

# Planning and Optimization

## E2. Invariants and Mutexes

Malte Helmert and Gabriele Röger

Universität Basel

November 3, 2025

# Planning and Optimization

November 3, 2025 — E2. Invariants and Mutexes

## E2.1 Invariants

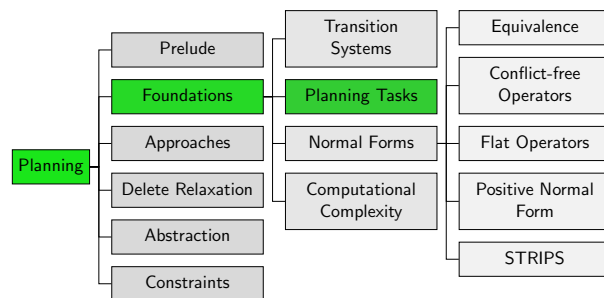
## E2.2 Computing Invariants

## E2.3 Mutexes

## E2.4 Reformulation

## E2.5 Summary

## Content of the Course



## E2.1 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  $\varphi$  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  $\Pi$  with state variables  $V$  is a **logical formula**  $\varphi$  over  $V$  such that  $s \models \varphi$  for **all reachable states**  $s$  of  $\Pi$ .

## E2.2 Computing Invariants

## Computing Invariants

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

- ▶ Theoretically, testing if a formula  $\varphi$  is an invariant is **as hard as planning** itself.
  - $\rightsquigarrow$  **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.
  - $\rightsquigarrow$  **sound**, but not **complete**
- ▶ Empirically, they tend to at least find the “obvious” invariants of a planning task.

## Invariant Synthesis Algorithms

Most algorithms for generating invariants are based on the **generate-test-repair** approach:

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

## Invariant Synthesis: References

We will not cover invariant synthesis algorithms in this course.

Literature on invariant synthesis:

- ▶ DISCOPLAN (Gerevini & Schubert, 1998)
- ▶ TIM (Fox & Long, 1998)
- ▶ Edelkamp & Helmert's algorithm (1999)
- ▶ Bonet & Geffner's algorithm (2001)
- ▶ Rintanen's algorithm (2008)
- ▶ Rintanen's algorithm for schematic invariants (2017)

## Exploiting Invariants

Invariants have many uses in planning:

- ▶ **Regression search (C3–C4):**  
**Prune subgoals** that violate (are inconsistent with) invariants.
- ▶ **Planning as satisfiability (C5–C6):**  
**Add invariants** to a SAT encoding of a planning task to get tighter constraints.
- ▶ **Proving unsolvability:**  
If  $\varphi$  is an invariant such that  $\varphi \wedge \gamma$  is **unsatisfiable**, the planning task with goal  $\gamma$  is unsolvable.
- ▶ **Finite-Domain Reformulation:**  
Derive a **more compact** FDR representation (equivalent, but with fewer states) from a given propositional planning task.

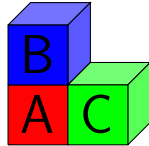
We now discuss the last point because it connects to our discussion of propositional vs. FDR planning tasks.

## E2.3 Mutexes

## Reminder: Blocks World (Propositional Variables)

### Example

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



$\rightsquigarrow 2^9 = 512 \text{ states}$

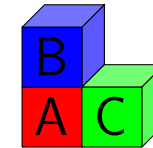
## Reminder: Blocks World (Finite-Domain Variables)

### Example

Use three finite-domain state variables:

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

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



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

## Task Reformulation

- ▶ Common modeling languages (like PDDL) often give us **propositional** tasks.
- ▶ More compact FDR tasks are often desirable.
- ▶ Can we do an **automatic reformulation**?

## Mutexes

Invariants that take the form of **binary clauses** are called **mutexes** because they express that certain variable assignments cannot be simultaneously true (are **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 **set of literals** is a **mutex group** if every subset of two literals is a mutex.

### Example (Blocks World)

$\{A\text{-on-}B, A\text{-on-}C, A\text{-on-table}\}$  is a mutex group.

## Encoding Mutex Groups as Finite-Domain Variables

Let  $G = \{\ell_1, \dots, \ell_n\}$  be a mutex group over  $n$  different propositional state variables  $V_G = \{v_1, \dots, v_n\}$ .

Then a single **finite-domain** state variable  $v_G$  with  $\text{dom}(v_G) = \{\ell_1, \dots, \ell_n, \text{none}\}$  can replace the  $n$  variables  $V_G$ :

- ▶  $s(v_G) = \ell_i$  represents situations where (exactly)  $\ell_i$  is true
- ▶  $s(v_G) = \text{none}$  represents situations where all  $\ell_i$  are false

**Note:** We can omit the “none” value if  $\ell_1 \vee \dots \vee \ell_n$  is an invariant.

## Mutex Covers

### Definition (Mutex Cover)

A **mutex cover** for a propositional planning task  $\Pi$  is a set of mutex groups  $\{G_1, \dots, G_n\}$  where each variable of  $\Pi$  occurs in exactly one group  $G_i$ .

A mutex cover is **positive** if all literals in all groups are positive.

**Note:** always exists (use trivial group  $\{v\}$  if  $v$  otherwise uncovered)

## Positive Mutex Covers

In the following, we stick to **positive** mutex covers for simplicity.

If we have  $\neg v$  in  $G$  for some group  $G$  in the cover, we can reformulate the task to use an “opposite” variable  $\hat{v}$  instead, as in the conversion to positive normal form ([Chapter B5](#)).

## E2.4 Reformulation

## Mutex-Based Reformulation of Propositional Tasks

Given a **conflict-free** propositional planning task  $\Pi$  with positive mutex cover  $\{G_1, \dots, G_n\}$ :

- ▶ In all **conditions** where variable  $v \in G_i$  occurs, replace  $v$  with  $v_{G_i} = v$ .
- ▶ In all effects  $e$  where variable  $v \in G_i$  occurs,
  - ▶ Replace all **atomic add effects**  $v$  with  $v_{G_i} := v$
  - ▶ Replace all **atomic delete effects**  $\neg v$  with  $(v_{G_i} = v \wedge \neg \bigvee_{v' \in G_i \setminus \{v\}} \text{effcond}(v', e)) \supset v_{G_i} := \text{none}$

This results in an FDR planning task  $\Pi'$  that is equivalent to  $\Pi$  (without proof).

**Note:** the conditional effects encoding delete effects can often be simplified away to an unconditional or empty effect.

## And Back?

- ▶ It can also be useful to reformulate an **FDR task into a propositional task**.
- ▶ For example, we might want positive normal form, which requires a propositional task.
- ▶ Key idea: each variable/value combination  $v = d$  becomes a separate propositional state variable  $\langle v, d \rangle$

## Converting FDR Tasks into Propositional Tasks

### Definition (Induced Propositional Planning Task)

Let  $\Pi = \langle V, I, O, \gamma \rangle$  be a **conflict-free** FDR planning task.

The **induced propositional planning task**  $\Pi'$  is the propositional 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  $\gamma'$  are obtained from  $O$  and  $\gamma$  by
  - ▶ replacing each atomic formula  $v = d$  by the proposition  $\langle v, d \rangle$
  - ▶ replacing each atomic effect  $v := d$  by the effect  $\langle v, d \rangle \wedge \bigwedge_{d' \in \text{dom}(v) \setminus \{d\}} \neg \langle v, d' \rangle$ .

**Notes:**

- ▶ Again, simplifications are often possible to avoid introducing so many delete effects.
- ▶  $\text{SAS}^+$  tasks induce STRIPS tasks.

## E2.5 Summary

## Summary (1)

- ▶ **Invariants** are common properties of all reachable states, expressed as 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 with reasonable (polynomial) computational effort.

## Summary (2)

- ▶ **Mutexes** are invariants that express that certain literals are mutually exclusive.
- ▶ **Mutex covers** provide a way to convert a set of propositional state variables into a potentially much smaller set of finite-domain state variables.
- ▶ Using mutex covers, we can **reformulate propositional tasks** as more compact FDR tasks.
- ▶ Conversely, we can **reformulate FDR tasks** as propositional tasks by introducing propositions for each variable/value pair.