

Planning and Optimization

A8. Finite Domain Representation

Malte Helmert and Gabriele Röger

Universität Basel

October 10, 2016

Propositional vs. Finite-Domain Planning Tasks

- In this chapter, we introduce **planning tasks in finite domain representation** a.k.a. **FDR planning tasks**.
- To distinguish them more clearly from the planning tasks using propositional state variables introduced in Chapter A4, we will refer to the latter as **propositional planning tasks** rather than just **planning tasks** in this chapter.

Reminder: Blocks World with Boolean State Variables

Example

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

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

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

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

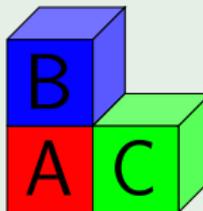
$$s(B\text{-on-}C) = \mathbf{F}$$

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

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

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

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



↪ $2^9 = 512$ 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*: {b, c, table}
- *below-b*: {a, c, table}
- *below-c*: {a, b, table}

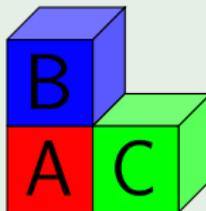
Example

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

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

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

$\rightsquigarrow 3^3 = 27 \text{ states}$



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

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 $\text{dom}(v)$ for the domain of v .

Example (Blocks World)

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

This state variable encodes the same information as the propositional variables $B\text{-on-}A$, $C\text{-on-}A$ and $A\text{-clear}$.

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 $s : V \rightarrow \bigcup_{v \in V} \text{dom}(v)$ such that $s(v) \in \text{dom}(v)$ for all $v \in V$.

Example (Blocks World)

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

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 $d \in \text{dom}(v)$.

Example (Blocks World)

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

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 $\neg v'$ with propositional state variables $v' \in V'$, there are atomic effects of the form $v := d$, where $v \in V$ and $d \in \text{dom}(v)$.

Example (Blocks World)

The effect

$(below-a := table) \wedge ((above-b = a) \triangleright (above-b := nothing))$

corresponds to the effect

$A-on-T \wedge \neg A-on-B \wedge \neg A-on-C \wedge (A-on-B \triangleright (B-clear \wedge \neg A-on-B)).$

\rightsquigarrow definition of **finite-domain operators** follows from this

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**.

FDR Task Semantics

FDR Task Semantics: Informally

- We have now defined what FDR tasks look like.
- We still have to define their **semantics**.
- Because they are similar to propositional planning tasks, it makes sense to define FDR semantics in terms of propositional planning task semantics.
- As with propositional planning tasks, there is a subtlety: what should an effect of the form $v := a \wedge v := b$ 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

Definition (Consistency Condition)

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

For all $v \in V$ and $d \in \text{dom}(v)$, let $\chi_{v:=d}$ be a logical formula with

$$s \models \chi_{v:=d} \quad \text{iff} \quad (v := d) \in [e]_s$$

for all states s .

The **consistency condition** for e , χ_e^{cons} is defined as

$$\bigwedge_{v \in V} \bigwedge_{d, d' \in \text{dom}(v), d \neq d'} \neg(\chi_{v:=d} \wedge \chi_{v:=d'}).$$

FDR Planning Task 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

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

↪ define operator semantics, transition systems, plans, ...
for FDR task Π in terms of its induced propositional task

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 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 \perp or \top)
- FDR tasks obtained by mutex-based reformulation of STRIPS plannings task are SAS⁺ tasks

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 ($\text{vars}(\varphi)$, $\text{vars}(e)$)

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

For an effect e over finite-domain variables V , $\text{vars}(e)$ 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 $o \in O$, $\text{vars}(\text{pre}(o)) = \text{vars}(\text{eff}(o))$, and
- $\text{vars}(\gamma) = V$.

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

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 $v \in \text{vars}(\text{eff}(o))$ with $v \notin \text{vars}(\text{pre}(o))$:
 - For each $d \in \text{dom}(v)$, 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 $d \in \text{dom}(v) \setminus \{u\}$, add a new operator to change the value of v from d to u at no cost: $\langle v = d, v := u, 0 \rangle$.
- 3 For all operators o and all variables $v \in \text{vars}(\text{eff}(o)) \setminus \text{vars}(\text{pre}(o))$, add the precondition $v = u$ to $\text{pre}(o)$.

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 $v \notin \text{vars}(\gamma)$, add the condition $v = u$ to γ .

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

Summary

Summary

- Planning tasks in **finite-domain representation (FDR)** are an alternative to propositional planning tasks.
- FDR is 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**.