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 $\varphi$ that is true in all reachable states.

Example: $\varphi = \neg(at\text{-}uni \land at\text{-}home)$

Such formulae are called invariants of the task.
Invariant synthesis algorithms

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 $\varphi$ 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.

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 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.

Invariant synthesis: references

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)

2 Planning tasks in finite-domain representation

- Mutexes
- FDR planning tasks
- Relationship to propositional planning tasks
- SAS* planning tasks
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 \( \neg A\text{-}\text{on-}B \lor \neg A\text{-}\text{on-}C \) states that \( A\text{-}\text{on-}B \) and \( A\text{-}\text{on-}C \) are mutex.

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

**Example (Blocksworld)**
Every pair in \{\( B\text{-}\text{on-}A \), \( C\text{-}\text{on-}A \), \( D\text{-}\text{on-}A \), \( A\text{-}\text{clear} \)\} is mutex.

---

### 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 \( D_v \) for the domain of \( v \).

**Example**
\( v = \text{above-}a \), \( D_{\text{above-}a} = \{b, c, d, \text{nothing} \} \)

This state variable encodes the same information as the propositional variables \( B\text{-}\text{on-}A \), \( C\text{-}\text{on-}A \), \( D\text{-}\text{on-}A \) and \( A\text{-}\text{clear} \).
Finite-domain formulae

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 $d \in D_v$.

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

Finite-domain effects

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 $d \in D_v$.

Example
The effect $(\text{below-a} := \text{table}) \land (\text{above-b} = a) \triangleright (\text{above-b} := \text{nothing})$ corresponds to the effect $A\text{-on-T} \land \neg A\text{-on-B} \land \neg A\text{-on-C} \land \neg A\text{-on-D} \land (A\text{-on-B} \triangleright B\text{-clear})$.

Planning tasks in finite-domain representation

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
- $\gamma$ is a formula over $V$ describing the goal states.

Relationship to propositional planning tasks

Definition (induced propositional planning task)
Let $\Pi = \langle V, I, O, \gamma \rangle$ be an FDR planning task. The induced propositional planning task $\Pi' = \langle A', I', O', \gamma' \rangle$, where
- $A' = \{(v, d) \mid v \in V, d \in D_v\}$
- $I'(v, d) = 1$ iff $I(v) = d$
- $O'$ and $\gamma'$ are obtained from $O$ and $\gamma$ by replacing
  - each atomic formula $v = d$ with the proposition $(v, d)$,
  - each atomic effect $v := d$ with the effect $(v, d) \land \bigwedge_{d' \in D_v \setminus \{d\}} \neg (v, d')$.

\rightarrow definition of finite-domain operators follows from this.

\rightarrow definition of finite-domain operators follows from this.
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 $\gamma$ 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⁺

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.