

Principles of AI Planning

9. Interlude: Finite-domain representation

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Bernhard Nebel and Robert Mattmüller

November 26th, 2014



Invariants

Invariants

- Introduction
- Computing invariants
- Exploiting invariants

FDR
planning
tasks

Summary



- 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 express this as a logical formula φ that is **true in all reachable states**.
 - **Example:** $\varphi = \neg(at\text{-}uni \wedge at\text{-}home)$
- Such formulae are called **invariants** of the task.

Invariants

Introduction

Computing
invariants

Exploiting
invariants

FDR
planning
tasks

Summary



How does an **automated** planner come up with invariants?

- Theoretically, testing if an arbitrary formula φ is an invariant is **as hard as planning** itself.
- Still, many practical invariant synthesis algorithms exist.
- To remain efficient (= polynomial-time), these algorithms only compute a **subset** of all useful invariants.
- Empirically, they tend to at least find the “obvious” invariants of a planning task.

Invariants

Introduction

Computing
invariants

Exploiting
invariants

FDR

planning
tasks

Summary



Most algorithms for generating invariants are based on a **generate-test-repair** paradigm:

- **Generate:** Suggest some invariant candidates, e. g., by enumerating all possible formulas φ of a certain size.
- **Test:** Try to prove that φ is indeed an invariant. Usually done **inductively**:
 - 1 Test that **initial state** satisfies φ .
 - 2 Test that if φ is true in the current state, it remains true after applying a single operator.
- **Repair:** If invariant test fails, replace candidate φ by a **weaker** formula, ideally exploiting **why** the proof failed.

Invariants

Introduction

Computing invariants

Exploiting invariants

FDR

planning tasks

Summary



We discussed invariant synthesis in detail in previous courses on AI planning, but this year we will focus on other aspects of planning.

Literature on invariant synthesis:

- DISCOPLAN (Gerevini & Schubert, 1998)
- TIM (Fox & Long, 1998)
- Edelkamp & Helmert's algorithm (1999)
- Rintanen's algorithm (2000)
- Bonet & Geffner's algorithm (2001)
- Helmert's algorithm (2009)

Invariants

Introduction

Computing
invariants

Exploiting
invariants

FDR

planning
tasks

Summary



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 lower percentage of unreachable states).

We now briefly discuss the last point, since it leads to **planning tasks in finite-domain representation**, which are very important for the next chapters.

Invariants

Introduction

Computing invariants

Exploiting invariants

FDR

planning tasks

Summary



Planning tasks in finite-domain representation

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary



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

Example (Blocksworld)

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.

Often, a larger **set of literals** is mutually exclusive because every pair of them forms a mutex.

Example (Blocksworld)

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

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary

Encoding mutex groups as finite-domain variables



Let $L = \{l_1, \dots, l_n\}$ be mutually exclusive literals over n different variables $A_L = \{a_1, \dots, a_n\}$.

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 A_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 has to be true in each state, this additional value can be omitted.

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

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary



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 \mathcal{D}_v for the domain of v .

Example

$v = \textit{above-a}$, $\mathcal{D}_{\textit{above-a}} = \{\textit{b}, \textit{c}, \textit{d}, \textit{nothing}\}$

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

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary



Definition (finite-domain state)

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

A **state** over V is an assignment $s : V \rightarrow \bigcup_{v \in V} \mathcal{D}_v$ such that $s(v) \in \mathcal{D}_v$ for all $v \in V$.

Example

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

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary



Definition (finite-domain formulae)

Logical formulae over finite-domain state variables V are defined as in the propositional case, except that instead of atomic formulae of the form $a \in A$, there are atomic formulae of the form $v = d$, where $v \in V$ and $d \in \mathcal{D}_v$.

Example

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

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary

Definition (finite-domain effects)

Effects over finite-domain state variables V are defined as in the propositional case, except that instead of atomic effects of the form a and $\neg a$ with $a \in A$, there are atomic effects of the form $v := d$, where $v \in V$ and $d \in \mathcal{D}_V$.

Example

The effect

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

corresponds to the effect

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

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

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS⁺ planning
tasks

Summary



Definition (planning task in finite-domain representation)

A **deterministic 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 an **initial state** over V ,
- O is a finite set of **finite-domain operators** over V , and
- γ is a formula over V describing the **goal states**.

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary



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 $\Pi' = \langle A', I', O', \gamma' \rangle$, where

- $A' = \{(v, d) \mid v \in V, d \in \mathcal{D}_v\}$
- $I'((v, d)) = 1$ iff $I(v) = d$
- O' and γ' are obtained from O and γ by replacing
 - each atomic formula $v = d$ with the proposition (v, d) ,
 - each atomic effect $v := d$ with the effect $(v, d) \wedge \bigwedge_{d' \in \mathcal{D}_v \setminus \{d\}} \neg(v, d')$.

- \rightsquigarrow can define operator semantics, plans, relaxed planning graphs, ... for Π in terms of its induced propositional planning task

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS* planning
tasks

Summary



Definition (SAS⁺ planning task)

An FDR planning task $\Pi = \langle V, I, O, \gamma \rangle$ is called an **SAS⁺ planning task** iff there are no conditional effects in O and all operator preconditions in O and the goal formula γ are conjunctions of atoms.

- analogue of STRIPS planning tasks for finite-domain representations
- induced propositional planning task of a SAS⁺ planning task is STRIPS
- FDR tasks obtained by invariant-based reformulation of STRIPS planning task are SAS⁺

Invariants

FDR
planning
tasks

Mutexes

FDR planning
tasks

Relationship to
propositional
planning tasks

SAS⁺ planning
tasks

Summary



- **Invariants** are common properties of all reachable states, expressed as logical formulas.
- A number of algorithms for **computing invariants** exist.
- These algorithms will not find **all useful invariants** (which is too hard), but try to find some useful subset within reasonable (polynomial) time.
- **Mutexes** are invariants that express that certain pairs of state variable assignments are mutually exclusive.
- Groups of mutexes can be used for **problem reformulation**, transforming a planning task into **finite-domain representation (FDR)**.
- Many planning algorithms are more efficient when working on these FDR tasks (rather than the original tasks) because they contain **fewer unreachable states**.

Invariants

FDR
planning
tasks

Summary