Transition systems

goal states

initial state

A

B

C

D

E

F

Reachability

Algorithm

Matrices

Example

Definition

AI Planning

Succinct TS
### 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 $s'$. 
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 \( \langle S, I, O, G \rangle \) 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

At most one block on/under a block is allowed
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>394353</td>
</tr>
<tr>
<td>9</td>
<td>4596553</td>
</tr>
<tr>
<td>10</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 $\langle S, I, O, G \rangle$ 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. $s_i = \text{app}_{o_i}(s_{i-1})$ for every $i \in \{1, \ldots, n\}$, and
3. $s_n \in G$.

This can be equivalently expressed as

$$\text{app}_{o_n}(\text{app}_{o_{n-1}}(\cdots \text{app}_{o_1}(I) \cdots )) \in G$$
Transition relations as matrices

1. If there are \( n \) states, each action (a binary relation) corresponds to an \( n \times n \) matrix: 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

Transition systems
Definition
Example
Matrices
Reachability
Algorithm
Succinct TS

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

Transition systems
Definition
Example
Matrices
Reachability
Algorithm
Succinct TS

<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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

### Transition systems

**Definition**

**Example**

**Matrices**

**Reachability**

**Algorithm**

**Succinct TS**

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

![Diagram showing transitions between states A, B, C, D, E, F]
Sum matrix $M_R + M_G + M_B$

Representing one-step reachability by any of the component actions

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

<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>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Sequential composition as matrix multiplication

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

\[
\begin{align*}
R_0 &= I_{n \times n} \\
R_1 &= I_{n \times n} + M \\
R_2 &= I_{n \times n} + M + M^2 \\
R_3 &= I_{n \times n} + M + M^2 + M^3 \\
\vdots
\end{align*}
\]

$R_i$ 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, $M_R$

\[
\begin{array}{cccccc}
 & A & B & C & D & E & F \\
A & 0 & 1 & 0 & 0 & 0 & 0 \\
B & 0 & 0 & 0 & 0 & 0 & 1 \\
C & 0 & 0 & 1 & 0 & 0 & 0 \\
D & 0 & 0 & 1 & 0 & 0 & 0 \\
E & 0 & 1 & 0 & 0 & 0 & 0 \\
F & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]
Reachability: example, $M_R + M_R^2$
Reachability: example, $M_R + M_R^2 + M_R^3$
Reachability: example, $M_R + M_R^2 + M_R^3 + I_{6\times6}$
Row vectors $S$ represent sets. $SM$ is the set of states reachable from $S$ by $M$. 

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

Idea

distance from the initial state

0  1  2  3
A simple planning algorithm

Idea

distance from the initial state

0 1 2 3
A simple planning algorithm

Idea

distance from the initial state

0 1 2 3
A simple planning algorithm

Idea

distance from the initial state

0 1 2 3
A simple planning algorithm

Idea

distance from **the initial state**

0 1 2 3
A simple planning algorithm

Idea

distance from the initial state

0 1 2 3
A simple planning algorithm

Idea

distance from the initial state

0  1  2  3
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

\[
\begin{array}{cccc}
A & B & C & D \\
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 0 & 0 \\
C & 0 & 0 & 0 & 1 \\
D & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 1 & 0 \\
C & 1 & 0 & 0 & 0 \\
D & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 1 & 0 \\
C & 1 & 0 & 0 & 1 \\
D & 0 & 0 & 0 & 0 \\
\end{array}
\]
Example

Transition systems
Definition
Example
Matrices
Reachability
Algorithm
Succinct TS

\[
R_0 = \begin{array}{c|cccc}
A & B & C & D \\
\hline
A & 1 & 0 & 0 & 0 \\
B & 0 & 1 & 0 & 0 \\
C & 0 & 0 & 1 & 0 \\
D & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
R_1 = \begin{array}{c|cccc}
A & B & C & D \\
\hline
A & 1 & 1 & 0 & 0 \\
B & 0 & 1 & 1 & 0 \\
C & 1 & 0 & 1 & 1 \\
D & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
R_2 = \begin{array}{c|cccc}
A & B & C & D \\
\hline
A & 1 & 1 & 1 & 0 \\
B & 1 & 1 & 1 & 1 \\
C & 1 & 1 & 1 & 1 \\
D & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
R_3 = \begin{array}{c|cccc}
A & B & C & D \\
\hline
A & 1 & 1 & 1 & 1 \\
B & 1 & 1 & 1 & 1 \\
C & 1 & 1 & 1 & 1 \\
D & 0 & 0 & 0 & 1 \\
\end{array}
\]
Succinct representation of transition systems

- More **compact** representation of actions than as relations is often
  
  1. **possible** because of symmetries and other regularities,
  2. **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**: \( \{\text{sunny, cloudy, rainy}\} = \text{cloudy} \)
- **HOLIDAY**: \( \{\text{T, F}\} = \text{F} \)

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.
Blocks world with state variables

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

Example

\[ s(LOCATIONofA) = \text{TABLE} \]
\[ s(LOCATIONofB) = A \]
\[ s(LOCATIONofC) = \text{TABLE} \]

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

\[
\begin{align*}
\text{s(A on B)} &= 0 & \text{s(A on C)} &= 0 & \text{s(A on TABLE)} &= 1 \\
\text{s(B on A)} &= 1 & \text{s(B on C)} &= 0 & \text{s(B on TABLE)} &= 0 \\
\text{s(Con A)} &= 0 & \text{s(Con B)} &= 0 & \text{s(Con TABLE)} &= 1
\end{align*}
\]
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 the 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 $(\phi \rightarrow \phi') \land (\phi' \rightarrow \phi)$.
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. $v \models T$
6. $v \not\models \bot$
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 $a$ 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**.
### Formulae vs. sets

<table>
<thead>
<tr>
<th>sets</th>
<th>formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>those $\frac{2^n}{2}$ states in which $a$ is true</td>
<td>$a \in A$</td>
</tr>
<tr>
<td>$E \cup F$</td>
<td>$E \lor F$</td>
</tr>
<tr>
<td>$E \cap F$</td>
<td>$E \land F$</td>
</tr>
<tr>
<td>$E \setminus F$ (set difference)</td>
<td>$E \land \neg F$</td>
</tr>
<tr>
<td>$\overline{E}$ (complement)</td>
<td>$\neg E$</td>
</tr>
<tr>
<td>the empty set $\emptyset$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>the universal set</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

### question about sets

- $E \subseteq F$?
- $E \subset F$?
- $E = F$?

### question about formulae

- $E \models F$?
- $E \models F$ and $F \not\models E$?
- $E \models F$ and $F \models E$?
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 then 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 $a := 0$. 
\( c \triangleright e \) means that change \( e \) takes place if \( c \) is true in the current state.

**Example**

Increment 4-bit numbers \( b_3b_2b_1b_0 \).

\[
\begin{align*}
(\neg b_0 \triangleright b_0) \land \\
((\neg b_1 \land b_0) \triangleright (b_1 \land \neg b_0)) \land \\
((\neg b_2 \land b_1 \land b_0) \triangleright (b_2 \land \neg b_1 \land \neg b_0)) \land \\
((\neg b_3 \land b_2 \land b_1 \land b_0) \triangleright (b_3 \land \neg b_2 \land \neg b_1 \land \neg b_0))
\end{align*}
\]
For convenience we use also state variables $A_{\text{clear}}$, $B_{\text{clear}}$, and $C_{\text{clear}}$ to denote that there is nothing on the block in question.

\[
\langle A_{\text{clear}} \land A_{\text{onT}} \land B_{\text{clear}}, \ A_{\text{onB}} \land \neg A_{\text{onT}} \land \neg B_{\text{clear}} \rangle \\
\langle A_{\text{clear}} \land A_{\text{onT}} \land C_{\text{clear}}, \ A_{\text{onC}} \land \neg A_{\text{onT}} \land \neg C_{\text{clear}} \rangle \\
\vdots \\
\langle A_{\text{clear}} \land A_{\text{onB}}, \ A_{\text{onT}} \land \neg A_{\text{onB}} \land \neg A_{\text{onC}} \rangle \\
\langle A_{\text{clear}} \land A_{\text{onC}}, \ A_{\text{onT}} \land \neg A_{\text{onB}} \land \neg A_{\text{onC}} \rangle \\
\langle B_{\text{clear}} \land B_{\text{onA}}, \ B_{\text{onT}} \land \neg B_{\text{onA}} \land A_{\text{clear}} \rangle \\
\langle B_{\text{clear}} \land B_{\text{onC}}, \ B_{\text{onT}} \land \neg B_{\text{onC}} \land C_{\text{clear}} \rangle \\
\vdots
\]
Operators: meaning

Changes caused by an operator

Assign 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. \( [e_1 \land \ldots \land e_n]_s = [e_1]_s \cup \ldots \cup [e_n]_s \).
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.
Operators: the successor state of a state

**Definition (Successor state)**

The successor state $app_o(s)$ of $s$ with respect to operator $o = \langle c, e \rangle$ is obtained from $s$ by making literals in $[e]_s$ true. This is defined only if $o$ is applicable in $s$.

**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 $app_{\langle a, \neg a \land (\neg c \triangleright \neg b) \rangle}(s) \models \neg a \land b \land c$. 
State variables are

\[ A = \{a, b, c\} \]

An operator is

\[
\langle (b \land c) \lor (\neg a \land b \land \neg c) \lor (\neg a \land c),
\((b \land c) \triangleright \neg c)
\land (\neg b \triangleright (a \land b))
\land (\neg c \triangleright a) \rangle
\]
The corresponding matrix is

<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>0</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>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>010</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>
<tr>
<td>011</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>100</td>
<td>0</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>101</td>
<td>0</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>110</td>
<td>0</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>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>
A succinct deterministic transition system is
\( \langle A, I, \{ o_1, \ldots, o_n \}, G \rangle \) 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.
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

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

Schematic operator

\[ x \in \{\text{car1, car2}\}, \quad y_1 \in \{\text{Freiburg, Strassburg}\}, \quad y_2 \in \{\text{Freiburg, Strassburg}\}, \quad y_1 \neq y_2 \]

\[ \langle \text{in}(x, y_1), \text{in}(x, y_2) \land \neg \text{in}(x, y_1) \rangle \]

corresponds to the operators

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

$\exists x \in \{A, B, C\} \text{in}(x, \text{Freiburg})$ 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

(\text{and} \ (\text{or} \ (\text{on} \ A \ B) \ (\text{on} \ A \ C)) \ (\text{or} \ (\text{on} \ B \ A) \ (\text{on} \ B \ C)) \ (\text{or} \ (\text{on} \ C \ A) \ (\text{on} \ A \ B)))
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>)
effect:

<schematic-state-var>
(not <schematic-state-var>)
(and <effect> ... <effect>)
(when <formula> <effect>)
(forall (?x1 - type1 ... ?xn - typen) <effect>)
(:action fromtable
  :parameters (?x - smallblock ?y - block)
  :precondition (and (not (= ?x ?y))
                  (clear ?x)
                  (ontable ?x)
                  (clear ?y))
  :effect
  (and (not (ontable ?x))
       (not (clear ?y))
       (on ?x ?y)))
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)
(define (problem blocks-10-0)
  (:domain BLOCKS)
  (:objects a b c - smallblock)
    d e - block
    f - blueblock)
  (:init (clear a) (clear b) (clear c) (clear d) (clear e) (clear f)
    (ontable a) (ontable b) (ontable c)
    (ontable d) (ontable e) (ontable f))
  (:goal (and (on a d) (on b e) (on c f))))
Example run on the FF planner

```
edu/PS04> ./ff -o hamiltonian.pddl -f haml.pddl
ff: parsing domain file, domain 'HAMILTONIAN-CYCLE' defined
ff: parsing problem file, problem 'HAM-1' defined
ff: found legal plan as follows
step 0: GO A B
    1: GO B D
    2: GO D F
    3: GO F C
    4: GO C E
    5: GO E A
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)))
)
(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)))
  )