Principles of AI Planning
2. Transition systems and planning tasks

Transition systems

Definition (transition system)
A transition system is a 5-tuple \( \mathcal{T} = (S, L, T, s_0, S_\star) \) 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
- \( S_\star \subseteq S \) is the set of goal states.

We say that \( \mathcal{T} \) has the transition \( (s, \ell, s') \) if \( (s, \ell, s') \in T \).

We also write this \( s \xrightarrow{\ell} s' \), or \( s \rightarrow s' \) when not interested in \( \ell \).

Note: Transition systems are also called state spaces.
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

\[
\begin{align*}
    s^0 \xrightarrow{\ell_1} s^1, \ldots, s^{n-1} \xrightarrow{\ell_n} s^n
\end{align*}
\]

s.t. \( s^0 = s \) and \( s^n = s' \)

- Note: \( n = 0 \) possible; then \( s = s' \)
- \( s^0 \xrightarrow{\ell_1} s^1, \ldots, s^{n-1} \xrightarrow{\ell_n} s^n \) 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, ...

October 16th, 2017  B. Nebel, R. Mattmüller – AI Planning  5 / 34

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 \( s \xrightarrow{\ell} s' \).

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.

\[
\begin{array}{c}
\text{Original State} \\
&&\downarrow \\
\text{New State}
\end{array}
\]

Location on a block does not matter.

\[
\begin{array}{c}
\text{Original State} \\
&&\downarrow \\
\text{New State}
\end{array}
\]

Blocks world rules (ctd.)

At most one block may be below a block.

\[
\begin{array}{c}
\text{Original State} \\
&&\downarrow \\
\text{New State}
\end{array}
\]

At most one block may be on top of a block.

\[
\begin{array}{c}
\text{Original State} \\
&&\downarrow \\
\text{New State}
\end{array}
\]

Blocks world transition system for three blocks

(Transition labels omitted for clarity.)

Blocks world computational properties

- 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).
Planning tasks

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, table\}
- \( \text{location-of-B} \): \{A, C, table\}
- \( \text{location-of-C} \): \{A, B, 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 \( s(\text{location-of-A}) = B, s(\text{location-of-B}) = A \).
Boolean state variables

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

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 formulae, then so are \( \phi \lor \psi \) (disjunction) and \( \phi \land \psi \) (conjunction).

Note: We often omit the word “propositional”.

Propositional logic conventions

Abbreviations:
- \( (\phi \rightarrow \psi) \) is short for \( (\neg \phi \lor \psi) \) (implication)
- \( (\phi \leftrightarrow \psi) \) is short for \( (\neg (\phi \rightarrow \psi) \land (\psi \rightarrow \phi)) \) (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 $v \models \phi$ (v satisfies $\phi$; $v$ is a model of $\phi$; $\phi$ is true under $v$) for valuations $v$ and formulae $\phi$ by

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

Propositional logic 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 $v$.
- A propositional formula $\psi$ is a logical consequence of a propositional formula $\phi$, written $\phi \models \psi$, if $v \models \psi$ for all valuations $v$ with $v \models \phi$.
- Two propositional formulae $\phi$ and $\psi$ are logically equivalent, written $\phi \equiv \psi$, if $\phi \models \psi$ and $\psi \models \phi$.

**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 $l$ consisting of a single literal, and the empty clause $\bot$ consisting of zero literals.

Normal forms: NNF, CNF, DNF

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 transition 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)
- ...

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 $[\neg a]_s = \{\neg a\}$ 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 $s$ if $s \models \chi$ 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 \( o = \langle \chi, e \rangle \) 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 \( \langle a, \neg a \land (\neg c \triangleright \neg b) \rangle \) and the state \( s = \{ a \mapsto \top, b \mapsto \top, c \mapsto \top, d \mapsto \top \} \).

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 \( \Pi = \langle A, I, O, \gamma \rangle \) 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 \( \Pi = \langle A, I, O, \gamma \rangle \) induces a corresponding deterministic transition system \( \mathcal{T}(\Pi) = \langle S, L, T, s_0, S^* \rangle \):

- \( S \) is the set of all valuations of \( A \),
- \( L \) is the set of operators \( O \),
- \( T = \{ \langle s, o, s' \rangle \mid s \in S, o \text{ applicable in } s, s' = \text{app}_o(s) \} \),
- \( s_0 = I \), and
- \( S^* = \{ s \in S \mid s \models \gamma \} \).

Planning tasks: terminology

- Terminology for transitions systems is also applied to the planning tasks that induce them.
- For example, when we speak of the states of \( \Pi \), we mean the states of \( \mathcal{T}(\Pi) \).
- A sequence of operators that forms a goal path of \( \mathcal{T}(\Pi) \) is called a plan of \( \Pi \).
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.