Transitionsystems

Definition

A transition system is a triple \( (S, I, O, G) \) 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.

Successor state wrt. an action

Given a state \( s \) and an action \( a \) so that \( a \) is applicable in \( s \), the successor state \( s' \) of \( s \) with respect to \( a \) is \( s' = \text{app}(a)(s) \).
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

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

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

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

Sequential composition as matrix multiplication

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

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

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

\( R_i \) 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, $M_R + M_R^2$

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

Relations and sets as matrices
Row vectors as sets of states

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

A simple planning algorithm

Idea

- 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.

Example

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 $s_f$ that is reachable from the initial state according to $R_{i-1}$.
4. Repeat the last step, now viewing $s_f$ as the goal state with distance $i - 1$. 
Example

State variables

- The state of the world is described in terms of a finite set of finite-valued state variables.

Example

<table>
<thead>
<tr>
<th>HOUR</th>
<th>MINUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0, ... , 23}</td>
<td>{0, ... , 59}</td>
</tr>
</tbody>
</table>

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 \([\log_2 n]\) Boolean (2-valued) state variables.
- Actions change the values of the state variables.

Blocks world with Boolean state variables

Example

s(A) = 0
s(B) = 1
s(C) = 0
s(D) = 1

Logical representations of state sets

- \(n\) state variables with \(m^n\) 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 (\(\sim\) 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 \(\psi\) are propositional formulae, then so is \(\phi \lor \psi\).
4. If \(\phi\) and \(\psi\) are propositional formulae, then so is \(\phi \land \psi\).
5. The symbols \(\bot\) and \(\top\) are propositional formulae.

The implication \(\phi \rightarrow \psi\) is an abbreviation for \(\neg \phi \lor \psi\).

The equivalence \(\phi \Leftrightarrow \psi\) is an abbreviation for \((\phi \rightarrow \psi) \land (\psi \rightarrow \phi)\).

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 \(v \not= \phi\).
3. \(v \models \phi \lor \psi\) if and only if \(v \models \phi\) or \(v \models \psi\).
4. \(v \models \phi \land \psi\) if and only if \(v \models \phi\) and \(v \models \psi\).
5. \(v \models \top\).
6. \(v \not= \bot\).
Propositional logic

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 \( \alpha \) or a negated proposition \( \neg \alpha \) for some \( \alpha \in \mathcal{A} \) is a **literal**.
- A formula that is a disjunction of literals is a **clause**.

### Operators

Actions are represented as operators \((c, e)\) where

- \( c \) (the **precondition**) is a propositional formula over \( \mathcal{A} \) 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

**Meaning of conditional effects** \( \triangleright \)

\( c \triangleright e \) means that change \( c \) takes place if \( e \) is true in the current state.

\[\text{Example}\]

Increment 4-bit numbers \( b_3b_2b_1b_0 \).

\((-b_0 \triangleright b_0) \land ((\neg b_1 \land b_0) \triangleright (b_1 \land \neg b_0)) \land (\neg b_3 \land b_2 \land b_1 \land b_0) \land (b_0 \land \neg b_1 \land \neg b_0)\)

### Operators: meaning

**Changes caused by an operator**

Assign each effect \( e \) and state \( s \) a set \([e],[s]\), of literals as follows.

1. \([\alpha]_e = \{\alpha\}\) and \([\neg \alpha]_e = \{\neg \alpha\}\) for \( \alpha \in \mathcal{A} \).
2. \([c_1 \land \cdots \land c_n]_e = [c_1]_e \cup \cdots \cup [c_n]_e\).
3. \([c \triangleright e]_s = [e]_s\) if \( s \models c \) and \([c \triangleright e]_s = \emptyset \) otherwise.

**Applicability of an operator**

Operator \((c, e)\) is applicable in a state \( s \) iff \( s \models c \) and \([c]_s\) is consistent.

### Example: operators for blocks world

For convenience we use also state variables \( Aclear, Bclear, \) and \( Cclear \) to denote that there is nothing on the block in question.

\[\begin{align*}
(Aclear \land AonT \land Bclear) \\
(Aclear \land AonC \land Cclear) \\
(Aclear \land AonB \land Bclear) \\
(Bclear \land Bona) \\
(Bclear \land Bona \land Aclear) \\
(Bclear \land Bona \land Cclear)
\end{align*}\]
Operators

Example

State variables are $A = \{a, b, c\}$.

An operator is
$$((b \land c) \lor \neg a \land \neg b \land \neg c) \lor (a \land c).$$

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>00</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>01</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>10</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>11</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>
</tbody>
</table>

Mapping from succinct TS To TS

From every succinct transition system $(A, I, O, G)$ we can produce a corresponding transition system $(S, I, O', G')$:

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

Schematic operators: example

Schematic operator

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

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

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

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

is the set of all valuationsof $A$.

is the set of all valuationsof $A$.

$O' = \{R(o) | o \in O\}$ where $R(o) = \{(s, s') \in S \times S | s' = app_o(s)\}$, and

$G' = \{s \in S | s \models G\}$.

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

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

PDDL: domain files

- 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

$$(\text{and} (\text{or} (\text{on} A B) (\text{on} A C))$$

$$(\text{or} (\text{on} B A) (\text{on} B C))$$

$$(\text{or} (\text{on} C A) (\text{on} A B))$$
Example: blocks world in PDDL

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

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