

Planning and Optimization

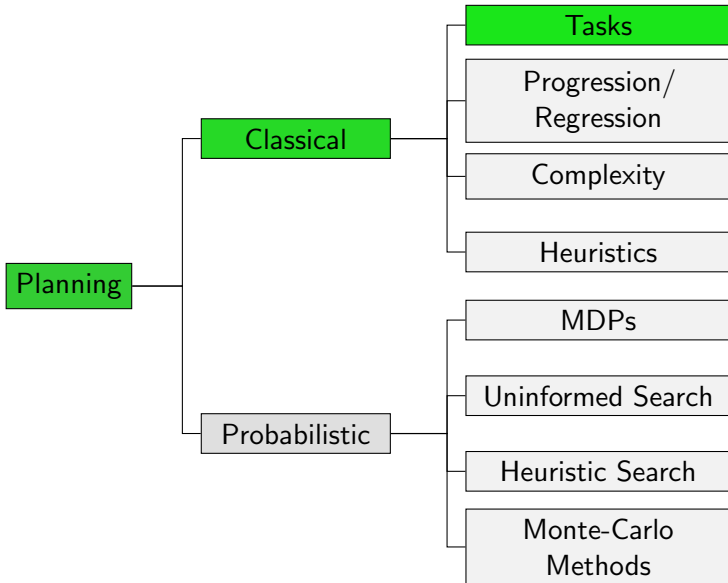
A7. Invariants, Mutexes and Finite Domain Representation

Gabriele Röger and Thomas Keller

Universität Basel

October 8, 2018

Content of this Course



Invariants

Invariants

- When we as humans reason about planning tasks, we implicitly make use of “obvious” properties of these tasks.
 - **Example:** we are never in two places at the same time
- We can represent such properties as logical formulas φ that are **true in all reachable states**.
 - **Example:** $\varphi = \neg(at\text{-}uni \wedge at\text{-}home)$
- Such formulas are called **invariants** of the task.

Invariants: Definition

Definition (Invariant)

An **invariant** of a planning task Π with state variables V is a logical formula φ over V such that $s \models \varphi$ for all reachable states s of Π .

Computing Invariants

- Theoretically, testing if an arbitrary formula φ is an invariant is **as hard as planning** itself.
 - ↪ **proof idea**: a planning task is **unsolvable** iff the negation of its goal is an invariant
- Still, many practical invariant synthesis algorithms exist.
- To remain efficient (= polynomial-time), these algorithms only compute a **subset** of all useful invariants.
 - ↪ **sound**, but not **complete**
- Empirically, they tend to at least find the “obvious” invariants of a planning task.

Exploiting Invariants

Invariants have many uses in planning:

- **Regression search:**
Prune states that violate (are inconsistent with) invariants.
- **Planning as satisfiability:**
Add invariants to a SAT encoding of a planning task to get tighter constraints.
- **Reformulation:**
Derive a more compact state space representation (i.e., with fewer unreachable states).

We now briefly discuss the last point because it is important for **planning tasks in finite-domain representation**, introduced in the following chapter.

Mutexes

Mutexes

Invariants that take the form of **binary clauses** are called **mutexes** because they express that certain variable assignments cannot be simultaneously true and are hence **mutually exclusive**.

Example (Blocks World)

The invariant $\neg A\text{-on-}B \vee \neg A\text{-on-}C$ states that $A\text{-on-}B$ and $A\text{-on-}C$ are mutex.

We say that a larger **set of literals** is mutually exclusive if every subset of two literals is a mutex.

Example (Blocks World)

Every pair in $\{B\text{-on-}A, C\text{-on-}A, D\text{-on-}A, A\text{-clear}\}$ is mutex.

Encoding Mutex Groups as Finite-Domain Variables

Let $L = \{\ell_1, \dots, \ell_n\}$ be mutually exclusive literals over n different variables $V_L = \{v_1, \dots, v_n\}$.

Then the planning task can be rephrased using a single **finite-domain** (i.e., non-binary) state variable v_L with $n + 1$ possible values in place of the n variables in V_L :

- n of the possible values represent situations in which **exactly one** of the literals in L is true.
- The remaining value represents situations in which **none of the literals** in L is true.
 - **Note:** If we can prove that one of the literals in L must be true in each state (i.e., $\ell_1 \vee \dots \vee \ell_n$ is an invariant), this additional value can be omitted.

In many cases, the reduction in the number of variables dramatically improves performance of a planning algorithm.

FDR Planning Tasks

Reminder: Blocks World with Boolean State Variables

Example

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

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

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

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

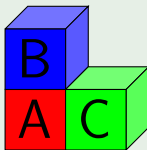
$$s(B\text{-on-}C) = \mathbf{F}$$

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

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

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

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



↪ $2^9 = 512$ 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}

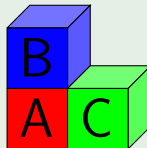
Example

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

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

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

$\rightsquigarrow 3^3 = 27$ states



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

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 $(\textit{above-a} = \textit{nothing}) \vee \neg(\textit{below-b} = \textit{c})$ 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

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

corresponds to the effect

$A\text{-on-table} \wedge \neg A\text{-on-B} \wedge \neg A\text{-on-C} \wedge (A\text{-on-B} \triangleright (B\text{-clear} \wedge \neg A\text{-on-B}))$.

\rightsquigarrow 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**.

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 $v := a \wedge v := b$ 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 and Applicability

Definition (Consistency Condition)

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

The **consistency condition** for e , $\mathit{consist}(e)$ is defined as

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

Definition (Applicable FDR Operator)

An FDR operator o is **applicable** in a state s if $s \models \mathit{pre}(o) \wedge \mathit{consist}(\mathit{eff}(o))$.

The definitions of $s \llbracket o \rrbracket$ 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 ,
- $c(o) = \text{cost}(o)$ 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
- $S_\star = \{ s \in S \mid s \models \gamma \}$.

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 ,
- $c(o) = \text{cost}(o)$ 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 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

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

SAS⁺ Planning Tasks

SAS⁺ Planning Tasks

Definition (SAS⁺ Planning Task)

An FDR planning task $\Pi = \langle V, I, O, \gamma \rangle$ is called a **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 \perp or \top)
- FDR tasks obtained by mutex-based reformulation of STRIPS planning tasks are SAS⁺ tasks

Summary

Summary

- **Invariants** are common properties of all reachable states, expressed as logical formulas.
- **Mutexes** are invariants that express that certain pairs of literals are mutually exclusive.
- 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.

Quiz



kahoot.it