# Planning and Optimization
## A4. Planning Tasks

Malte Helmert and Gabriele Röger

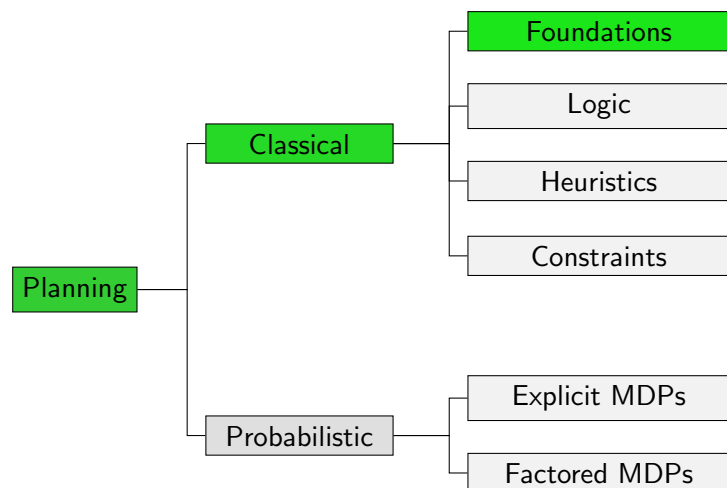Universität Basel

September 23, 2020

---

A4.1 State Variables

A4.2 Operators

A4.3 Planning Tasks

A4.4 Summary

---

## Content of this Course

---

# A4.1 State Variables

## State Variables

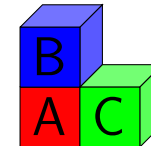How to specify huge transition systems
without enumerating the states?

- ▶ represent different aspects of the world
  in terms of different state variables (Boolean or finite domain)
- ▶ individual state variables induce atomic propositions
  ⤳ a state is a valuation of state variables
- ▶ $n$ Boolean state variables induce $2^n$ states
  ⤳ exponentially more compact than "flat" representations

Example: $O(n^2)$ Boolean variables or $O(n)$ finite-domain variables
with domain size $O(n)$ suffice for blocks world with $n$ blocks

---

## Blocks World State with Propositional Variables

**Example**

$$s(A\text{-}on\text{-}B) = F$$
$$s(A\text{-}on\text{-}C) = F$$
$$s(A\text{-}on\text{-}table) = T$$
$$s(B\text{-}on\text{-}A) = T$$
$$s(B\text{-}on\text{-}C) = F$$
$$s(B\text{-}on\text{-}table) = F$$
$$s(C\text{-}on\text{-}A) = F$$
$$s(C\text{-}on\text{-}B) = F$$
$$s(C\text{-}on\text{-}table) = T$$

Note: it may be useful to add auxiliary state variables like $A$-clear.
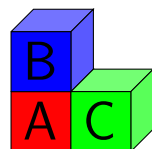
---

## Blocks World State with Finite-Domain Variables

**Example**

Use three finite-domain state variables:

- ▶ below-a: $\{b, c, table\}$
- ▶ below-b: $\{a, c, table\}$
- ▶ below-c: $\{a, b, table\}$

$$s(below\text{-}a) = table$$
$$s(below\text{-}b) = a$$
$$s(below\text{-}c) = table$$

⤳ $3^3 = 27$ states

Note: it may be useful to add auxiliary state variables like above-a.

---

## Propositional State Variables

**Definition (Propositional State Variable)**
A propositional state variable is a symbol $X$.

Let $V$ be a finite set of propositional state variables.

A state $s$ over $V$ is a valuation for $V$, i.e.,
a truth assignment $s : V \rightarrow \{\mathbf{T}, \mathbf{F}\}$.

A formula over $V$ is a propositional logic formula using $V$
as the set of atomic propositions.

## Propositional State Variables

**Definition (Finite-Domain State Variable)**

A finite-domain state variable is a symbol $v$ with an associated domain $\text{dom}(v)$, which is a finite non-empty set of values.

Let $V$ be a finite set of finite-domain state variables.

A state $s$ over $V$ is an assignment $s : V \to \bigcup_{v \in V} \text{dom}(v)$ such that $s(v) \in \text{dom}(v)$ for all $v \in V$.

A formula over $V$ is a propositional logic formula whose atomic propositions are of the form $v = d$ where $v \in V$ and $d \in \text{dom}(v)$.

Slightly extending propositional logic, we treat states $s$ over finite-domain variables as logical valuations where $s \models v = d$ iff $s(v) = d$.

## State Variables: Either/Or

- ► State variables are the basis of compact descriptions of transition systems.
- ► For a given transition system, we will either use propositional or finite-domain state variables. We will not mix them.
- ► However, finite-domain variables can have any finite domain including the domain $\{\mathbf{T}, \mathbf{F}\}$, so are in some sense a proper generalization of propositional state variables.

## From State Variables to Succinct Transition Systems

Problem:
- ► How to succinctly represent transitions and goal states?

Idea: Use formulas to describe sets of states
- ► states: all assignments to the state variables
- ► goal states: defined by a formula
- ► transitions: defined by operators (see following section)

# A4.2 Operators

## Syntax of Operators

> **Definition (Operator)**
>
> An operator $o$ over state variables $V$ is an object
> with three properties:
> - a precondition $pre(o)$, a formula over $V$
> - an effect $eff(o)$ over $V$, defined on the following slides
> - a cost $cost(o) \in \mathbb{R}_0^+$

Notes:
- Operators are also called actions.
- Operators are often written as triples $\langle pre(o), eff(o), cost(o) \rangle$.
- This can be abbreviated to pairs $\langle pre(o), eff(o) \rangle$
  when the cost of the operator is irrelevant.

## Operators: Intuition

Intuition for operators $o$:
- The operator precondition describes the set of states
  in which a transition labeled with $o$ can be taken.
- The operator effect describes how taking such a transition
  changes the state.
- The operator cost describes the cost of taking a transition
  labeled with $o$.

## Syntax of Effects

> **Definition (Effect)**
>
> Effects over state variables $V$ are inductively defined as follows:
> - If $v \in V$ is a propositional state variable,
>   then $v$ and $\neg v$ are effects (atomic effect).
> - If $v \in V$ is a finite-domain state variable and $d \in \text{dom}(v)$,
>   then $v := d$ is an effect (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 $\chi$ is a formula over $V$ and $e$ is an effect,
>   then $(\chi \rhd e)$ is an effect (conditional effect).

Parentheses can be omitted when this does not cause ambiguity.

## Effects: Intuition

Intuition for effects:
- Atomic effects can be understood as assignments
  that update the value of a state variable.
  - For propositional state variables, $v$ means "$v := \mathsf{T}$"
    and $\neg v$ means "$v := \mathsf{F}$".
- A conjunctive effect $e = (e_1 \wedge \cdots \wedge e_n)$ means that
  all subeffects $e_1, \ldots, e_n$ take place simultaneously.
- A conditional effect $e = (\chi \rhd e')$ means that subeffect $e'$
  takes place iff $\chi$ is true in the state where $e$ takes place.

## Semantics of Effects

> **Definition (Effect Condition for an Effect)**
> Let $e$ be an atomic effect.
> The effect condition $effcond(e, e')$ under which $e$ triggers
> given the effect $e'$ is a propositional formula defined as follows:
> - $effcond(e, e) = \top$
> - $effcond(e, e') = \bot$ for atomic effects $e' \neq e$
> - $effcond(e, (e_1 \wedge \cdots \wedge e_n)) = effcond(e, e_1) \vee \cdots \vee effcond(e, e_n)$
> - $effcond(e, (\chi \rhd e')) = \chi \wedge effcond(e, e')$

Intuition: $effcond(e, e')$ represents the condition that must be true
in the current state for the effect $e'$ to lead to the atomic effect $e$

## Semantics of Operators: Propositional Case

> **Definition (Applicable, Resulting State)**
> Let $V$ be a set of propositional state variables.
> Let $s$ be a state over $V$, and let $o$ be an operator over $V$.
> Operator $o$ is applicable in $s$ if $s \models pre(o)$.
>
> If $o$ is applicable in $s$, the resulting state of applying $o$ in $s$,
> written $s[\![o]\!]$, is the state $s'$ defined as follows for all $v \in V$:
>
> $$s'(v) = \begin{cases} \mathsf{T} & \text{if } s \models effcond(v, e) \\ \mathsf{F} & \text{if } s \models effcond(\neg v, e) \wedge \neg effcond(v, e) \\ s(v) & \text{if } s \not\models effcond(v, e) \vee effcond(\neg v, e) \end{cases}$$
>
> where $e = eff(o)$.

## Add-after-Delete Semantics

Note:
- The definition implies that if a variable is simultaneously "added" (set to T) and "deleted" (set to F), the value T takes precedence.
- This is called add-after-delete semantics.
- This detail of semantics is somewhat arbitrary, but has proven useful in applications.
- For finite-domain variables, there are no distinguished values like "true" and "false", and a different semantics is used.

## Conflicting Effects and Consistency Condition

- What should an effect of the form $v := a \wedge v := b$ mean?
- For finite-domain representations, the accepted semantics is to make this illegal, i.e., to make an operator inapplicable if it would lead to conflicting effects.

> **Definition (Consistency Condition)**
> Let $e$ be an effect over finite-domain state variables $V$.
> The consistency condition for $e$, $consist(e)$ is defined as
>
> $$\bigwedge_{v \in V} \bigwedge_{d, d' \in dom(v), d \neq d'} \neg(effcond(v := d, e) \wedge effcond(v := d', e)).$$

## Semantics of Operators: Finite-Domain Case

### Definition (Applicable, Resulting State)

Let $V$ be a set of finite-domain state variables.
Let $s$ be a state over $V$, and let $o$ be an operator over $V$.
Operator $o$ is applicable in $s$ if $s \models pre(o) \land consist(eff(o))$.

If $o$ is applicable in $s$, the resulting state of applying $o$ in $s$,
written $s[\![o]\!]$, is the state $s'$ defined as follows for all $v \in V$:

$$s'(v) = \begin{cases} d & \text{if } s \models effcond(v := d, eff(o)) \text{ for some } d \in dom(v) \\ s(v) & \text{otherwise} \end{cases}$$

---

## Applying Operators: Example

### Example

Consider the operator $o = \langle a, \neg a \land (\neg c \rhd \neg b) \rangle$
and the state $s = \{a \mapsto \mathsf{T}, b \mapsto \mathsf{T}, c \mapsto \mathsf{T}, d \mapsto \mathsf{T}\}$.

The operator $o$ is applicable in $s$ because $s \models a$.

Effect conditions of $eff(o)$:

$$\begin{aligned} effcond(a, eff(o)) &= effcond(a, \neg a \land (\neg c \rhd \neg b)) \\ &= effcond(a, \neg a) \lor effcond(a, \neg c \rhd \neg b) \\ &= \bot \lor (\neg c \land effcond(a, \neg b)) \\ &= \bot \lor (\neg c \land \bot) \\ &\equiv \bot \quad \rightsquigarrow \text{ false in state } s \end{aligned}$$

---

## Applying Operators: Example

### Example

Consider the operator $o = \langle a, \neg a \land (\neg c \rhd \neg b) \rangle$
and the state $s = \{a \mapsto \mathsf{T}, b \mapsto \mathsf{T}, c \mapsto \mathsf{T}, d \mapsto \mathsf{T}\}$.

The operator $o$ is applicable in $s$ because $s \models a$.

Effect conditions of $eff(o)$:

$$\begin{aligned} effcond(\neg a, eff(o)) &= effcond(\neg a, \neg a \land (\neg c \rhd \neg b)) \\ &= effcond(\neg a, \neg a) \lor effcond(\neg a, \neg c \rhd \neg b) \\ &= \top \lor effcond(\neg a, \neg c \rhd \neg b) \\ &\equiv \top \quad \rightsquigarrow \text{ true in state } s \end{aligned}$$

---

## Applying Operators: Example

### Example

Consider the operator $o = \langle a, \neg a \land (\neg c \rhd \neg b) \rangle$
and the state $s = \{a \mapsto \mathsf{T}, b \mapsto \mathsf{T}, c \mapsto \mathsf{T}, d \mapsto \mathsf{T}\}$.

The operator $o$ is applicable in $s$ because $s \models a$.

Effect conditions of $eff(o)$:

$$\begin{aligned} effcond(b, eff(o)) &= effcond(b, \neg a \land (\neg c \rhd \neg b)) \\ &= effcond(b, \neg a) \lor effcond(b, \neg c \rhd \neg b) \\ &= \bot \lor (\neg c \land effcond(b, \neg b)) \\ &= \bot \lor (\neg c \land \bot) \\ &\equiv \bot \quad \rightsquigarrow \text{ false in state } s \end{aligned}$$

## Applying Operators: Example

> **Example**
>
> Consider the operator $o = \langle a, \neg a \wedge (\neg c \rhd \neg b) \rangle$
> and the state $s = \{a \mapsto \mathsf{T}, b \mapsto \mathsf{T}, c \mapsto \mathsf{T}, d \mapsto \mathsf{T}\}$.
>
> The operator $o$ is applicable in $s$ because $s \models a$.
>
> Effect conditions of $\mathit{eff}(o)$:
>
> $$\begin{aligned}
> \mathit{effcond}(\neg b, \mathit{eff}(o)) &= \mathit{effcond}(\neg b, \neg a \wedge (\neg c \rhd \neg b)) \\
> &= \mathit{effcond}(\neg b, \neg a) \vee \mathit{effcond}(\neg b, \neg c \rhd \neg b) \\
> &= \bot \vee (\neg c \wedge \mathit{effcond}(\neg b, \neg b)) \\
> &= \bot \vee (\neg c \wedge \top) \\
> &\equiv \neg c \quad \rightsquigarrow \quad \text{false in state } s
> \end{aligned}$$

## Applying Operators: Example

> **Example**
>
> Consider the operator $o = \langle a, \neg a \wedge (\neg c \rhd \neg b) \rangle$
> and the state $s = \{a \mapsto \mathsf{T}, b \mapsto \mathsf{T}, c \mapsto \mathsf{T}, d \mapsto \mathsf{T}\}$.
>
> The operator $o$ is applicable in $s$ because $s \models a$.
>
> Effect conditions of $\mathit{eff}(o)$:
>
> $$\begin{aligned}
> \mathit{effcond}(c, \mathit{eff}(o)) &\equiv \bot \quad &\rightsquigarrow \quad \text{false in state } s \\
> \mathit{effcond}(\neg c, \mathit{eff}(o)) &\equiv \bot \quad &\rightsquigarrow \quad \text{false in state } s \\
> \mathit{effcond}(d, \mathit{eff}(o)) &\equiv \bot \quad &\rightsquigarrow \quad \text{false in state } s \\
> \mathit{effcond}(\neg d, \mathit{eff}(o)) &\equiv \bot \quad &\rightsquigarrow \quad \text{false in state } s
> \end{aligned}$$
>
> The resulting state of applying $o$ in $s$ is the state
> $\{a \mapsto \mathsf{F}, b \mapsto \mathsf{T}, c \mapsto \mathsf{T}, d \mapsto \mathsf{T}\}$.

## Example Operators: Blocks World

> **Example (Blocks World Operators)**
>
> To model blocks world operators conveniently,
> we use auxiliary state variables *A-clear*, *B-clear*, and *C-clear*
> to express that there is nothing on top of a given block.
>
> Then blocks world operators can be modeled as:
> - $\langle$ *A-clear* $\wedge$ *A-on-table* $\wedge$ *B-clear*,  *A-on-B* $\wedge$ ¬*A-on-table* $\wedge$ ¬*B-clear* $\rangle$
> - $\langle$ *A-clear* $\wedge$ *A-on-table* $\wedge$ *C-clear*,  *A-on-C* $\wedge$ ¬*A-on-table* $\wedge$ ¬*C-clear* $\rangle$
> - $\langle$ *A-clear* $\wedge$ *A-on-B*,  *A-on-table* $\wedge$ ¬*A-on-B* $\wedge$ *B-clear* $\rangle$
> - $\langle$ *A-clear* $\wedge$ *A-on-C*,  *A-on-table* $\wedge$ ¬*A-on-C* $\wedge$ *C-clear* $\rangle$
> - $\langle$ *A-clear* $\wedge$ *A-on-B* $\wedge$ *C-clear*,  *A-on-C* $\wedge$ ¬*A-on-B* $\wedge$ *B-clear* $\wedge$ ¬*C-clear* $\rangle$
> - $\langle$ *A-clear* $\wedge$ *A-on-C* $\wedge$ *B-clear*,  *A-on-B* $\wedge$ ¬*A-on-C* $\wedge$ *C-clear* $\wedge$ ¬*B-clear* $\rangle$
> - . . .

## Example Operator: 4-Bit Counter

> **Example (Incrementing a 4-Bit Counter)**
>
> Operator to increment a 4-bit number $b_3 b_2 b_1 b_0$ represented
> by 4 state variables $b_0, \ldots, b_3$:
>
> precondition:
> $$\neg b_0 \vee \neg b_1 \vee \neg b_2 \vee \neg b_3$$
>
> effect:
> $$\begin{aligned}
> &(\neg b_0 \rhd b_0) \wedge \\
> &((\neg b_1 \wedge b_0) \rhd (b_1 \wedge \neg b_0)) \wedge \\
> &((\neg b_2 \wedge b_1 \wedge b_0) \rhd (b_2 \wedge \neg b_1 \wedge \neg b_0)) \wedge \\
> &((\neg b_3 \wedge b_2 \wedge b_1 \wedge b_0) \rhd (b_3 \wedge \neg b_2 \wedge \neg b_1 \wedge \neg b_0))
> \end{aligned}$$

# A4.3 Planning Tasks

## Planning Tasks

#### Definition (Planning Task)

A planning task is a 4-tuple $\Pi = \langle V, I, O, \gamma \rangle$ where

- $V$ is a finite set of state variables,
- $I$ is a valuation over $V$ called the initial state,
- $O$ is a finite set of operators over $V$, and
- $\gamma$ is a formula over $V$ called the goal.

$V$ must either consist only of propositional
or only of finite-domain state variables.

In the first case, $\Pi$ is called a propositional planning task,
otherwise an FDR planning task (finite-domain representation).

Note: Whenever we just say planning task (without
"propositional" or "FDR"), both kinds of tasks are allowed.

## Mapping Planning Tasks to Transition Systems

#### Definition (Transition System Induced by a Planning Task)

The planning task $\Pi = \langle V, I, O, \gamma \rangle$ induces
the transition system $\mathcal{T}(\Pi) = \langle S, L, c, T, s_0, S_\star \rangle$, where

- $S$ is the set of all states over $V$,
- $L$ is the set of operators $O$,
- $c(o) = cost(o)$ for all operators $o \in O$,
- $T = \{\langle s, o, s' \rangle \mid s \in S, \ o \text{ applicable in } s, \ s' = s[\![o]\!]\}$,
- $s_0 = I$, and
- $S_\star = \{s \in S \mid s \models \gamma\}$.

## Planning Tasks: Terminology

- Terminology for transitions systems is also applied
  to the planning tasks $\Pi$ 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 solution of $\mathcal{T}(\Pi)$
  is called a plan of $\Pi$.

## Satisficing and Optimal 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 cost among all plans for Π,
          or **unsolvable** if no plan for Π exists

---

# A4.4 Summary

## Summary

- ► Planning tasks compactly represent transition systems
  and are suitable as inputs for planning algorithms.
- ► They are based on concepts from propositional logic,
  enhanced to model state change.
- ► Planning tasks can be propositional or finite-domain.
- ► States of planning tasks are assignments to its state variables.
- ► Operators of propositional planning tasks describe
  in which situations (precondition), how (effect) and
  at which cost the state of the world can be changed.
- ► In satisficing planning, we must find a solution
  for a planning task (or show that no solution exists).
- ► In optimal planning, we must additionally guarantee
  that generated solutions are of minimal cost.