Planning and Optimization

A7. Invariants and Mutexes

Malte Helmert and Gabriele Röger

Universität Basel

October 10, 2016

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 a logical formulas φ that are true in all reachable states.
  - Example: $\varphi = \neg(\mathit{at\text{-}uni} \wedge \mathit{at\text{-}home})$
- 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 of Π.

Computing Invariants

Computing Invariants

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.
  $\rightsquigarrow$ 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 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.

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 $\neg A\text{-}on\text{-}B \lor \neg A\text{-}on\text{-}C$ states that $A\text{-}on\text{-}B$ and $A\text{-}on\text{-}C$ 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 $\{B\text{-}on\text{-}A, C\text{-}on\text{-}A, D\text{-}on\text{-}A, A\text{-}clear\}$ 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.

Summary

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 with reasonable (polynomial) computational effort.
- Mutexes are invariants that express that certain pairs of literals are mutually exclusive.