 0 & 1 & 1 \\
C & 0 & 0 & 1 \\
D & 0 & 0 & 0 \\
\end{array} + \begin{array}{cccc}
A & 1 & 0 & 0 \\
B & 0 & 1 & 0 \\
C & 0 & 0 & 1 \\
D & 0 & 0 & 0 \\
\end{array} = R_0 + R_1 = A$

$R_2 = \begin{array}{cccc}
A & 1 & 0 & 0 \\
B & 0 & 1 & 0 \\
C & 0 & 1 & 0 \\
D & 0 & 0 & 0 \\
\end{array} + \begin{array}{cccc}
A & 0 & 1 & 0 \\
B & 0 & 0 & 1 \\
C & 0 & 0 & 1 \\
D & 0 & 0 & 0 \\
\end{array} = R_0 + R_1 + R_2 = A$

Succinct representation of transition systems

- More compact representation of actions than as relations is often possible because of symmetries and other regularities.
- Unavoidable because the relations are too big.
- Represent different aspects of the world in terms of different state variables. $\Rightarrow$ A state is a valuation of state variables.
- Represent actions in terms of changes to the 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.

State variables

Blocks world with state variables

State variables:

LOCATIONofA : \{ B, C, TABLE \}
LOCATIONofB : \{ A, C, TABLE \}
LOCATIONofC : \{ A, B, TABLE \}

Example

\( s(LOCATIONofA) = TABLE \)
\( s(LOCATIONofB) = A \)
\( s(LOCATIONofC) = TABLE \)

Not all valuations correspond to an intended blocks world state, e.g. \( s \) such that \( s(LOCATIONofA) = B \) and \( s(LOCATIONofB) = A \).

Blocks world with Boolean state variables

Example

\( s(AonB) = 0 \)
\( s(AonC) = 0 \)
\( s(AonTABLE) = 1 \)
\( s(BonA) = 1 \)
\( s(BonC) = 0 \)
\( s(BonTABLE) = 0 \)
\( s(ConA) = 0 \)
\( s(ConB) = 0 \)
\( s(ConTABLE) = 1 \)

Logical representations of state sets

\( n \) 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 propositional logic.

Logical connectives correspond to set-theoretical operations.

Logical relations correspond to set-theoretical relations.
Propositional logic

Let $A$ be a set of atomic propositions (~state variables.)

1. For all $a \in A$, $a$ is a propositional formula.
2. If $\phi$ is a propositional formula, then so is $\neg \phi$.
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 $\neg \phi \lor \phi'$.
The equivalence $\phi \leftrightarrow \phi'$ is an abbreviation for $(\phi \rightarrow \phi') \land (\phi' \rightarrow \phi)$.

M. Helmert, B. Nebel (Universität Freiburg) AI Planning November 3rd, 2006 33 / 62

Propositional logic

Valuations and truth

A valuation of $A$ is a function $v : A \rightarrow \{0, 1\}$. Define the notation $v \models \phi$ for valuations $v$ and formulae $\phi$ by

1. $v \models a$ if and only if $v(a) = 1$, for $a \in A$.
2. $v \models \neg \phi$ if and only if $v \not\models \phi$
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. $v \models \top$
6. $v \not\models \bot$

Formulae vs. sets

Some terminology

- 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 $a$ or a negated proposition $\neg a$ for some $a \in A$ is a **literal**.
- A formula that is a disjunction of literals is a **clause**.

M. Helmert, B. Nebel (Universität Freiburg) AI Planning November 3rd, 2006 35 / 62
Operators

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

- $c$ (the precondition) is a propositional formula over $A$ 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 do the values of the state variable change?

Effects

For deterministic operators

Definition

Effects are recursively defined as follows.

1. $a$ and $\neg a$ for state variables $a \in A$ are effects.
2. $e_1 \land \cdots \land e_n$ 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 and $e$ is an effect.

Atomic effects $a$ and $\neg a$ are best understood respectively as assignments $a := 1$ and $a := 0$.

Example: operators for blocks world

In addition to state variables like $AonT$ and $BonC$, for convenience we also use state variables $Aclear$, $Bclear$, and $Cclear$ to denote that there is nothing on the block in question.

$$\langle Aclear \land AonT \land Bclear, \ AonB \land \neg AonT \land \neg Bclear \rangle$$
$$\langle Aclear \land AonT \land Cclear, \ AonC \land \neg AonT \land \neg Cclear \rangle$$
$$\vdots$$
$$\langle Aclear \land AonB, \ AonT \land \neg AonB \land Bclear \rangle$$
$$\langle Aclear \land AonC, \ AonT \land \neg AonC \land Cclear \rangle$$
$$\langle Bclear \land BonA, \ BonT \land \neg BonA \land Aclear \rangle$$
$$\langle Bclear \land BonC, \ BonT \land \neg BonC \land Cclear \rangle$$
$$\vdots$$
Operators: meaning

Changes caused by an operator
Assign to each effect $e$ and state $s$ a set $[e]_s$ of literals as follows.

1. $[a]_s = \{a\}$ and $[\neg a]_s = \{\neg a\}$ for $a \in A$.
2. $[e_1 \land \cdots \land e_n]_s = [e_1]_s \cup \ldots \cup [e_n]_s$.
3. $[c \triangleright e]_s = [e]_s$ if $s \models c$ and $[c \triangleright e]_s = \emptyset$ otherwise.

Applicability of an operator
Operator $\langle c, e \rangle$ is applicable in a state $s$ iff $s \models c$ and $[e]_s$ is consistent.

Example
Consider the operator $\langle a, \neg a \land (\neg c \triangleright \neg b) \rangle$ and a state $s$ such that $s \models a \land b \land c$.

The operator is applicable because $s \models a$ and $[\neg a \land (\neg c \triangleright \neg b)]_s = \{\neg a\}$ is consistent.

Hence $app(\langle a, \neg a \land (\neg c \triangleright \neg b) \rangle)(s) \models \neg a \land b \land c$.

Succinct transition systems
Deterministic case

Definition
A succinct deterministic transition system is $(A, I, \{o_1, \ldots, o_n\}, G)$ where

- $A$ is a finite set of state variables,
- $I$ is an initial state,
- every $o_i$ is an operator,
- $G$ is a formula describing the goal states.
Mapping from succinct TS to TS

From every succinct transition system \( \langle A, I, O, G \rangle \) we can produce a corresponding transition system \( \langle S, I, O', G' \rangle \).

1. \( S \) is the set of all valuations of \( A \),
2. \( O' = \{ R(o) | o \in O \} \) where \( R(o) = \{(s, s') \in S \times S | s' = \text{app}_o(s)\} \), and
3. \( G' = \{ s \in S | s \models G \} \).

Schematic operators: example

Schematic operator

\[
\begin{align*}
  x & \in \{\text{car1}, \text{car2}\} \\
  y_1 & \in \{\text{Freiburg}, \text{Strassburg}\}, \\
  y_2 & \in \{\text{Freiburg}, \text{Strassburg}\}, y_1 \neq y_2 \\
  \langle \text{in}(x, y_1), \text{in}(x, y_2) \land \neg \text{in}(x, y_1) \rangle
\end{align*}
\]

corresponds to the operators

\[
\begin{align*}
  \langle \text{in(car1, Freiburg)}, \text{in(car1, Strassburg)} \land \neg \text{in(car1, Freiburg)} \rangle, \\
  \langle \text{in(car1, Strassburg)}, \text{in(car1, Freiburg)} \land \neg \text{in(car1, Strassburg)} \rangle, \\
  \langle \text{in(car2, Freiburg)}, \text{in(car2, Strassburg)} \land \neg \text{in(car2, Freiburg)} \rangle, \\
  \langle \text{in(car2, Strassburg)}, \text{in(car2, Freiburg)} \land \neg \text{in(car2, Strassburg)} \rangle
\end{align*}
\]

Schematic operators: quantification

Existential quantification (for formulae only)

Finite disjunctions \( \phi(a_1) \lor \cdots \lor \phi(a_n) \) represented as

\[ \exists x \in \{a_1, \ldots, a_n\} \phi(x). \]

Universal quantification (for formulae and effects)

Finite conjunctions \( \phi(a_1) \land \cdots \land \phi(a_n) \) represented as

\[ \forall x \in \{a_1, \ldots, a_n\} \phi(x). \]

Example

\[ \exists x \in \{A, B, C\} \text{in}(x, \text{Freiburg}) \text{ is a short-hand for } \text{in}(A, \text{Freiburg}) \lor \text{in}(B, \text{Freiburg}) \lor \text{in}(C, \text{Freiburg}). \]
PDDL: the Planning Domain Description Language

- Used by almost all implemented systems for deterministic planning.
- Supports a language comparable to what we have defined above (including schematic operators and quantification).
- Syntax inspired by the Lisp programming language: e.g. prefix notation for formulae.

```plaintext
(and (or (on A B) (on A C))
 (or (on B A) (on B C))
 (or (on C A) (on A B)))
```

PDDL: domain files

A domain file consists of
- (define (domain DOMAINNAME)
- a :requirements definition (use :adl :typing by default)
- definitions of types (each parameter has a type)
- definitions of predicates
- definitions of operators

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

PDDL: operator definition

- (:action OPERATORNAME
- list of parameters: (?x - type1 ?y - type2 ?z - type3)
- precondition: a formula
  <schematic-state-var>
  (and <formula> ... <formula>)
  (or <formula> ... <formula>)
  (not <formula>)
  (forall (?x1 - type1 ... ?xn - typen) <formula>)
  (exists (?x1 - type1 ... ?xn - typen) <formula>)
PDDL: problem files

A problem file consists of

- (define (problem PROBLEMNAME)
  declaration of which domain is needed for this problem
- definitions of objects belonging to each type
- definition of the initial state (list of state variables initially true)
- definition of goal states (a formula like operator precondition)
Example run on the FF planner

```
edu/PS04> ./ff -o blocks-dom.pddl -f blocks-ex.pddl
ff: parsing domain file, domain 'BLOCKS' defined
ff: parsing problem file, problem 'EXAMPLE' defined
ff: found legal plan as follows
step 0: FROMTABLE A D
  1: FROMTABLE B E
  2: FROMTABLE C F
0.01 seconds total time
```

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)))