# Planning and Optimization

## E1. Planning Tasks in Finite-Domain Representation

Malte Helmert and Gabriele Röger

Universität Basel

November 2, 2026

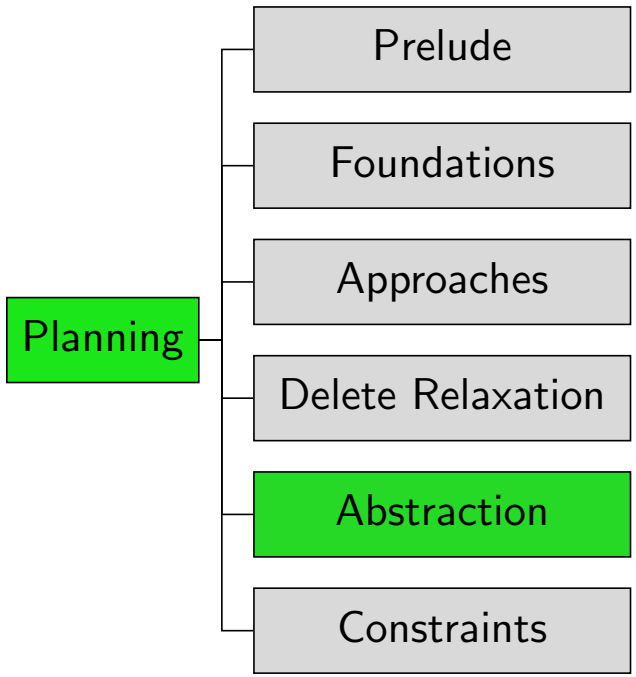
## E1.1 Finite-Domain Representation

## E1.2 Equivalence and Normal Forms

## E1.3 Summary

# How We Continue

- The next class of heuristics we will consider are abstraction heuristics.




- However, this requires some preparations.

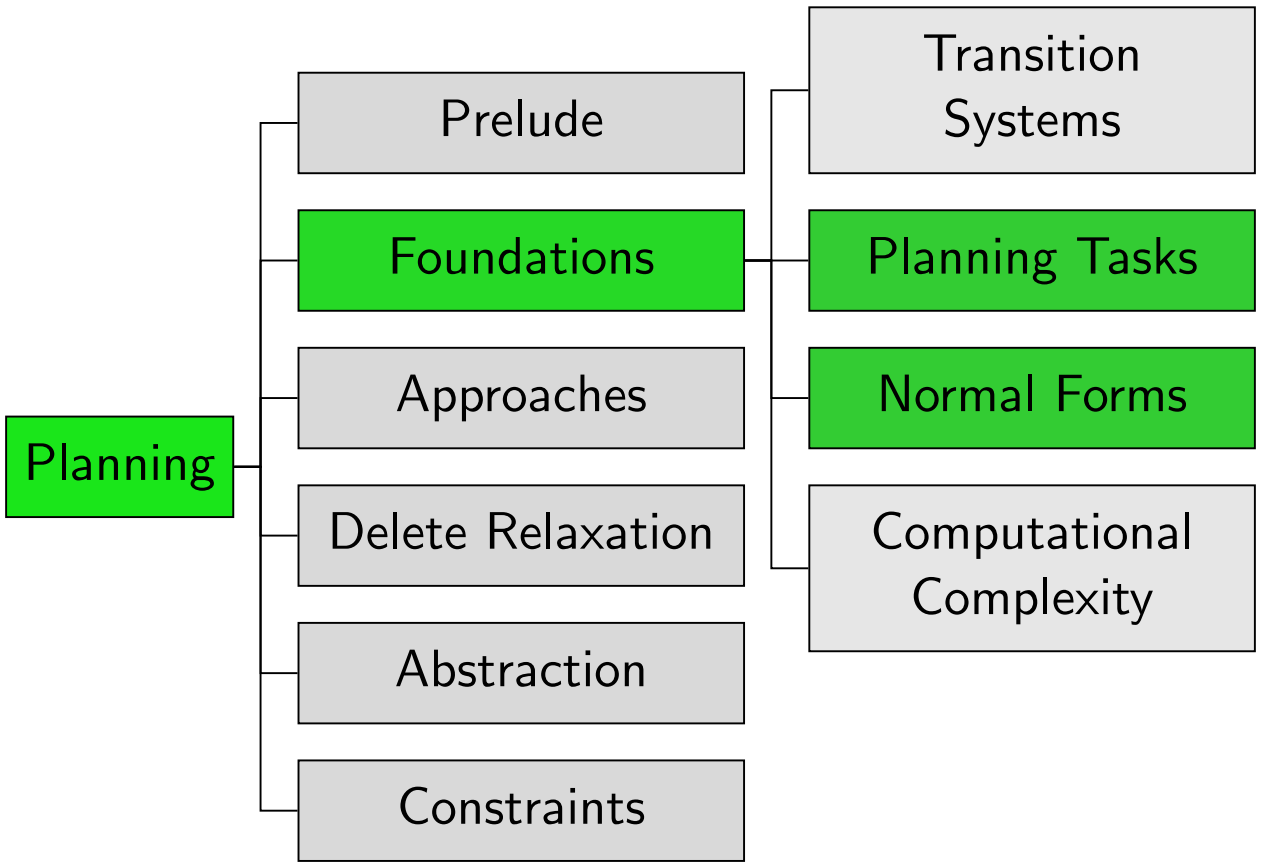
# Back to Foundations: Finite-Domain Representation

- Abstraction heuristics benefit from a more compact task representation, called finite-domain representation.
- To understand the relationship to the propositional task representation, we need to know a special kind of invariants, namely mutexes.

$\rightsquigarrow$ We first get to know finite-domain representation (this chapter) and then speak about invariants and transformations between the representations (next chapter).

$\rightsquigarrow$ not specific to abstraction heuristics, but general foundations

# Content of the Course



# E1.1 Finite-Domain Representation

# Finite-Domain State Variables

- So far, we used propositional (Boolean) state variables.
  $\rightsquigarrow$ possible values **T** and **F**
- We now consider finite-domain variables.
  $\rightsquigarrow$ every variable has a finite set of possible values
- A state is still an assignment to the state variables.

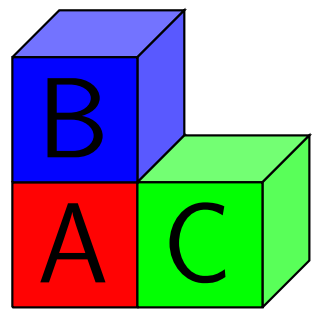
Example: $O(n^2)$ Boolean variables or $O(n)$ finite-domain variables with domain size $O(n)$ suffice for blocks world with $n$ blocks.

# Blocks World State with Propositional Variables

## Example

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

$\rightsquigarrow 2^9 = 512$ states




Note: it may be useful to add auxiliary state variables like *A-clear*.

# Blocks World State with Finite-Domain Variables

**Example**

Use three finite-domain state variables:

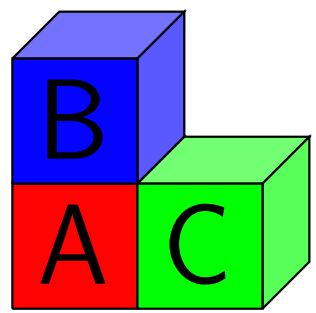
- *below-a*: $\{\mathsf{b}, \mathsf{c}, \mathsf{table}\}$
- *below-b*: $\{\mathsf{a}, \mathsf{c}, \mathsf{table}\}$
- *below-c*: $\{\mathsf{a}, \mathsf{b}, \mathsf{table}\}$

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

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

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

$\rightsquigarrow 3^3 = 27$ states




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

# Advantage of Finite-Domain Representation

How many "useless" (physically impossible) states are there with these blocks world state representations?

- There are 13 physically possible states with three blocks:
  - all blocks on table: 1 state
  - all blocks in one stack: $3! = 6$ states
  - two block stacked, the other separate: $\binom{3}{2}2! = 6$
- With propositional variables, $2^9 - 13 = 499$ states are useless.
- With finite-domain variables, only $27 - 13 = 14$ are useless.

Although useless states are unreachable,
they can introduce "shortcuts" in some heuristics
and thus lead to worse heuristic estimates.

# Finite-Domain State Variables

## Definition (Finite-Domain State Variable)

A finite-domain state variable is a symbol $v$ with an associated domain $\mathrm{dom}(v)$, which is a finite non-empty set of values.

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

A state $s$ over $V$ is an assignment $s : V \to \bigcup_{v \in V} \mathrm{dom}(v)$ such that $s(v) \in \mathrm{dom}(v)$ for all $v \in V$.

A formula over $V$ is a propositional logic formula whose atomic propositions are of the form $v = d$ where $v \in V$ and $d \in \mathrm{dom}(v)$.

Slightly extending propositional logic, we treat states $s$ over finite-domain variables as logical interpretations where $s \models v = d$ iff $s(v) = d$.

# Example: Finite-Domain State Variables

**Example**

Consider finite-domain variables $V = \{\mathit{location}, \mathit{bike}\}$ with $\mathrm{dom}(\mathit{location}) = \{\text{at-home}, \text{in-front-of-uni}, \text{in-lecture}\}$ and $\mathrm{dom}(\mathit{bike}) = \{\text{locked}, \text{unlocked}, \text{stolen}\}$.

Consider state $s = \{\mathit{location} \mapsto \text{at-home}, \mathit{bike} \mapsto \mathit{locked}\}$.

Does $s \models (\mathit{location} = \text{at-home} \wedge \neg \mathit{bike} = \text{stolen})$ hold?

# Reminder: Syntax of Operators

**Definition (Operator)**

An operator $o$ over state variables $V$ is an object with three properties:

- a precondition $pre(o)$, a formula over $V$
- an effect $eff(o)$ over $V$
- a cost $cost(o) \in \mathbb{R}_0^+$

Only necessary adaptation: What is an effect?

**Example**

$\langle location = \text{in-front-of-uni},$
$location := \text{in-lecture} \wedge (bike = \text{unlocked} \rhd bike := \text{stolen}), 1\rangle$

# Syntax of Effects

**Definition (Effect over Finite-Domain State Variables)**

Effects over finite-domain state variables $V$
are inductively defined as follows:

- $\top$ is an effect (empty effect).
- If $v \in V$ is a finite-domain state variable and $d \in \mathrm{dom}(v)$, then $v := d$ is an effect (atomic effect).
- If $e$ and $e'$ are effects, then $(e \wedge e')$ is an effect (conjunctive effect).
- If $\chi$ is a formula over $V$ and $e$ is an effect, then $(\chi \rhd e)$ is an effect (conditional effect).

Parentheses can be omitted when this does not cause ambiguity.

only change compared to propositional case: atomic effects

# Semantics of Effects: Trigger Conditions

## Definition (Trigger Condition with Finite-Domain Representation)

Let $v := d$ be an atomic effect, and let $e$ be an effect.

The trigger condition or trigger $\mathit{trig}(v := d, e)$
for the firing of $v := d$ given the effect $e$
is a propositional formula defined as follows:

- $\mathit{trig}(v := d, \top) = \bot$
- $\mathit{trig}(v := d, v := d) = \top$
- $\mathit{trig}(v := d, v' := d') = \bot$
  for atomic effects with $v' \neq v$ or $d' \neq d$
- $\mathit{trig}(v := d, (e \wedge e')) =$
  $(\mathit{trig}(v := d, e) \vee \mathit{trig}(v := d, e'))$
- $\mathit{trig}(v := d, (\chi \rhd e)) = (\chi \wedge \mathit{trig}(v := d, e))$

Same definition as for propositional tasks,
we just use the adapted definition of atomic effects.

# Conflicting Effects and Consistency Condition

- What should an effect of the form $v := a \wedge v := b$ mean?
- For finite-domain representations, the accepted semantics is to make this illegal, i.e., to make an operator inapplicable if it would lead to conflicting effects.

**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

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

How did we handle conflicting effects in propositional planning tasks?

# Semantics of Operators: Finite-Domain Case

## Definition (Applicable, Resulting State)

Let $V$ be a set of finite-domain state variables
and $e$ be an effect over $V$.
If $s \models \mathit{consist}(e)$, the resulting state of applying $e$ in $s$,
written $s[\![e]\!]$, is the state $s'$ defined as follows for all $v \in V$:

$$s'(v) = \begin{cases} d & \text{if } s \models \mathit{trig}(v := d, e) \text{ for some } d \in \mathrm{dom}(v) \\ s(v) & \text{otherwise} \end{cases}$$

Let $o$ be an operator over $V$.
Operator $o$ is applicable in $s$ if $s \models \mathit{pre}(o) \wedge \mathit{consist}(\mathit{eff}(o))$.

If $o$ is applicable in $s$, the resulting state of applying $o$ in $s$,
written $s[\![o]\!]$, is the state $s[\![\mathit{eff}(o)]\!]$.

# Applying Operators: Example

**Example**

$V = \{\mathit{location}, \mathit{bike}\}$ with
$\mathrm{dom}(\mathit{location}) = \{\text{at-home}, \text{in-front-of-uni}, \text{in-lecture}\}$ and
$\mathrm{dom}(\mathit{bike}) = \{\text{locked}, \text{unlocked}, \text{stolen}\}$.

State $s = \{\mathit{location} \mapsto \text{in-front-of-uni}, \mathit{bike} \mapsto \text{unlocked}\}$

$o = \langle \mathit{location} = \text{in-front-of-uni}, \mathit{location} := \text{at-home}, 1 \rangle$

$o' = \langle \mathit{location} = \text{in-front-of-uni},$
$\quad \mathit{location} := \text{in-lecture} \wedge (\mathit{bike} = \text{unlocked} \rhd \mathit{bike} := \text{stolen}), 1 \rangle$

What is $s[\![o]\!]$? What is $s[\![o']\!]$?

# FDR Planning Tasks

## Definition (Planning Task)

An FDR planning task (or planning task in finite-domain representation) 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 assignment for $V$ called the initial state,
- $O$ is a finite set of operators over $V$, and
- $\gamma$ is a formula over $V$ called the goal.

Apart from the variables, this is the same definition as for propositional planning tasks, but the underlying concepts have been adapted.

# Mapping FDR Planning Tasks to Transition Systems

## Definition (Transition System Induced by an FDR Planning Task)

The FDR 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 states over $V$,
- $L$ is the set of operators $O$,
- $c(o) = cost(o)$ 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
- $S_\star = \{s \in S \mid s \models \gamma\}$.

Exactly the same definition as for propositional planning tasks, but the underlying concepts have been adapted.

# E1.2 Equivalence and Normal Forms

# Equivalence and Flat Operators

- The definitions of equivalent effects/operators and flat effects/operators apply equally to finite-domain representation.
- The same is true for the equivalence transformations.

You find the definitions and transformations in Chapter B4.

# Conflict-Free Operators

**Definition (Conflict-Free)**

An effect $e$ over finite-domain state variables $V$
is called conflict-free if $\mathit{trig}(v := d, e) \wedge \mathit{trig}(v := d', e)$
is unsatisfiable for all $v \in V$ and $d, d' \in \mathrm{dom}(v)$ with $d \neq d'$.

An operator $o$ is called conflict-free if $\mathit{eff}(o)$ is conflict-free.

Note: $\mathit{consist}(e) \equiv \top$ for conflict-free $e$.

Algorithm to make given operator $o$ conflict-free:

- replace $\mathit{pre}(o)$ with $\mathit{pre}(o) \wedge \mathit{consist}(\mathit{eff}(o))$
- replace all atomic effects $v := d$ by $(\mathit{consist}(\mathit{eff}(o)) \rhd v := d)$

The resulting operator $o'$ is conflict-free and $o \equiv o'$.

# $SAS^+$ Operators and Planning Tasks

## Definition ($SAS^+$ Operator)

An operator $o$ of an FDR planning task is a $SAS^+$ operator if

- $pre(o)$ is a satisfiable conjunction of atoms, and
- $eff(o)$ is a conflict-free conjunction of atomic effects.

## Definition ($SAS^+$ Planning Task)

An FDR planning task $\langle V, O, I, \gamma \rangle$ is a $SAS^+$ planning task
if all operators $o \in O$ are $SAS^+$ operators
and $\gamma$ is a satisfiable conjunction of atoms.

Note: $SAS^+$ operators are conflict-free and flat.

# SAS$^+$ Operators: Remarks

- Every SAS$^+$ operator is of the form

$$\langle v_1 = d_1 \wedge \cdots \wedge v_n = d_n, \quad v'_1 := d'_1 \wedge \cdots \wedge v'_m := d'_m \rangle$$

  where all $v_i$ are distinct and all $v'_j$ are distinct.
- Often, SAS$^+$ operators $o$ are described via two sets of partial assignments:
  - the preconditions $\{v_1 \mapsto d_1, \ldots, v_n \mapsto d_n\}$
  - the effects $\{v'_1 \mapsto d'_1, \ldots, v'_m \mapsto d'_m\}$

# $SAS^+$ vs. STRIPS

- $SAS^+$ is an analogue of STRIPS planning tasks for FDR, but there is no special role of "positive" conditions.
- Apart from this difference, all comments for STRIPS apply analogously.
- If all variable domains are binary, $SAS^+$ is essentially STRIPS with negation.

**$SAS^+$**

Derives from SAS = Simplified Action Structures
(Bäckström & Klein, 1991)

# E1.3 Summary

# Summary

- 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.