Principles of AI Planning

2. Transition systems and planning tasks

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel and Robert Mattmüller

October 23rd, 2019

Transition systems

Transition systems

Definition (transition system)

A transition system is a 5-tuple $\mathscr{T} = \langle S, L, T, s_0, S_\star \rangle$ 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.

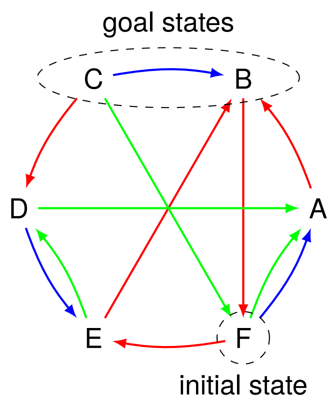
We say that $\mathscr{T}$ has the transition $\langle s, \ell, s' \rangle$ if $\langle s, \ell, s' \rangle \in T$.

We also write this $s \xrightarrow{\ell} s'$, or $s \to s'$ when not interested in ℓ.

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 $s \to s'$
- s predecessor of s' if $s \to s'$
- s' reachable from s if there exists a sequence of transitions
  $s^0 \xrightarrow{\ell_1} s^1, \dots, s^{n-1} \xrightarrow{\ell_n} s^n$ s.t. $s^0 = s$ and $s^n = s'$
  - Note: $n = 0$ possible; then $s = s'$
  - $s^0 \xrightarrow{\ell_1} s^1, \dots, s^{n-1} \xrightarrow{\ell_n} s^n$ is called path from s to s'
  - $s^0, \dots, s^n$ is also called path from s to s'
  - length of that path is n
- additional terms: strongly connected, weakly connected, strong/weak connected components, ...

Transition system terminology (ctd.)

Some additional terminology:

- s' reachable (without reference state) means reachable from initial state s_0
- solution or goal path from s: path from s to some $s' \in S_\star$
  - if s is omitted, $s = s_0$ is implied
- transition system solvable if a goal path from s_0 exists

Deterministic transition systems

Definition (deterministic transition system)

A transition system with transitions T is called deterministic if for all states s and labels ℓ, there is at most one state s' with $s \xrightarrow{\ell} s'$.

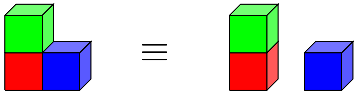
Example: previously shown transition system

Running example: blocks world

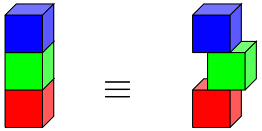
- Throughout the course, we will often use the blocks world domain as an example.
- In the blocks world, a number of differently coloured blocks are arranged on our table.
- Our job is to rearrange them according to a given goal.

Blocks world rules

Location on the table does not matter.

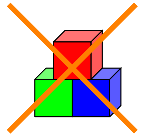


Location on a block does not matter.

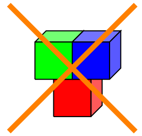


Blocks world rules (ctd.)

At most one block may be below a block.

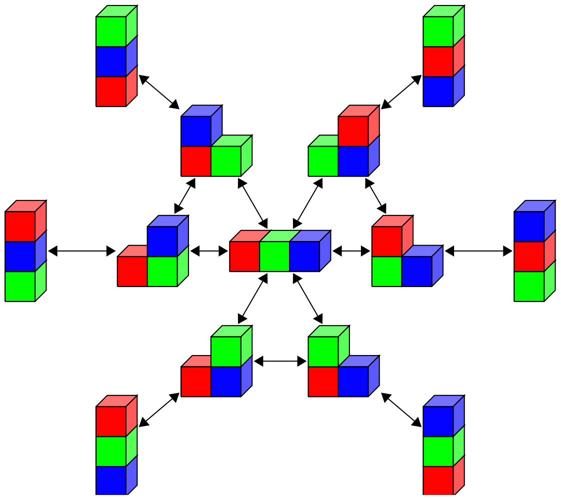


At most one block may be on top of a block.



Blocks world transition system for three blocks

(Transition labels omitted for clarity.)



Blocks world computational properties

blocks	states	blocks	states
1	1	10	58941091
2	3	11	824073141
3	13	12	12470162233
4	73	13	202976401213
5	501	14	3535017524403
6	4051	15	65573803186921
7	37633	16	1290434218669921
8	394353	17	26846616451246353
9	4596553	18	588633468315403843

- 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

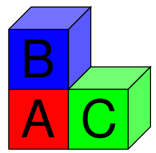
Blocks world with finite-domain state variables

Describe blocks world state with three state variables:

- *location-of-A*: $\{B, C, \text{table}\}$
- *location-of-B*: $\{A, C, \text{table}\}$
- *location-of-C*: $\{A, B, \text{table}\}$

Example

s(*location-of-A*) = table
s(*location-of-B*) = A
s(*location-of-C*) = table



Not all valuations correspond to intended blocks world states.
Example: *s* with *s*(*location-of-A*) = B, *s*(*location-of-B*) = A.

Boolean state variables

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

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?

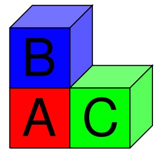
Note: general finite-domain state variables can be compactly encoded as Boolean variables

Blocks world with Boolean state variables

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

Example

$s(A\text{-}on\text{-}B) = 0$
$s(A\text{-}on\text{-}C) = 0$
$s(A\text{-}on\text{-}table) = 1$
$s(B\text{-}on\text{-}A) = 1$
$s(B\text{-}on\text{-}C) = 0$
$s(B\text{-}on\text{-}table) = 0$
$s(C\text{-}on\text{-}A) = 0$
$s(C\text{-}on\text{-}B) = 0$
$s(C\text{-}on\text{-}table) = 1$



Syntax of propositional logic

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

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 φ is a propositional formula, then so is $\neg\varphi$ (negation)
- If φ and ψ are propositional formulas, then so are $(\varphi \vee \psi)$ (disjunction) and $(\varphi \wedge \psi)$ (conjunction).

Note: We often omit the word “propositional”.

Propositional logic conventions

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

Abbreviations:

- $(\varphi \to \psi)$ is short for $(\neg\varphi \vee \psi)$ (implication)
- $(\varphi \leftrightarrow \psi)$ is short for $((\varphi \to \psi) \wedge (\psi \to \varphi))$ (equivalence)
- parentheses omitted when not necessary
- $(\neg)$ binds more tightly than binary connectives
- $(\wedge)$ binds more tightly than $(\vee)$ than $(\to)$ than $(\leftrightarrow)$

Semantics of propositional logic

Definition (propositional valuation)

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

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

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

Propositional logic terminology

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

- $\langle \textit{A-clear} \wedge \textit{A-on-T} \wedge \textit{B-clear},\ \textit{A-on-B} \wedge \neg \textit{A-on-T} \wedge \neg \textit{B-clear} \rangle$
- $\langle \textit{A-clear} \wedge \textit{A-on-T} \wedge \textit{C-clear},\ \textit{A-on-C} \wedge \neg \textit{A-on-T} \wedge \neg \textit{C-clear} \rangle$
- $\langle \textit{A-clear} \wedge \textit{A-on-B},\ \textit{A-on-T} \wedge \neg \textit{A-on-B} \wedge \textit{B-clear} \rangle$
- $\langle \textit{A-clear} \wedge \textit{A-on-C},\ \textit{A-on-T} \wedge \neg \textit{A-on-C} \wedge \textit{C-clear} \rangle$
- $\langle \textit{A-clear} \wedge \textit{A-on-B} \wedge \textit{C-clear},\ \textit{A-on-C} \wedge \neg \textit{A-on-B} \wedge \textit{B-clear} \wedge \neg \textit{C-clear} \rangle$
- $\langle \textit{A-clear} \wedge \textit{A-on-C} \wedge \textit{B-clear},\ \textit{A-on-B} \wedge \neg \textit{A-on-C} \wedge \textit{C-clear} \wedge \neg \textit{B-clear} \rangle$
- ...

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 $e_1 \wedge \cdots \wedge e_n$ is an effect (conjunctive effect).
  The special case with $n = 0$ is the empty effect $\top$.
- If χ is a propositional formula and e is an effect, then $\chi \rhd e$ is an effect (conditional effect).

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

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

Effect example

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

Example

Increment 4-bit number $b_3 b_2 b_1 b_0$ represented as four state variables $b_0, \ldots, b_3$:

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

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

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, $\neg a$
- $[e_1 \wedge \cdots \wedge e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$
- $[\chi \rhd e]_s = [e]_s$ if $s \models \chi$ and $[\chi \rhd e]_s = \emptyset$ otherwise

Definition (applicable operators)

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

Transition systems
Planning tasks
State variables
Logic
Operators
Tasks
Summary

Operator semantics (ctd.)

Definition (successor state)

The successor state $app_o(s)$ 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 $\langle a, \neg a \wedge (\neg c \rhd \neg b) \rangle$ 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 $[\neg a \wedge (\neg c \rhd \neg b)]_s = \{\neg a\}$ is consistent.
Applying the operator results in the successor state $app_{\langle a, \neg a \wedge (\neg c \rhd \neg b) \rangle}(s) = \{a \mapsto 0, b \mapsto 1, c \mapsto 1, d \mapsto 1\}$.

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
- γ 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 $\mathcal{T}(\Pi) = \langle S, L, T, s_0, S_\star \rangle$:

- S is the set of all valuations of A,
- L is the set of operators O,
- $T = \{\langle s, o, s' \rangle \mid s \in S,\ o \text{ applicable in } s,\ s' = app_o(s)\}$,
- $s_0 = I$, and
- $S_\star = \{s \in S \mid s \models \gamma\}$

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 Π, 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 Π.

Planning

By planning, we mean the following two algorithmic problems:

Definition (satisficing planning)

Given: a planning task Π
Output: a plan for Π, or **unsolvable** if no plan for Π exists

Definition (optimal planning)

Given: a planning task Π
Output: a plan for Π with minimal length among all plans for Π, or **unsolvable** if no plan for Π exists

Summary

- 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.