Planning and Optimization

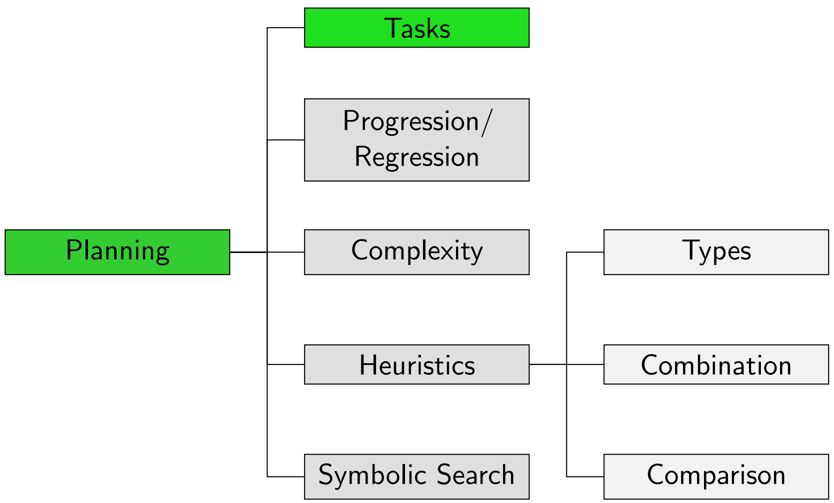
A8. Finite Domain Representation

Malte Helmert and Gabriele Röger

Universität Basel

October 9, 2017

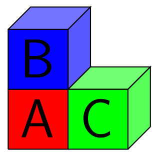
Content of this Course



Reminder: Blocks World with Boolean State Variables

Example

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



$\rightsquigarrow 2^9 = 512$ states

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

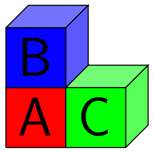
Blocks World with Finite-Domain State Variables

Use three finite-domain state variables:

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

Example

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



$\rightsquigarrow 3^3 = 27$ states

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

A8.1 FDR Planning Tasks

Finite-Domain State Variables

Definition (Finite-Domain State Variable)

A finite-domain state variable is a symbol v
with an associated finite domain, i.e., a non-empty finite set.

We write $\mathrm{dom}(v)$ for the domain of v.

Example (Blocks World)

$v = \textit{above-a}$, $\mathrm{dom}(\textit{above-a}) = \{\mathrm{b}, \mathrm{c}, \mathrm{nothing}\}$

This state variable encodes the same information as the propositional variables *B-on-A*, *C-on-A* and *A-clear*.

Finite-Domain States

Definition (Finite-Domain State)

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

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

Example (Blocks World)

$s = \{\textit{above-a} \mapsto \mathrm{nothing}, \textit{above-b} \mapsto \mathrm{a}, \textit{above-c} \mapsto \mathrm{b},$
$\textit{below-a} \mapsto \mathrm{b}, \textit{below-b} \mapsto \mathrm{c}, \textit{below-c} \mapsto \mathrm{table}\}$

Finite-Domain Formulas

Definition (Finite-Domain Formula)

Logical formulas over finite-domain state variables V are defined identically to the propositional case, except that instead of atomic formulas of the form $v' \in V'$ with propositional state variables V', there are atomic formulas of the form $v = d$, where $v \in V$ and $d \in \mathrm{dom}(v)$.

Example (Blocks World)

The formula $(above\text{-}a = \mathrm{nothing}) \lor \lnot(below\text{-}b = \mathrm{c})$ corresponds to the formula $A\text{-}clear \lor \lnot B\text{-}on\text{-}C$.

Finite-Domain Effects

Definition (Finite-Domain Effect)

Effects over finite-domain state variables V are defined identically to the propositional case, except that instead of atomic effects of the form v' and $\lnot v'$ with propositional state variables $v' \in V'$, there are atomic effects of the form $v := d$, where $v \in V$ and $d \in \mathrm{dom}(v)$.

Example (Blocks World)

The effect

$(below\text{-}a := \mathrm{table}) \land ((above\text{-}b = \mathrm{a}) \rhd (above\text{-}b := \mathrm{nothing}))$

corresponds to the effect

$A\text{-}on\text{-}table \land \lnot A\text{-}on\text{-}B \land \lnot A\text{-}on\text{-}C \land (A\text{-}on\text{-}B \rhd (B\text{-}clear \land \lnot A\text{-}on\text{-}B))$.

$\leadsto$ finite-domain operators, effect conditions etc. follow

Planning Tasks in Finite-Domain Representation

Definition (Planning Task in Finite-Domain Representation)

A planning task in finite-domain representation or FDR planning task is a 4-tuple $\Pi = \langle V, I, O, \gamma \rangle$ where

- V is a finite set of finite-domain state variables,
- I is a state over V called the initial state,
- O is a finite set of finite-domain operators over V, and
- γ is a formula over V called the goal.

A8.2 FDR Task Semantics

FDR Task Semantics: Introduction

- We have now defined what FDR tasks look like.
- We still have to define their semantics.
- Because they are similar to propositional planning tasks, we can define their semantics in a very similar way.

Direct vs. Compilation Semantics

We describe two ways of defining semantics for FDR tasks:

- directly, mirroring our definitions for propositional tasks
- by compilation to propositional tasks

Comparison of the semantics:

- The two semantics are equivalent in terms of the reachable state space and hence in terms of the set of solutions. (We will not prove this.)
- They are not equivalent w.r.t. the set of all states.

Where the distinction matters, we use the direct semantics in this course unless stated otherwise.

Conflicting Effects

- As with propositional planning tasks, there is a subtlety: what should an effect of the form $v := \mathsf{a} \land v := \mathsf{b}$ mean?
- For FDR tasks, the common convention is to make this illegal, i.e., to make an operator inapplicable if it would lead to conflicting effects.

Consistency Condition and Applicability

Definition (Consistency Condition)

Let e be an effect over finite-domain state variables V.

The consistency condition for e, $consist(e)$ is defined as

$$\bigwedge_{v \in V} \bigwedge_{d, d' \in \mathrm{dom}(v), d \neq d'} \neg(effcond(v := d, e) \land effcond(v := d', e)).$$

Definition (Applicable FDR Operator)

An FDR operator o is applicable in a state s if $s \models pre(o) \land consist(eff(o))$.

The definitions of $s[\![o]\!]$ etc. then follow in the natural way.

Reminder: Semantics of Propositional Planning Tasks

Reminder from Chapter A4:

Definition (Transition System Induced by a Prop. Planning Task)

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

Semantics of Planning Tasks

A definition that works for both types of planning tasks:

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

Planning task here refers to either a propositional or FDR task.

Compilation Semantics

Definition (Induced Propositional Planning Task)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be an FDR planning task. The induced propositional planning task Π' is the (regular) planning task $\Pi' = \langle V', I', O', \gamma' \rangle$, where

- $V' = \{\langle v, d \rangle \mid v \in V, d \in \mathrm{dom}(v)\}$
- $I'(\langle v, d \rangle) = \mathbf{T}$ iff $I(v) = d$
- O' and γ' are obtained from O and γ by
  - replacing each operator precondition $pre(o)$ by $pre(o) \land consist(eff(o))$, and then
  - replacing each atomic formula $v = d$ by the proposition $\langle v, d \rangle$,
  - replacing each atomic effect $v := d$ by the effect $\langle v, d \rangle \land \bigwedge_{d' \in \mathrm{dom}(v) \setminus \{d\}} \neg \langle v, d' \rangle$.

A8.3 SAS$^+$ Planning Tasks

SAS$^+$ Planning Tasks

Definition (SAS$^+$ Planning Task)

An FDR planning task $\Pi = \langle V, I, O, \gamma \rangle$ is called an SAS$^+$ planning task if

- there are no conditional effects in O, and
- all operator preconditions in O and the goal formula γ are conjunctions of atoms.

SAS$^+$ vs. STRIPS

- SAS$^+$ is the analogue of STRIPS planning tasks for FDR
- induced propositional planning task of a SAS$^+$ task is a STRIPS planning task after simplification (consistency conditions simplify to $\bot$ or $\top$)
- FDR tasks obtained by mutex-based reformulation of STRIPS planning tasks are SAS$^+$ tasks

A8.4 Transition Normal Form

Variables Occurring in Conditions and Effects

- Many algorithmic problems for SAS$^+$ planning tasks become simpler when we can make two further restrictions.
- These are related to the variables that occur in conditions and effects of the task.

Definition ($vars(\varphi)$, $vars(e)$)

For a logical formula φ over finite-domain variables V, $vars(\varphi)$ denotes the set of finite-domain variables occurring in φ.

For an effect e over finite-domain variables V, $vars(e)$ denotes the set of finite-domain variables occurring in e.

Transition Normal Form

Definition (Transition Normal Form)

A SAS$^+$ planning task $\Pi = \langle V, I, O, \gamma \rangle$ is in transition normal form (TNF) if

- for all $o \in O$, $vars(pre(o)) = vars(eff(o))$, and
- $vars(\gamma) = V$.

In words, an operator in TNF must mention the same variables in the precondition and effect, and a goal in TNF must mention all variables (= specify exactly one goal state).

Converting Operators to TNF: Violations

There are two ways in which an operator o can violate TNF:

- There exists a variable $v \in vars(pre(o)) \setminus vars(eff(o))$.
- There exists a variable $v \in vars(eff(o)) \setminus vars(pre(o))$.

The first case is easy to address: if $v = d$ is a precondition with no effect on v, just add the effect $v := d$.

The second case is more difficult: if we have the effect $v := d$ but no precondition on v, how can we add a precondition on v without changing the meaning of the operator?

Converting Operators to TNF: Multiplying Out

Solution 1: multiplying out

1. While there exists an operator o and a variable $v \in vars(eff(o))$ with $v \notin vars(pre(o))$:
   - For each $d \in \mathrm{dom}(v)$, add a new operator that is like o but with the additional precondition $v = d$.
   - Remove the original operator.
2. Repeat the previous step until no more such variables exist.

Problem:

- If an operator o has n such variables, each with k values in its domain, this introduces k^n variants of o.
- Hence, this is an exponential transformation.

Converting Operators to TNF: Auxiliary Values

Solution 2: auxiliary values

1. For every variable v, add a new auxiliary value u to its domain.
2. For every variable v and value $d \in \mathrm{dom}(v) \setminus \{\mathrm{u}\}$, add a new operator to change the value of v from d to u at no cost: $\langle v = d, v := \mathrm{u}, 0 \rangle$.
3. For all operators o and all variables $v \in vars(eff(o)) \setminus vars(pre(o))$, add the precondition $v = \mathrm{u}$ to $pre(o)$.

Properties:

- Transformation can be computed in linear time.
- Due to the auxiliary values, there are new states and transitions in the induced transition system, but all path costs between original states remain the same.

Converting Goals to TNF

- The auxiliary value idea can also be used to convert the goal γ to TNF.
- For every variable $v \notin \mathit{vars}(\gamma)$, add the condition $v = \mathrm{u}$ to γ.

With these ideas, every SAS$^+$ planning task can be converted into transition normal form in linear time.

A8.5 Summary

Summary

- Planning tasks in finite-domain representation (FDR) are an alternative to propositional planning tasks.
- FDR tasks are often more compact (have fewer states).
- This makes many planning algorithms more efficient when working with a finite-domain representation.
- SAS$^+$ tasks are a restricted form of FDR tasks where only conjunctions of atoms are allowed in the preconditions, effects and goal. No conditional effects are allowed.
- Transition normal form (TNF) is even more restricted: for each operator, preconditions and effects must mention the same variables, and there must be a unique goal state.
- SAS$^+$ tasks can be converted to TNF in linear time.