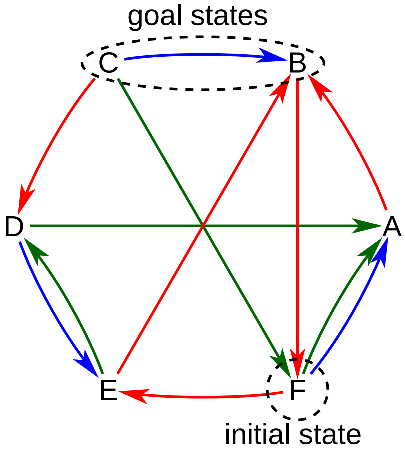
Transition systems



Transition systems

Formalization of the dynamics of the world/application

Definition

A transition system is $\langle S, I, \{a_1, \dots, a_n\}, G\rangle$ 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 $\langle S, I, O, G \rangle$ 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.

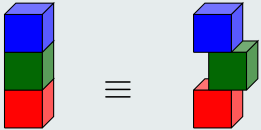
Successor state wrt. an action

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 $s' = app_a(s)$.

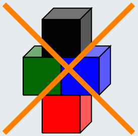
Blocks world
The rules of the game

Location on the table does not matter

Location on a block does not matter

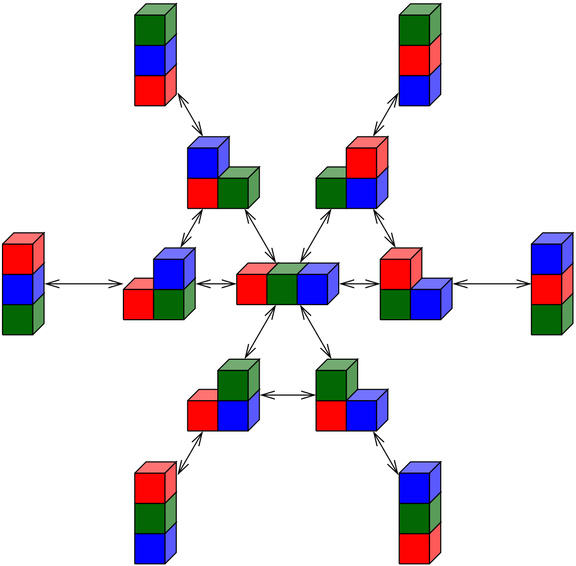


At most one block on/under a block is allowed



Blocks world

The transition graph for three blocks



Blocks world
Properties

blocks	states
1	1
2	3
3	13
4	73
5	501
6	4051
7	37633
8	394353
9	4596553
10	58941091

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 π) so that

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

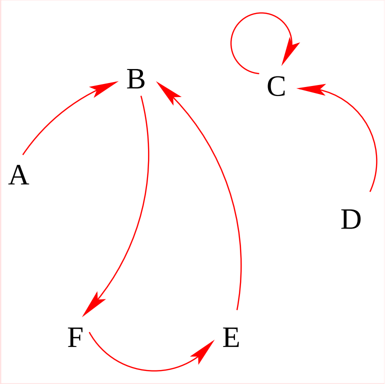
This can be equivalently expressed as

$$\mathit{app}_{o_n}(\mathit{app}_{o_{n-1}}(\cdots \mathit{app}_{o_1}(I)\cdots)) \in G$$

Transition relations as matrices

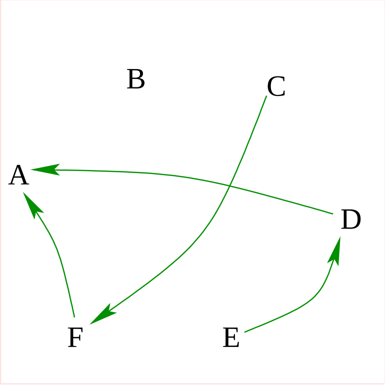
1. If there are n states, each action (a binary relation) corresponds to an $n \times n$ matrix: Element at row i and column j is 1 if the action maps state i to state j, and 0 otherwise.
   For deterministic actions there is at most one non-zero element in each row.
2. Matrix multiplication corresponds 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



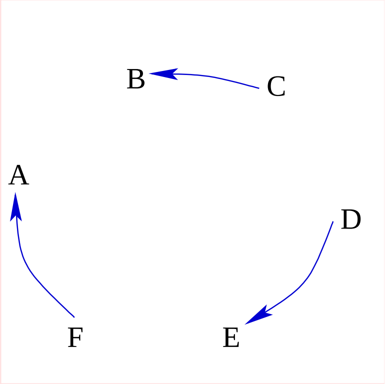
	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

Example



	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

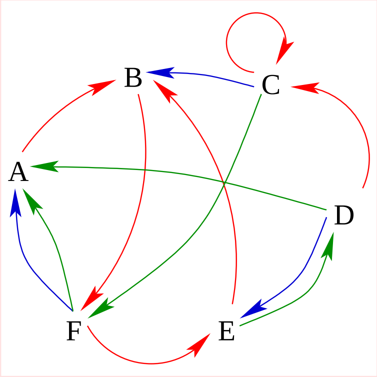
Example



	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 $M_R + M_G + M_B$

Representing one-step reachability by any of the component actions



	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

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

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.

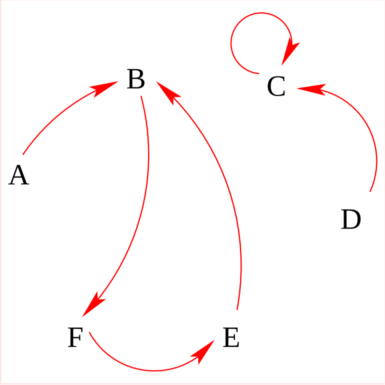
Reachability

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

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

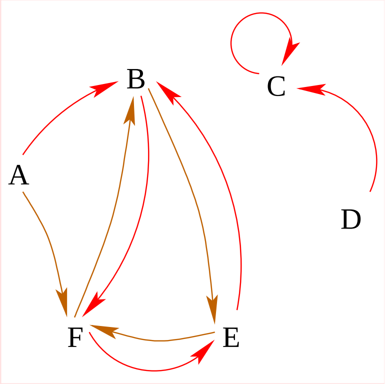
R_i represents reachability by i actions or less. If s' is reachable from s, then it is reachable with $\leq n-1$ actions: $R_{n-1} = R_n$.

Reachability: example, M_R



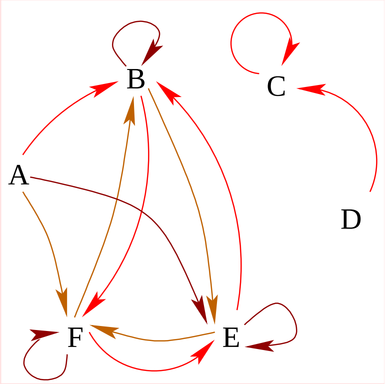
	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

Reachability: example, $M_R + M_R^2$



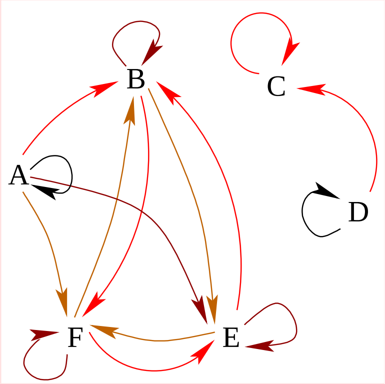
	A	B	C	D	E	F
A	0	1	0	0	0	1
B	0	0	0	0	1	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	1
F	0	1	0	0	1	0

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

Relations and sets as matrices

Row vectors as sets of states

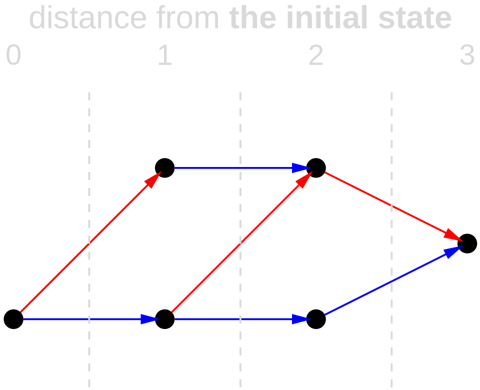
Row vectors S represent sets.
SM is the set of states reachable from S by M.

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

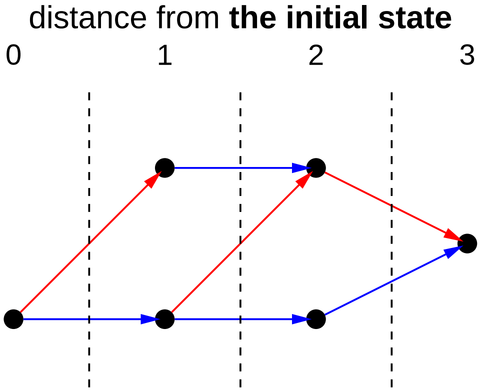
A simple planning algorithm

- 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.

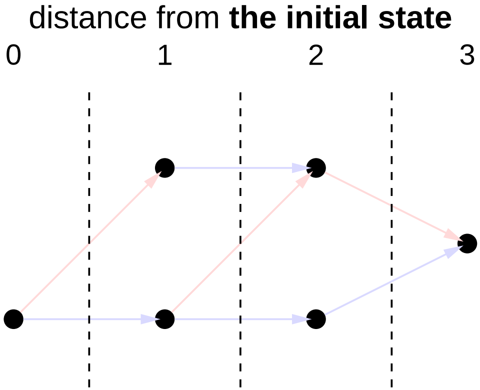
A simple planning algorithm
Idea



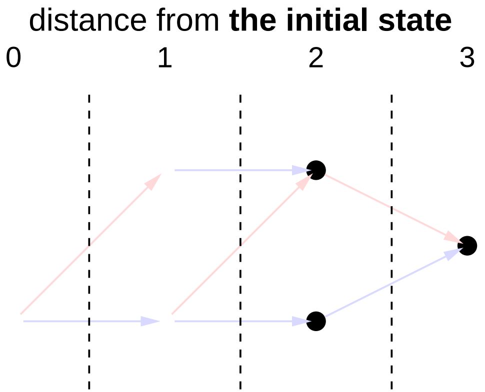
A simple planning algorithm
Idea



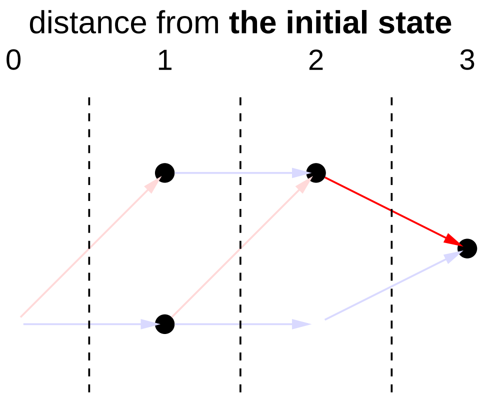
A simple planning algorithm
Idea



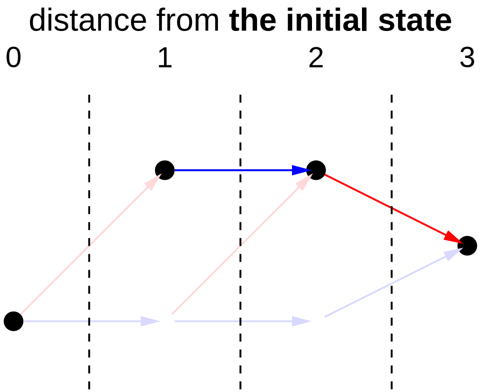
A simple planning algorithm
Idea



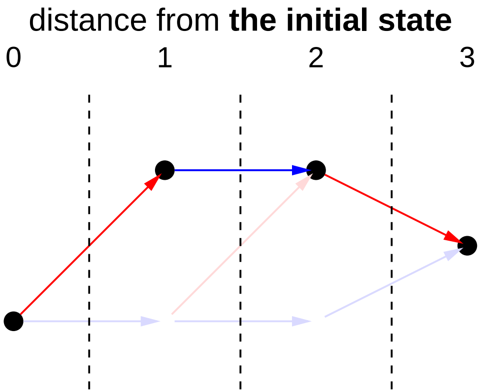
A simple planning algorithm
Idea



A simple planning algorithm
Idea



A simple planning algorithm
Idea

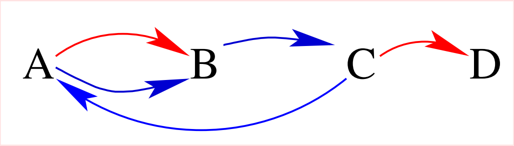


Transition systems
Definition
Example
Matrices
Reachability
Algorithm
Succinct TS

A simple planning algorithm

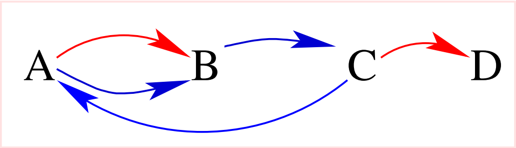
1. Compute the matrices $R_0, R_1, R_2, \ldots, R_n$ representing reachability with $0, 1, 2, \ldots, n$ steps with all actions.
2. Find the smallest i such that a goal state s_g is reachable from the initial state according to R_i.
3. Find an action (the last action of the plan) by which s_g is reached 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



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

Example



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

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

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. $\Longrightarrow$ A state is a valuation of state variables.
- Represent actions in terms of changes to the state variables.

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.

Blocks world with state variables

State variables:

LOCATIONofA : $\{B, C, \text{TABLE}\}$

LOCATIONofB : $\{A, C, \text{TABLE}\}$

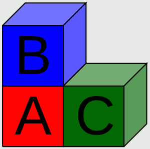
LOCATIONofC : $\{A, B, \text{TABLE}\}$

Example

$s(\text{LOCATIONofA}) = \text{TABLE}$

$s(\text{LOCATIONofB}) = A$

$s(\text{LOCATIONofC}) = \text{TABLE}$

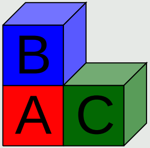


Not all valuations correspond to an intended blocks world state, e.g. s such that $s(LOCATIONofA) = B$ and $s(LOCATIONofB) = A$.

Blocks world with Boolean state variables

Example

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 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 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 A is a function $v : A \to \{0, 1\}$. Define the notation $v \models \phi$ for valuations v and formulae ϕ 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. $v \models \phi \vee \phi'$ 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. $v \not\models \bot$

Propositional logic
Some terminology

- 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 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

sets	**formulae**
those $\frac{2^n}{2}$ states in which a is true	$a \in A$
$E \cup F$	$E \vee F$
$E \cap F$	$E \wedge F$
$E \backslash F$ (set difference)	$E \wedge \neg F$
$\overline{E}$ (complement)	$\neg E$
the empty set $\emptyset$	$\bot$
the universal set	$\top$
question about sets	**question about formulae**
$E \subseteq F$?	$E \models F$?
$E \subset F$?	$E \models F$ and $F \not\models E$?
$E = F$?	$E \models F$ 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. $e_1 \wedge \cdots \wedge e_n$ 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 and e is an effect.

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

Effects

Meaning of conditional effects $\rhd$

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

Example

Increment 4-bit numbers $b_3 b_2 b_1 b_0$.

$$
\begin{gathered}
(\neg b_0 \rhd b_0) \wedge \\
((\neg b_1 \wedge b_0) \rhd (b_1 \wedge \neg b_0)) \wedge \\
((\neg b_2 \wedge b_1 \wedge b_0) \rhd (b_2 \wedge \neg b_1 \wedge \neg b_0)) \wedge \\
((\neg b_3 \wedge b_2 \wedge b_1 \wedge b_0) \rhd (b_3 \wedge \neg b_2 \wedge \neg b_1 \wedge \neg b_0))
\end{gathered}
$$

AI Planning

Transition systems

Succinct TS
State variables
Logic
Operators
Schemata

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.

$$\langle Aclear \wedge AonT \wedge Bclear, \;\; AonB \wedge \neg AonT \wedge \neg Bclear \rangle$$
$$\langle Aclear \wedge AonT \wedge Cclear, \;\; AonC \wedge \neg AonT \wedge \neg Cclear \rangle$$
$$\vdots$$
$$\langle Aclear \wedge AonB, \;\; AonT \wedge \neg AonB \wedge \neg AonC \rangle$$
$$\langle Aclear \wedge AonC, \;\; AonT \wedge \neg AonB \wedge \neg AonC \rangle$$
$$\langle Bclear \wedge BonA, \;\; BonT \wedge \neg BonA \wedge Aclear \rangle$$
$$\langle Bclear \wedge BonC, \;\; BonT \wedge \neg BonC \wedge Cclear \rangle$$
$$\vdots$$

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. $[e_1 \wedge \cdots \wedge e_n]_s = [e_1]_s \cup \ldots \cup [e_n]_s$.
3. $[c \rhd e]_s = [e]_s$ if $s \models c$ and $[c \rhd e]_s = \emptyset$ otherwise.

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 $\mathit{app}_o(s)$ 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 $\langle a, \neg a \wedge (\neg c \rhd \neg b) \rangle$ and a state s such that $s \models a \wedge b \wedge c$.

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

Hence $\mathit{app}_{\langle a, \neg a \wedge (\neg c \rhd \neg b) \rangle}(s) \models \neg a \wedge b \wedge c$.

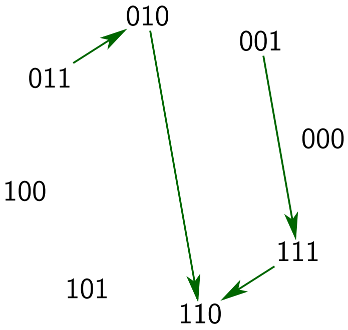
Operators

Example

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

An operator is

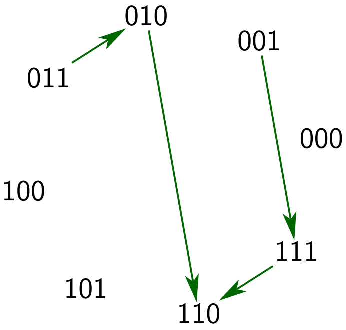
$$\langle (b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge c),$$
$$((b \wedge c) \rhd \neg c)$$
$$\wedge (\neg b \rhd (a \wedge b))$$
$$\wedge (\neg c \rhd a) \rangle$$



Operators
Example

The corresponding matrix is

	000	001	010	011	100	101	110	111
000	0	0	0	0	0	0	0	0
001	0	0	0	0	0	0	0	1
010	0	0	0	0	0	0	1	0
011	0	0	1	0	0	0	0	0
100	0	0	0	0	0	0	0	0
101	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	1	0



Succinct transition systems
Deterministic case

Definition

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.

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
   $R(o) = \{(s, s') \in S \times S | s' = \mathit{app}_o(s)\}$, 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

$$x \in \{\text{car1}, \text{car2}\}$$
$$y_1 \in \{\text{Freiburg}, \text{Strassburg}\},$$
$$y_2 \in \{\text{Freiburg}, \text{Strassburg}\}, y_1 \neq y_2$$

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

corresponds to the operators

$$\langle \text{in}(\text{car1}, \text{Freiburg}), \text{in}(\text{car1}, \text{Strassburg}) \wedge \neg\text{in}(\text{car1}, \text{Freiburg}) \rangle,$$
$$\langle \text{in}(\text{car1}, \text{Strassburg}), \text{in}(\text{car1}, \text{Freiburg}) \wedge \neg\text{in}(\text{car1}, \text{Strassburg}) \rangle,$$
$$\langle \text{in}(\text{car2}, \text{Freiburg}), \text{in}(\text{car2}, \text{Strassburg}) \wedge \neg\text{in}(\text{car2}, \text{Freiburg}) \rangle,$$
$$\langle \text{in}(\text{car2}, \text{Strassburg}), \text{in}(\text{car2}, \text{Freiburg}) \wedge \neg\text{in}(\text{car2}, \text{Strassburg}) \rangle$$

Schematic operators: quantification

Existential quantification (for formulae only)

Finite disjunctions $\phi(a_1) \vee \cdots \vee \phi(a_n)$ represented as $\exists x \in \{a_1, \ldots, a_n\}\phi(x)$.

Universal quantification (for formulae and effects)

Finite conjunctions $\phi(a_1) \wedge \cdots \wedge \phi(a_n)$ represented as $\forall x \in \{a_1, \ldots, a_n\}\phi(x)$.

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$)$.

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

```
(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

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)
               )
```

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)))
```

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)

```
(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) (clea
         (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 ham1.pddl
ff: parsing domain file, domain ’HAMILTONIAN-CYCLE’ de
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)))
)
```