Principles of AI Planning
2. Transition systems and planning tasks

Transition systems

Definition (transition system)
A transition system is a 5-tuple $\mathcal{T} = (S, L, T, s_0, S_*)$ where
- $S$ is a finite set of states,
- $L$ is a finite set of (transition) labels,
- $T \subseteq S \times L \times S$ is the transition relation,
- $s_0 \in S$ is the initial state, and
- $S_* \subseteq S$ is the set of goal states.

We say that $\mathcal{T}$ has the transition $\langle s, \ell, s' \rangle$ if $\langle s, \ell, s' \rangle \in T$.
We also write this $s \overset{\ell}{\rightarrow} s'$, or $s \rightarrow s'$ when not interested in $\ell$.

Note: Transition systems are also called state spaces.

Transition systems: example

Transition systems are often depicted as directed arc-labeled graphs with marks to indicate the initial state and goal states.
## Transition system terminology

We use common graph theory terms for transition systems:

- **\( s' \) successor of \( s \)** if \( s \rightarrow s' \)
- **\( s \) predecessor of \( s' \)** if \( s' \rightarrow s \)
- **\( s' \) reachable** from \( s \) if there exists a sequence of transitions
  \[ s^0 \stackrel{\ell_1}{\rightarrow} s^1, \ldots, s^{n-1} \stackrel{\ell_n}{\rightarrow} s^n \text{ s.t. } s^0 = s \text{ and } s^n = s' \]
  - **Note:** \( n = 0 \) possible; then \( s = s' \)
  - \( s^0, \ldots, s^{n-1} \) is called **path** from \( s \) to \( s' \)
  - \( s^0, \ldots, s^n \) is also called **path** from \( s \) to \( s' \)
  - **length** of that path is \( n \)
- **additional terms:** strongly connected, weakly connected, strong/weak connected components, \ldots

---

## Deterministic transition systems

**Definition (deterministic transition system)**

A transition system with transitions \( T \) is called **deterministic** if for all states \( s \) and labels \( \ell \), there is at most one state \( s' \) with \( s \stackrel{\ell}{\rightarrow} s' \).

**Example:** previously shown transition system

---

## Running example: blocks world

- Throughout the course, we will often use the **blocks world** domain as an example.
- In the blocks world, a number of differently coloured blocks are arranged on our table.
- Our job is to rearrange them according to a given goal.
**Blocks world rules**

Location on the table does not matter.

\[
\begin{array}{c|c}
\text{Location on a block does not matter.} & \\
\end{array}
\]

**Blocks world computational properties**

<table>
<thead>
<tr>
<th>blocks</th>
<th>states</th>
<th>blocks</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
<td>58941091</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11</td>
<td>824073141</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>1247016233</td>
</tr>
<tr>
<td>4</td>
<td>73</td>
<td>13</td>
<td>202976401213</td>
</tr>
<tr>
<td>5</td>
<td>501</td>
<td>14</td>
<td>3535017524403</td>
</tr>
<tr>
<td>6</td>
<td>4051</td>
<td>15</td>
<td>65573803186921</td>
</tr>
<tr>
<td>7</td>
<td>37633</td>
<td>16</td>
<td>1290434218669921</td>
</tr>
<tr>
<td>8</td>
<td>394353</td>
<td>17</td>
<td>26846616451246353</td>
</tr>
<tr>
<td>9</td>
<td>4596553</td>
<td>18</td>
<td>588633468315403843</td>
</tr>
</tbody>
</table>

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

- Classical (i.e., deterministic) planning is in essence the problem of finding solutions in huge transition systems.
- The transition systems we are usually interested in are too large to explicitly enumerate all states or transitions.
- Hence, the input to a planning algorithm must be given in a more concise form.
- In the rest of chapter, we discuss how to represent planning tasks in a suitable way.

State variables

How to represent huge state sets without enumerating them?
- Represent different aspects of the world in terms of different state variables
  - A state is a valuation of state variables
- \( n \) state variables with \( m \) possible values each induce \( m^n \) different states
  - Exponentially more compact than "flat" representations
- Example: \( n \) variables suffice for blocks world with \( n \) blocks

Blocks world with finite-domain state variables

Describe blocks world state with three state variables:
- \( \text{location-of-A}: \{B, C, \text{table}\} \)
- \( \text{location-of-B}: \{A, C, \text{table}\} \)
- \( \text{location-of-C}: \{A, B, \text{table}\} \)

Example

\[
\begin{align*}
  s(\text{location-of-A}) &= \text{table} \\
  s(\text{location-of-B}) &= A \\
  s(\text{location-of-C}) &= \text{table}
\end{align*}
\]

Not all valuations correspond to intended blocks world states.
Example: \( s \) with \( s(\text{location-of-A}) = B, s(\text{location-of-B}) = A \).
Problem:
- How to succinctly represent transitions and goal states?

Idea: Use propositional logic
- state variables: propositional variables (0 or 1)
- goal states: defined by a propositional formula
- transitions: defined by actions given by
  - precondition: when is the action applicable?
  - effect: how does it change the valuation?

Note: general finite-domain state variables can be compactly encoded as Boolean variables

Boolean state variables

Blocks world with Boolean state variables

Example

\[
\begin{align*}
s(A-on-B) &= 0 \\
s(A-on-C) &= 0 \\
s(A-on-table) &= 1 \\
s(B-on-A) &= 1 \\
s(B-on-C) &= 0 \\
s(B-on-table) &= 0 \\
s(C-on-A) &= 0 \\
s(C-on-B) &= 0 \\
s(C-on-table) &= 1
\end{align*}
\]

Syntax of propositional logic

Definition (propositional formula)
Let \( A \) be a set of atomic propositions (here: state variables).
The propositional formulae over \( A \) are constructed by finite application of the following rules:
- \( \top \) and \( \bot \) are propositional formulae (truth and falsity).
- For all \( a \in A \), \( a \) is a propositional formula (atom).
- If \( \phi \) is a propositional formula, then so is \( \neg \phi \) (negation)
- If \( \phi \) and \( \psi \) are propositional formulas, then so are \( (\phi \lor \psi) \) (disjunction) and \((\phi \land \psi)\) (conjunction).

Note: We often omit the word “propositional”.

Propositional logic conventions

Abbreviations:
- \( (\varphi \rightarrow \psi) \) is short for \( (\neg \varphi \lor \psi) \) (implication)
- \( (\varphi \leftrightarrow \psi) \) is short for \( ((\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi)) \) (equivalence)
- parentheses omitted when not necessary
- \( (\neg) \) binds more tightly than binary connectives
- \( (\land) \) binds more tightly than \((\lor)\) than \((\rightarrow)\) than \((\leftrightarrow)\)
Semantics of propositional logic

**Definition (propositional valuation)**

A *valuation* of propositions $A$ is a function $v : A \rightarrow \{0, 1\}$.

Define the notation $v \models \varphi$ ($v$ satisfies $\varphi$; $v$ is a model of $\varphi$; $\varphi$ is true under $v$) for valuations $v$ and formulae $\varphi$ by

- $v \models \top$
- $v \not\models \bot$
- $v \models a$ iff $v(a) = 1$, for $a \in A$.
- $v \models \neg \varphi$ iff $v \not\models \varphi$
- $v \models \varphi \lor \psi$ iff $v \models \varphi$ or $v \models \psi$
- $v \models \varphi \land \psi$ iff $v \models \varphi$ and $v \models \psi$


Propositional logic terminology

- A propositional formula $\varphi$ is *satisfiable* if there is at least one valuation $v$ so that $v \models \varphi$.
- Otherwise it is *unsatisfiable*.
- A propositional formula $\varphi$ is *valid* or a tautology if $v \models \varphi$ for all valuations $v$.
- A propositional formula $\psi$ is a *logical consequence* of a propositional formula $\varphi$, written $\varphi \models \psi$, if $v \models \psi$ for all valuations $v$ with $v \models \varphi$.
- Two propositional formulae $\varphi$ and $\psi$ are *logically equivalent*, written $\varphi \equiv \psi$, if $v \models \varphi$ and $\psi \models \varphi$.

**Question:** How to phrase these in terms of models?

Propositional logic terminology (ctd.)

- 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*.
  This includes *unit clauses* / consisting of a single literal, and the empty clause $\bot$ consisting of zero literals.

**Normal forms:** NNF, CNF, DNF

October 24th, 2012 B. Nebel, R. Mattmüller – AI Planning 24 / 36

Operators

Transitions for state sets described by propositions $A$ can be concisely represented as operators or actions $\langle \chi, e \rangle$, where

- the *precondition* $\chi$ is a propositional formula over $A$ describing the set of states in which the transition can be taken (states in which a transition starts), and
- the *effect* $e$ describes how the resulting successor states are obtained from the state where the transitions is taken (where the transition goes).

Example: blocks world operators

Blocks world operators
To model blocks world operators conveniently, we use auxiliary state variables $A$-clear, $B$-clear, and $C$-clear to denote that there is nothing on top of a given block.

Then blocks world operators can be modeled as:

- $(A$-clear $\land A$-on-$T$ $\land$ $B$-clear, $A$-on-$B$ $\land$ $\neg$ $A$-on-$T$ $\land$ $\neg$ $B$-clear)
- $(A$-clear $\land$ $A$-on-$T$ $\land$ $C$-clear, $A$-on-$C$ $\land$ $\neg$ $A$-on-$T$ $\land$ $\neg$ $C$-clear)
- $(A$-clear $\land$ $A$-on-$B$, $A$-on-$T$ $\land$ $\neg$ $A$-on-$B$ $\land$ $B$-clear)
- $(A$-clear $\land$ $A$-on-$C$, $A$-on-$T$ $\land$ $\neg$ $A$-on-$C$ $\land$ $C$-clear)
- $(A$-clear $\land$ $A$-on-$B$ $\land$ $C$-clear, $A$-on-$C$ $\land$ $\neg$ $A$-on-$B$ $\land$ $B$-clear $\land$ $\neg$ $C$-clear)
- $(A$-clear $\land$ $A$-on-$C$ $\land$ $B$-clear, $A$-on-$B$ $\land$ $\neg$ $A$-on-$C$ $\land$ $C$-clear $\land$ $\neg$ $B$-clear)
- ...

Effect example

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

Example

Increment 4-bit number $b_3b_2b_1b_0$ represented as four state variables $b_0, \ldots, b_3$:

\[
\begin{align*}
(-b_3 \triangleright b_0) & \land \\
((-b_1 \land b_0) \triangleright (b_1 \land \neg b_0)) & \land \\
((-b_2 \land b_1 \land b_0) & \triangleright (b_2 \land \neg b_1 \land \neg b_0)) \land \\
((-b_3 \land b_2 \land b_1 \land b_0) & \triangleright (b_3 \land \neg b_2 \land \neg b_1 \land \neg b_0))
\end{align*}
\]

Effects (for deterministic operators)

Definition (effects)

(Deterministic) effects are recursively defined as follows:

- If $a \in A$ is a state variable, then $a$ and $\neg a$ are effects (atomic effect).
- If $e_1, \ldots, e_n$ are effects, then $e_1 \land \cdots \land e_n$ is an effect (conjunctive effect).
- If $\chi$ is a propositional formula and $e$ is an effect, then $\chi \triangleright e$ is an effect (conditional effect).

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

Operator semantics

Definition (changes caused by an operator)

For each effect $e$ and state $s$, we define the change set of $e$ in $s$, written $[e]_s$, as the following set of literals:

- $[a]_s = \{a\}$ and $[-a]_s = \{-a\}$ for atomic effects $a$, $\neg a$
- $[e_1 \land \cdots \land e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$
- $[\chi \triangleright e]_s = [e]_s$ if $s \models \chi$ and $[\chi \triangleright e]_s = \emptyset$ otherwise

Definition (applicable operators)

Operator $(\chi, e)$ is applicable in a state $s$ iff $s \models \chi$ and $[e]_s$ is consistent (i.e., does not contain two complementary literals).
Operator semantics (ctd.)

Definition (successor state)
The successor state \( \text{app}_o(s) \) of \( s \) with respect to operator \( o = (\chi, e) \) is the state \( s' \) with \( s' \models [e]_s \) and \( s'(v) = s(v) \) for all state variables \( v \) not mentioned in \( [e]_s \). This is defined only if \( o \) is applicable in \( s \).

Example
Consider the operator \( o = (a, \neg a \land (\neg c \triangleright \neg b)) \) and the state \( s = \{ a \mapsto 1, b \mapsto 1, c \mapsto 1, d \mapsto 1 \} \).
The operator is applicable because \( s \models a \) and \( [\neg a \land (\neg c \triangleright \neg b)]_s = \{ \neg a \} \) is consistent.
Applying the operator results in the successor state \( \text{app}_o(a, \neg a \land (\neg c \triangleright \neg b))(s) = \{ a \mapsto 0, b \mapsto 1, c \mapsto 1, d \mapsto 1 \} \).

Deterministic planning tasks

Definition (deterministic planning task)
A deterministic planning task is a 4-tuple \( \Pi = (A, I, O, \gamma) \) where
- \( A \) is a finite set of state variables (propositions),
- \( I \) is a valuation over \( A \) called the initial state,
- \( O \) is a finite set of operators over \( A \), and
- \( \gamma \) is a formula over \( A \) called the goal.

Note:
- When we talk about deterministic planning tasks, we usually omit the word “deterministic”.
- When we will talk about nondeterministic planning tasks later, we will explicitly qualify them as “nondeterministic”.

Mapping planning tasks to transition systems

Definition (induced transition system of a planning task)
Every planning task \( \Pi = (A, I, O, \gamma) \) induces a corresponding deterministic transition system \( \mathcal{T}(\Pi) = (S, L, T, s_0, S^\star) \):
- \( S \) is the set of all valuations of \( A \),
- \( L \) is the set of operators \( O \),
- \( T = \{ (s, o, s') | s \in S, o \text{ applicable in } s, s' = \text{app}_o(s) \} \),
- \( s_0 = I \), and
- \( S^\star = \{ s \in S | s \models \gamma \} \).
By planning, we mean the following two algorithmic problems:

**Definition (satisficing planning)**
Given: a planning task $\Pi$
Output: a plan for $\Pi$, or **unsolvable** if no plan for $\Pi$ exists

**Definition (optimal planning)**
Given: a planning task $\Pi$
Output: a plan for $\Pi$ with minimal length among all plans for $\Pi$, or **unsolvable** if no plan for $\Pi$ exists

- **Transition systems** are (typically huge) directed graphs that encode how the state of the world can change.
- **Planning tasks** are compact representations for transition systems, suitable as input for planning algorithms.
- Planning tasks are based on concepts from propositional logic, enhanced to model state change.
- States of planning tasks are propositional valuations.
- Operators of planning tasks describe when (precondition) and how (effect) to change the current state of the world.
- In **satisficing planning**, we must find a solution to planning tasks (or show that no solution exists).
- In **optimal planning**, we additionally guarantee that generated solutions are of the shortest possible length.