Principles of AI Planning

3. Deterministic planning tasks

Malte Helmert

Albert-Ludwigs-Universität Freiburg

October 24th, 2008

Principles of AI Planning

October 24th, 2008 — 3. Deterministic planning tasks

Succinct representation of transition systems

- More compact representation of actions than as relations is often
  - possible because of symmetries and other regularities,
  - unavoidable because the relations are too big.
- Represent different aspects of the world in terms of different state variables. $\rightsquigarrow$ 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

$hour$: $\{0, \ldots, 23\} = 13$
$minute$: $\{0, \ldots, 59\} = 55$
$location$: $\{51, 52, 82, 101, 102\} = 101$
$weather$: $\{\text{sunny}, \text{cloudy}, \text{rainy}\} = \text{cloudy}$
$holiday$: $\{\text{T}, \text{F}\} = \text{F}$

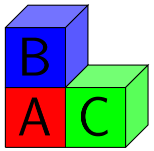
- Any n-valued state variable can be replaced by $\lceil \log_2 n \rceil$ Boolean (2-valued) state variables.
- Actions change the values of the state variables.

Blocks world with state variables

State variables:
location-of-A: $\{B, C, table\}$
location-of-B: $\{A, C, table\}$
location-of-C: $\{A, B, table\}$

Example

$$s(\textit{location-of-A}) = \text{table}$$
$$s(\textit{location-of-B}) = \text{A}$$
$$s(\textit{location-of-C}) = \text{table}$$

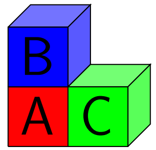


Not all valuations correspond to an intended blocks world state, e. g. s such that $s(\textit{location-of-A}) = \text{B}$ and $s(\textit{location-of-B}) = \text{A}$.

Blocks world with Boolean state variables

Example

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



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)*: propositional logic
- logical connectives $\approx$ set-theoretical operations

Syntax of 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 $\bot$ 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 $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

Semantics of propositional logic

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 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 with $v \models \phi'$.
- Two propositional formulae ϕ and ϕ' are logically equivalent, written $\phi \equiv \phi'$, if $\phi \models \phi'$ and $\phi' \models \phi$.

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

Formulae vs. sets

sets	formulae
those $\frac{2^n}{2}$ states in which a is true	$a \in A$
$E \cup F$	$E \vee F$
$E \cap F$	$E \wedge F$
$E \setminus F$ (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 for a state set with propositional state variables A can be concisely represented as operators $\langle c, e \rangle$ where

- the precondition c is a propositional formula over A describing the set of states in which the action can be taken (*states in which an arrow starts*), and
- the effect e describes the successor states of states in which the action can be taken (*where the arrows go*). Effect descriptions are procedural: how do the values of the state variable change?

Effects (for deterministic operators)

Definition (effects)

(Deterministic) effects are recursively defined as follows:

1. If $a \in A$ is a state variable, then a and $\neg a$ are effects (atomic effects).
2. If $e_1, \ldots, e_n$ are effects, then $e_1 \wedge \cdots \wedge e_n$ is an effect (conjunctive effects). The special case with $n = 0$ is the empty conjunction $\top$.
3. If c is a propositional formula and e is an effect, then $c \rhd e$ is an effect (conditional effects).

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

Effect example

$c \rhd e$ means that change e takes place if c 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.

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

Blocks world operators

In addition to state variables likes *A-on-T* and *B-on-C*, for convenience we also use state variables *A-clear*, *B-clear*, and *C-clear* to denote that there is nothing on the block in question.

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

Operator semantics

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:

1. $[a]_s = \{a\}$ and $[\neg a]_s = \{\neg a\}$ for atomic effects a, $\neg a$
2. $[e_1 \wedge \cdots \wedge e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$
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.

Operator semantics (ctd.)

Definition (successor state)

The successor state $app_o(s)$ of s with respect to operator $o = \langle c, e\rangle$ 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 $\Pi = \langle A, I, O, G\rangle$ where

- A is a finite set of state variables,
- I is an initial state over A,
- O is a finite set of operators over A, and
- G is a formula over A describing the goal states.

Note: We will omit the word "deterministic" where it is clear from context.

Mapping planning tasks to transition systems

From every deterministic planning task $\Pi = \langle A, I, O, G\rangle$ we can produce a corresponding transition system $\mathcal{T}(\Pi) = \langle S, I, O', G'\rangle$:

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

Equivalence of operators and effects

Definition (equivalent effects)

Two effects e and e' over state variables A are equivalent, written $e \equiv e'$, if for all states s over A, $[e]_s = [e']_s$.

Definition (equivalent operators)

Two operators o and o' over state variables A are equivalent, written $o \equiv o'$, if they are applicable in the same states, and for all states s where they are applicable, $app_o(s) = app_{o'}(s)$.

Theorem

Let $o = \langle c, e \rangle$ and $o' = \langle c', e' \rangle$ be operators with $c \equiv c'$ and $e \equiv e'$. Then $o \equiv o'$.

Note: The converse is not true. (Why not?)

Equivalence transformations for effects

$$
\begin{aligned}
e_1 \wedge e_2 &\equiv e_2 \wedge e_1 && (1)\\
(e_1 \wedge e_2) \wedge e_3 &\equiv e_1 \wedge (e_2 \wedge e_3) && (2)\\
\top \wedge e &\equiv e && (3)\\
c \rhd e &\equiv c' \rhd e \quad \text{if } c \equiv c' && (4)\\
\top \rhd e &\equiv e && (5)\\
\bot \rhd e &\equiv \top && (6)\\
c_1 \rhd (c_2 \rhd e) &\equiv (c_1 \wedge c_2) \rhd e && (7)\\
c \rhd (e_1 \wedge \cdots \wedge e_n) &\equiv (c \rhd e_1) \wedge \cdots \wedge (c \rhd e_n) && (8)\\
(c_1 \rhd e) \wedge (c_2 \rhd e) &\equiv (c_1 \vee c_2) \rhd e && (9)
\end{aligned}
$$

Normal form for effects

Similarly to normal forms in propositional logic (DNF, CNF, NNF, ...) we can define a normal form for effects.
This is useful because algorithms (and proofs) then only need to deal with effects in normal form.

- Nesting of conditionals, as in $a \rhd (b \rhd c)$, can be eliminated.
- Effects e within a conditional effect $\phi \rhd e$ can be restricted to atomic effects (a or $\neg a$).

Transformation to normal form only gives a small polynomial size increase.
Compare: transformation to CNF or DNF may increase formula size exponentially.

Normal form for operators and effects

Definition

An operator $\langle c, e \rangle$ is in normal form if for all occurrences of $c' \rhd e'$ in e the effect e' is either a or $\neg a$ for some $a \in A$, and there is at most one occurrence of any atomic effect in e.

Theorem

For every operator there is an equivalent one in normal form.

Proof is constructive: we can transform any operator into normal form using the equivalence transformations for effects.

Normal form example

Example

$$(a \rhd (b \land (c \rhd (\neg d \land e)))) \land (\neg b \rhd e)$$

transformed to normal form is

$$(a \rhd b) \land ((a \land c) \rhd \neg d) \land ((\neg b \lor (a \land c)) \rhd e)$$

STRIPS operators

Definition

An operator $\langle c, e\rangle$ is a STRIPS operator if

1. c is a conjunction of literals, and
2. e is a conjunction of atomic effects.

Hence every STRIPS operator is of the form

$$\langle l_1 \land \cdots \land l_n, \quad l'_1 \land \cdots \land l'_m \rangle$$

where l_i are literals and l'_j are atomic effects.

Note: Many texts also require that all literals in c are positive.

STRIPS

STanford Research Institute Planning System
(Fikes & Nilsson, 1971)

Why STRIPS is interesting

- STRIPS operators are particularly simple, yet expressive enough to capture general planning problems.
- In particular, STRIPS planning is no easier than general planning problems.
- Most algorithms in the planning literature are only presented for STRIPS operators (generalization is often, but not always, obvious).

Transformation to STRIPS

- Not every operator is equivalent to a STRIPS operator.
- However, each operator can be transformed into a set of STRIPS operators whose "combination" is equivalent to the original operator. (How?)
- However, this transformation may exponentially increase the number of required operators. There are planning tasks for which such a blow-up is unavoidable.
- There are polynomial transformations of planning tasks to STRIPS, but these do not preserve the structure of the transition system (e. g., length of shortest plans may change).