Planning and Optimization

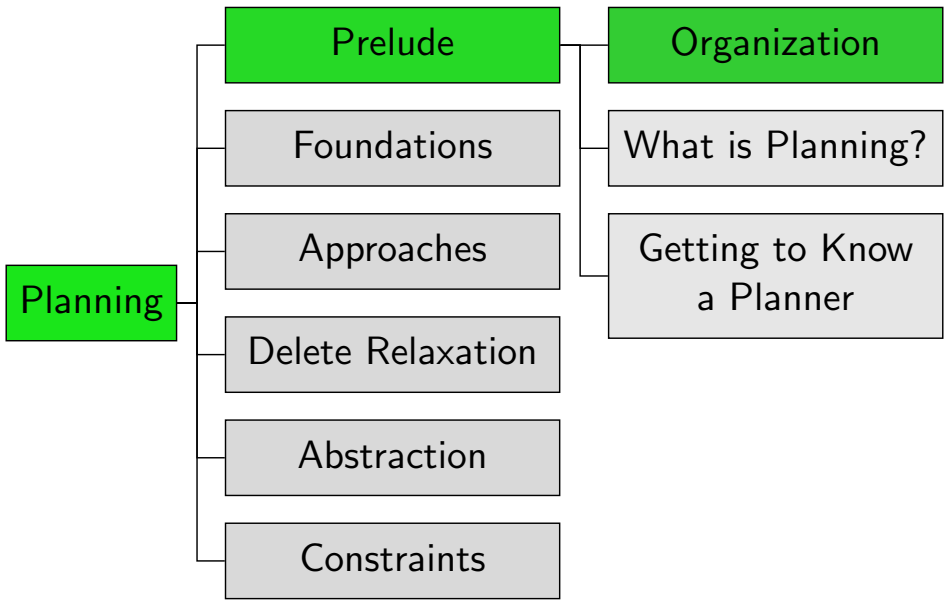
A1. Organizational Matters

Malte Helmert and Gabriele Röger

Universität Basel

September 17, 2025

Content of the Course



People & Coordinates

People: Lecturers

Malte Helmert

Gabriele Röger

Lecturers

Malte Helmert

- email: `malte.helmert@unibas.ch`
- office: room 06.004, Spiegelgasse 1

Gabriele Röger

- email: `gabriele.roeger@unibas.ch`
- office: room 04.005, Spiegelgasse 1

People: Assistant

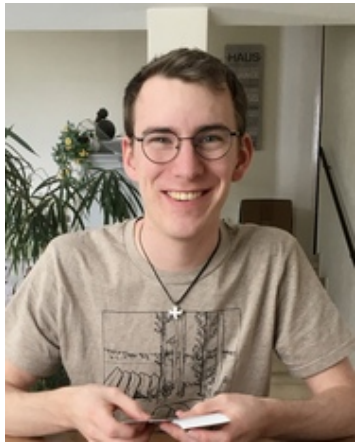
Tanja Schindler

Assistant

Tanja Schindler

- email: tanja.schindler@unibas.ch
- office: room 04.005, Spiegelgasse 1

People: Tutors



Clemens Büchner

Esther Mugdan

Tutors

Clemens Büchner

- email: clemens.buechner@unibas.ch
- office: room 04.001, Spiegelgasse 5

Esther Mugdan

- email: esther.mugdan@unibas.ch
- office: room 04.001, Spiegelgasse 5

Time & Place

Lectures

- time: Mon 14:15–16:00, Wed 14:15–16:00
- place: room 00.003, Spiegelgasse 1

Exercise Sessions

- time: Wed 16:15–18:00
- place: room 00.003, Spiegelgasse 1

first exercise session: today

Communication Channels

- lecture sessions (Mon, Wed)
- exercise sessions (Wed)
- course homepage
- ADAM workspace
- Discord server (invitation link on ADAM workspace)
- email

registration:

- `https://services.unibas.ch/`
- Please register today to receive all course-related emails!

Planning and Optimization Course on the Web

Course Homepage

`https://dmi.unibas.ch/en/studies/computer-science/course-offer-fall-semester-25/lecture-planning-and-optimization/`

- course information
- slides
- link to ADAM workspace
- bonus materials (not relevant for the exam)

Target Audience & Rules

Target Audience

target audience:

- M.Sc. Computer Science
  - Major in Machine Intelligence:
    module Concepts of Machine Intelligence
    module Methods of Machine Intelligence
  - Major in Distributed Systems:
    module Applications of Distributed Systems
- M.A. Computer Science ("Master-Studienfach")
  module Concepts of Machine Intelligence
- M.Sc. Data Science: module Electives in Data Science
- other students welcome

Prerequisites

prerequisites:

- general computer science background: good knowledge of
  - algorithms and data structures
  - complexity theory
  - mathematical logic
  - programming
- background in Artificial Intelligence:
  - Foundations of Artificial Intelligence course (13548)
  - in particular chapters on state-space search

Gaps?

$\rightsquigarrow$ talk to us to discuss a self-study plan to catch up

Exam

- written examination (105 min)
- date and time: January 28, 14:00–16:00
- place: Biozentrum, room U1.131
- 8 ECTS credits
- admission to exam: 50% of the exercise marks
- final grade based on exam exclusively
- no repeat exam (except in case of illness)

Exercise Sheets

exercise sheets (homework assignments):

- solved in groups of two or three ($3 < 4$), submitted in ADAM
- weekly homework assignments
  - released Monday before the lecture
  - have questions or need help?
    $\rightsquigarrow$ assistance provided in Wednesday exercises
  - not sure if you need help?
    $\rightsquigarrow$ start before Wednesday!
  - due following Monday at 23:59
- mixture of theory, programming and experiments
- range from basic understanding to research-oriented

Programming Exercises

programming exercises:

- part of regular assignments
- solutions that obviously do not work: 0 marks
- work with existing C++ and Python code

Exercise Sessions

exercise sessions:

- ask questions about current assignments (and course)
- work on homework assignments
- discuss past homework assignments

Plagiarism

Plagiarism

Plagiarism is presenting someone else's work, ideas, or words as your own, without proper attribution.

For example:

- Using someone's text without citation
- Paraphrasing too closely
- Using information from a source without attribution
- Passing off AI-generated content as your own original work

Plagiarism

Plagiarism

Plagiarism is presenting someone else's work, ideas, or words as your own, without proper attribution.

For example:

- Using someone's text without citation
- Paraphrasing too closely
- Using information from a source without attribution
- Passing off AI-generated content as your own original work

Long-term impact:

- You undermine your own learning.
- You start to lose confidence in your ability to think, write, and solve problems independently.
- Damage to academic reputation and professional consequences in future careers

Plagiarism in Exercises

- You may discuss material from the course, including the exercise assignments, with your peers.
- But: You have to independently write down your exercise solutions (in your team).
- Help from an LLM is acceptable to the same extent as it is acceptable from someone who is not a member of your team.

Plagiarism in Exercises

- You may discuss material from the course, including the exercise assignments, with your peers.
- But: You have to independently write down your exercise solutions (in your team).
- Help from an LLM is acceptable to the same extent as it is acceptable from someone who is not a member of your team.

Immediate consequences of plagiarism:

- 0 marks for the exercise sheet (first time)
- exclusion from exam (second time)

Plagiarism in Exercises

- You may discuss material from the course, including the exercise assignments, with your peers.
- But: You have to independently write down your exercise solutions (in your team).
- Help from an LLM is acceptable to the same extent as it is acceptable from someone who is not a member of your team.

Immediate consequences of plagiarism:

- 0 marks for the exercise sheet (first time)
- exclusion from exam (second time)

If in doubt: check with us what is (and isn't) OK before submitting
Exercises too difficult? We are happy to help!

Special Needs?

- We (and the university) strive for equality of students with disabilities or chronic illnesses.
- Contact the lecturers for small adaptations.
- Contact the Students Without Barriers (StoB) service point for general adaptations and disadvantage compensation.

Course Content

Learning Objectives

Learning Objectives

- get to know theoretical and algorithmic foundations of classical planning and work on practical implementations
- understand fundamental concepts underlying modern planning algorithms and theoretical relationships that connect them
- become equipped to understand research papers and conduct projects in this area

Course Material

course material:

- slides (online)
- no textbook
- additional material on request

Git Repository

- We use a git repository for programming exercises and for demos during the lecture.
- Setting up the repository is your first task for the exercises.

Demo Examples

When working with the repository, go to its base directory:

Base Directory for Demos and Exercises

```
$ cd planopt-hs25
```

One-time demo set-up (from the base directory)
if the necessary software is installed on your machine:

Demo Set-Up

```
$ cd demo/fast-downward
$ ./build.py
```

Under Construction...

- Advanced courses are close to the frontiers of research and therefore constantly change.
- We are always happy about feedback, corrections and suggestions!

Planning and Optimization

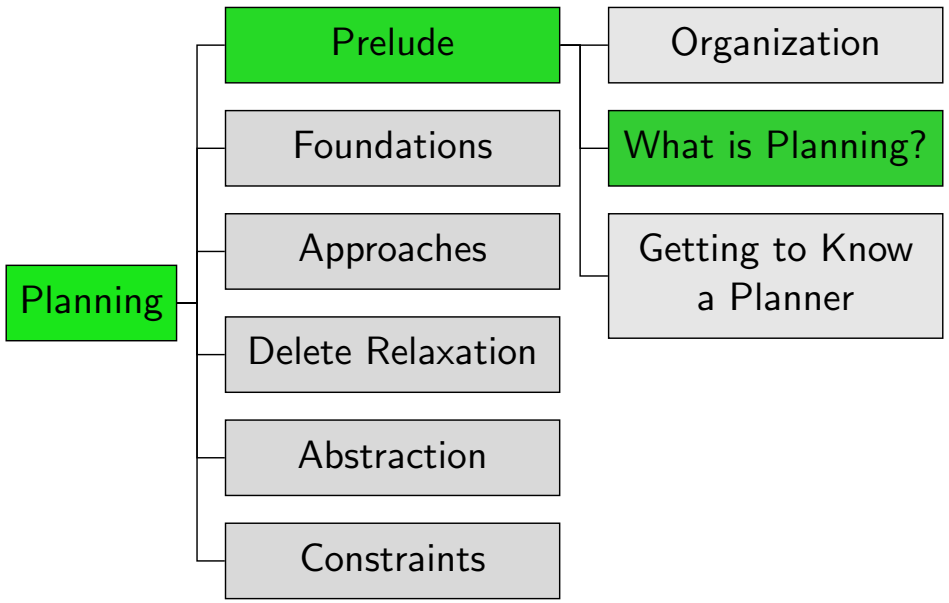
A2. What is Planning?

Malte Helmert and Gabriele Röger

Universität Basel

September 17, 2025

Content of the Course



Before We Start...

Prelude (Chapters A1–A3): very high-level intro to planning

- our goal: give you a little feeling what planning is about
- preface to the actual course

$\rightsquigarrow$ main course content (beginning with Chapter B1)
will be mathematically formal and rigorous

- You can ignore the prelude when preparing for the exam.

Planning

General Problem Solving

Wikipedia: General Problem Solver

General Problem Solver (GPS) was a computer program created in 1959 by Herbert Simon, J.C. Shaw, and Allen Newell intended to work as a universal problem solver machine.

Any formalized symbolic problem can be solved, in principle, by GPS. [...]

GPS was the first computer program which separated its knowledge of problems (rules represented as input data) from its strategy of how to solve problems (a generic solver engine).

$\rightsquigarrow$ these days called "domain-independent automated planning"
$\rightsquigarrow$ this is what the course is about

So What is Domain-Independent Automated Planning?

Automated Planning (Pithy Definition)

"Planning is the art and practice of thinking before acting."

— Patrik Haslum

Automated Planning (More Technical Definition)

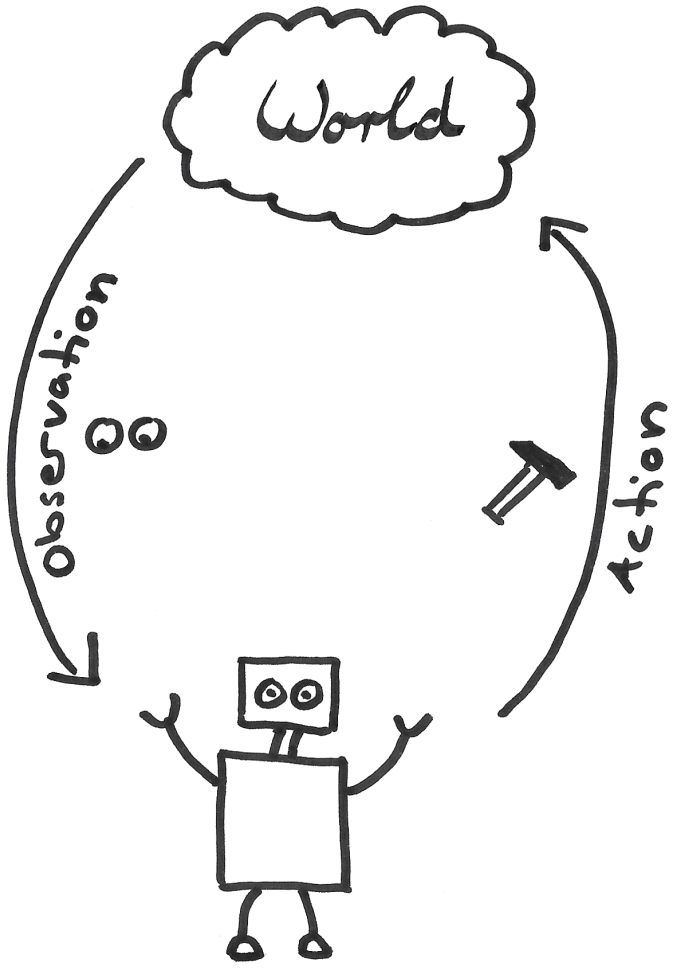
"Selecting a goal-leading course of action
based on a high-level description of the world."

— Jörg Hoffmann

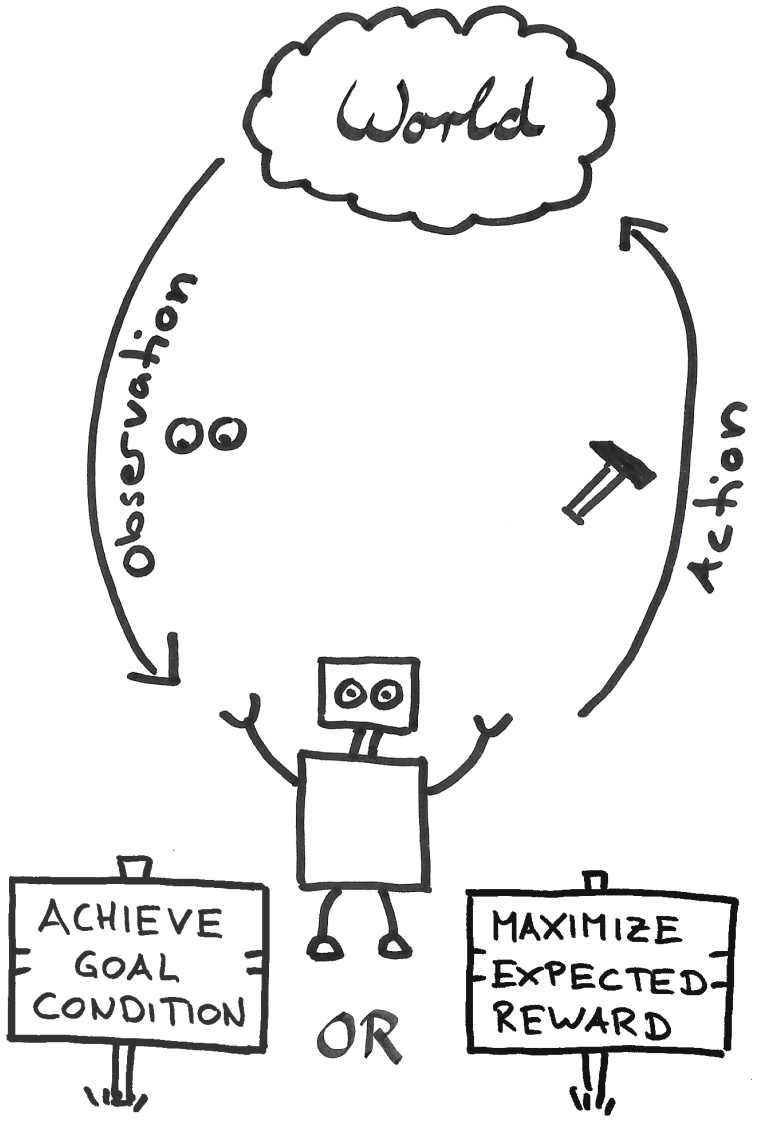
Domain-Independence of Automated Planning

Create one planning algorithm that performs sufficiently well on many application domains (including future ones).

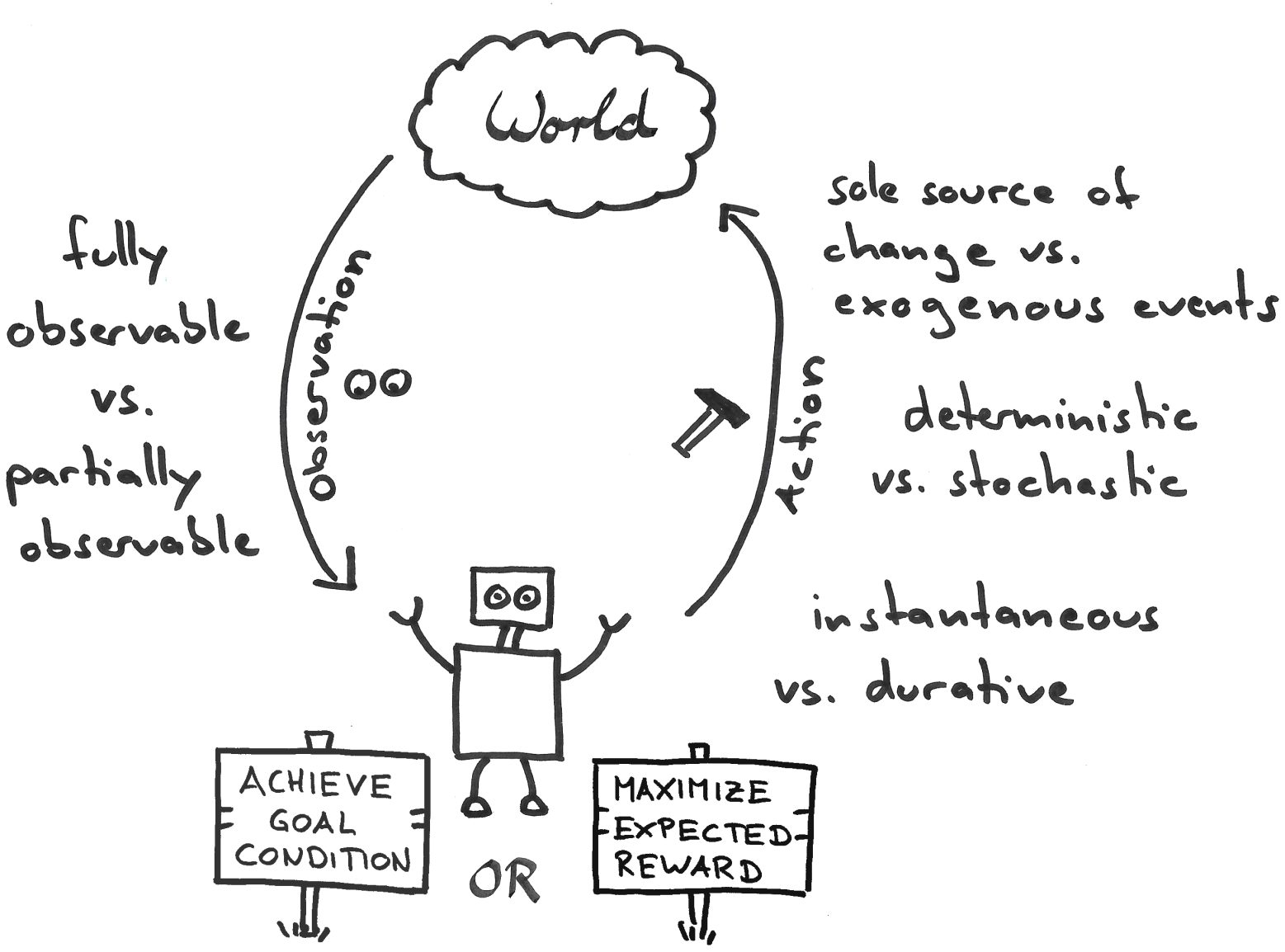
General Perspective on Planning



General Perspective on Planning



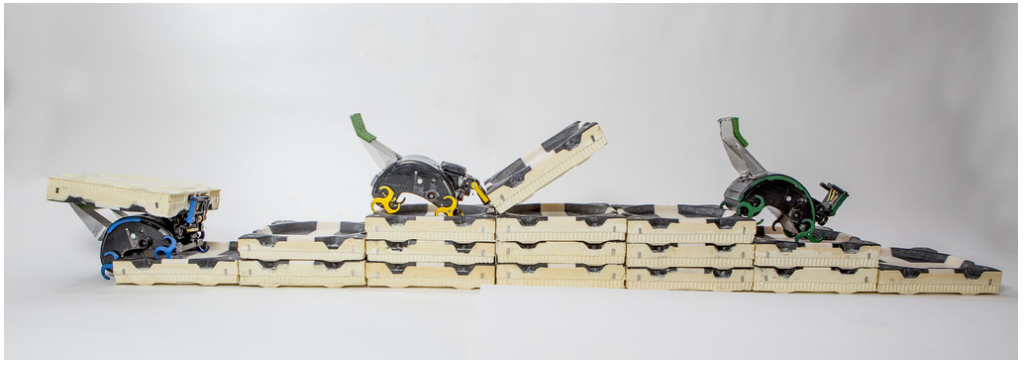
General Perspective on Planning



Example: Earth Observation

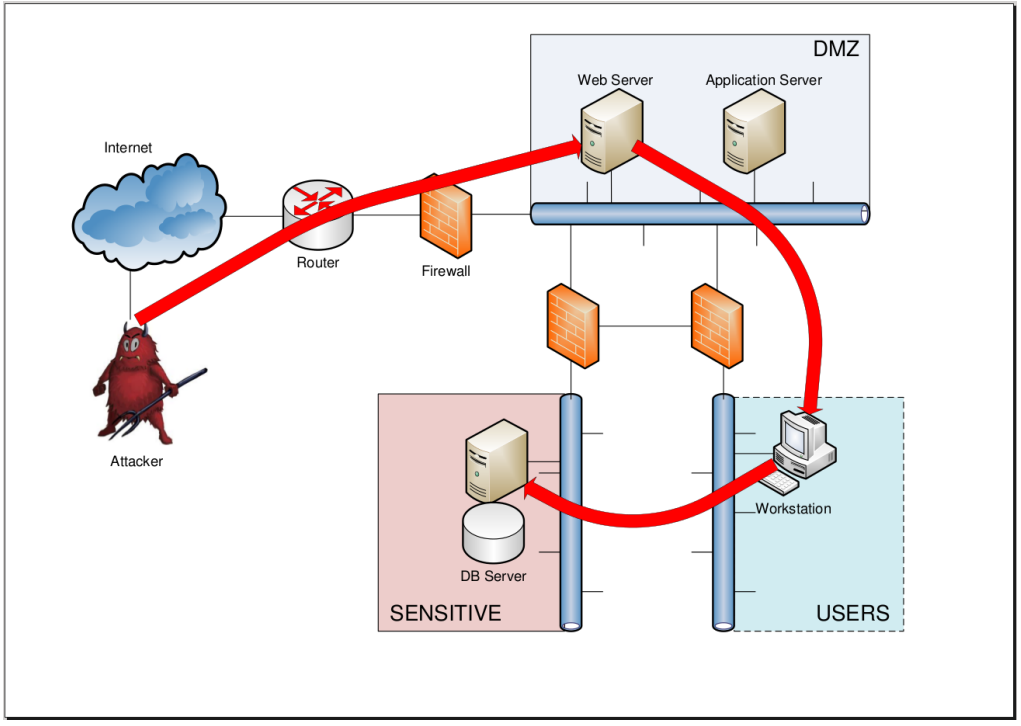
- satellite takes images of patches on Earth
- use weather forecast to optimize probability of high-quality images

Example: Termes



Harvard TERMES robots, based on termites

Example: Cybersecurity

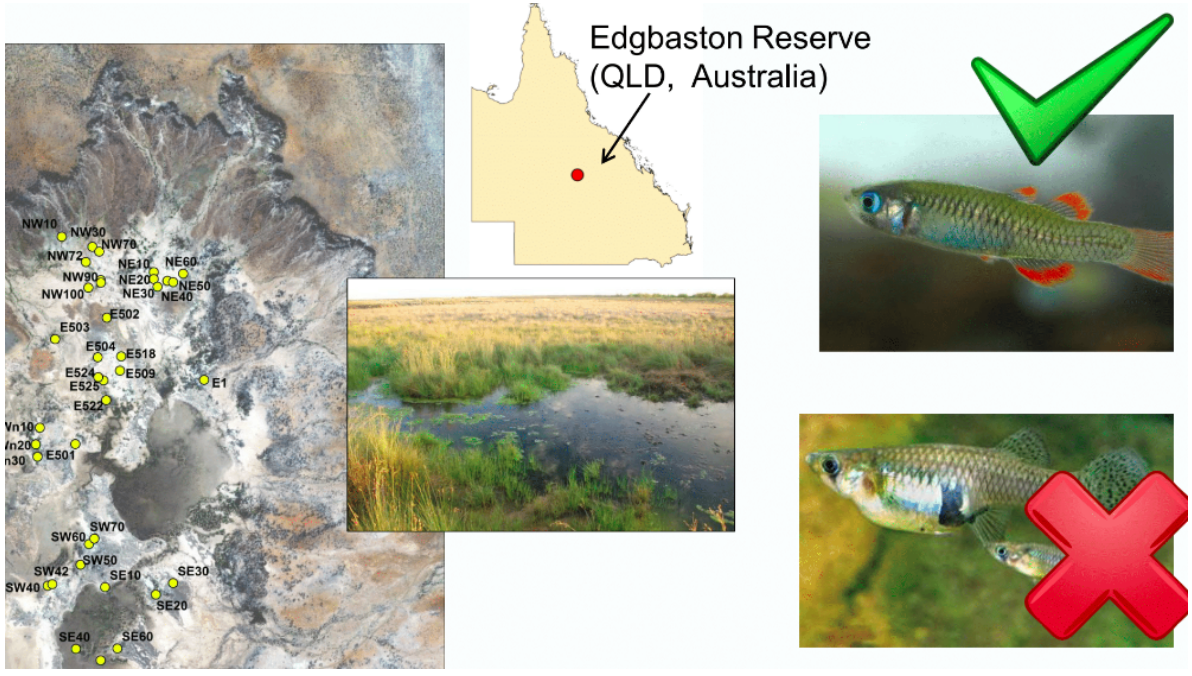


CALDERA automated adversary emulation system

Example: Intelligent Greenhouse

photo © LemnaTec GmbH

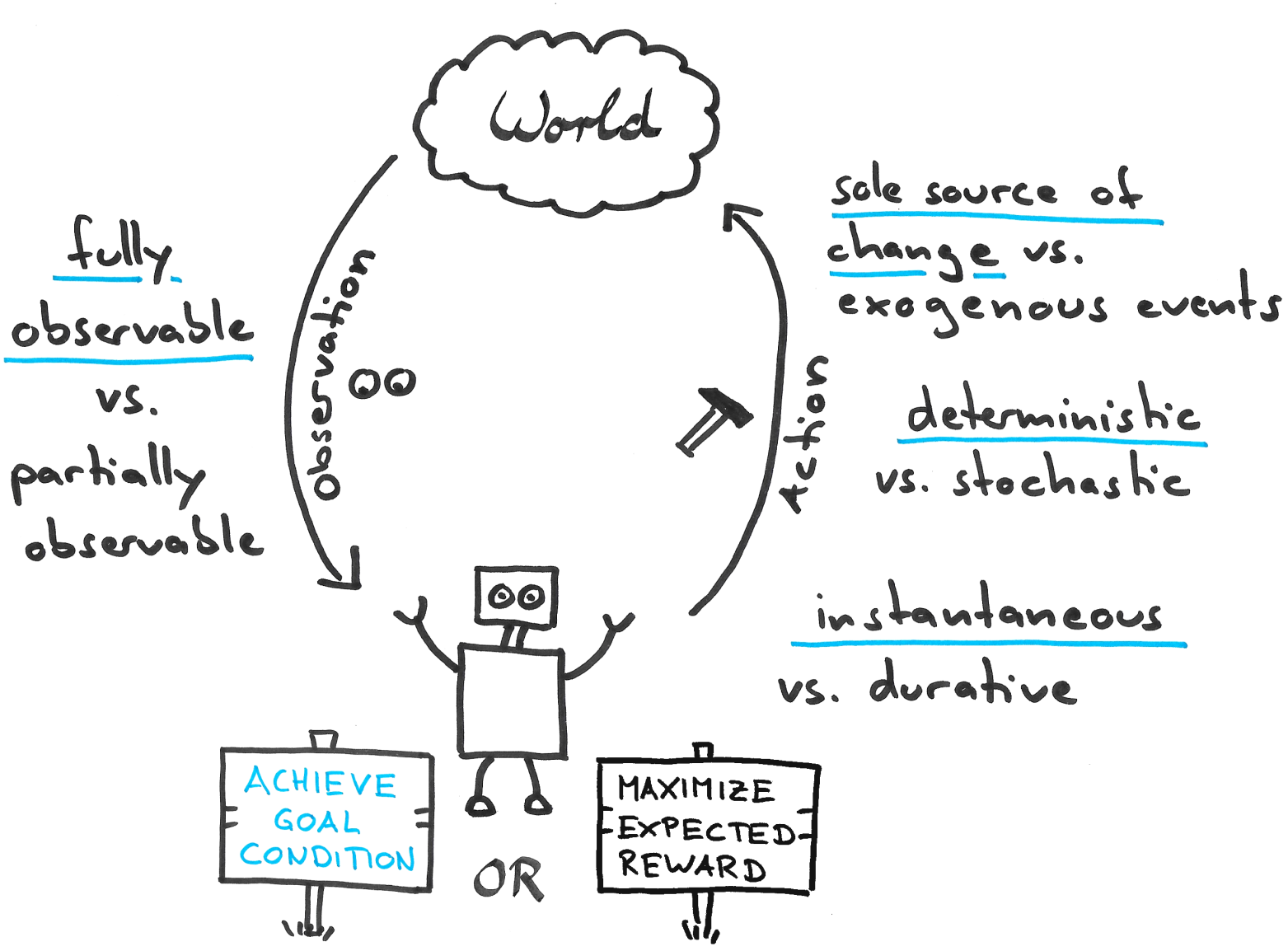
Example: Red-finned Blue-eye



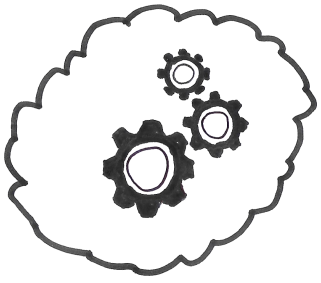
Picture by Iadine Chadès

- red-finned blue-eye population threatened by gambusia
- springs connected probabilistically during rain season
- find strategy to save red-finned blue-eye from extinction

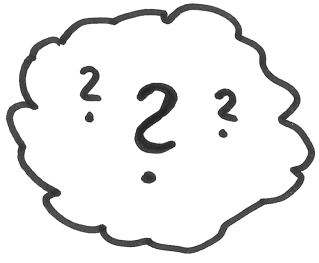
Classical Planning



Model-based vs. Data-driven Approaches



Model-based approaches know
the "inner workings" of the world
$\rightsquigarrow$ reasoning



Data-driven approaches rely only
on collected data from a black-box world
$\rightsquigarrow$ learning

We focus on model-based approaches.

Planning Tasks

input to a planning algorithm: planning task

- initial state of the world
- actions that change the state
- goal to be achieved

output of a planning algorithm:

- plan: sequence of actions taking initial state to a goal state
- or confirmation that no plan exists

$\rightsquigarrow$ formal definitions later in the course

The Planning Research Landscape

- one of the major subfields of Artificial Intelligence (AI)
- represented at major AI conferences (IJCAI, AAAI, ECAI)
- annual specialized conference ICAPS ($\approx$ 250 participants)
- major journals: general AI journals (AIJ, JAIR)

Classical Planning

This course covers classical planning:

- offline (static)
- discrete
- deterministic
- fully observable
- single-agent
- sequential (plans are action sequences)
- domain-independent

This is just one facet of planning.

Many others are studied in AI. Algorithmic ideas often (but not always) translate well to more general problems.

More General Planning Topics

More general kinds of planning include:

- ~~offline~~: online planning; planning and execution
- ~~discrete~~: continuous planning (e.g., real-time/hybrid systems)
- ~~deterministic~~: FOND planning; probabilistic planning
- ~~single-agent~~: multi-agent planning; general game playing; game-theoretic planning
- ~~fully observable~~: POND planning; conformant planning
- ~~sequential~~: e.g., temporal planning

Domain-dependent planning problems in AI include:

- pathfinding, including grid-based and multi-agent (MAPF)
- continuous motion planning

Planning Task Examples

Example: The Seven Bridges of Königsberg

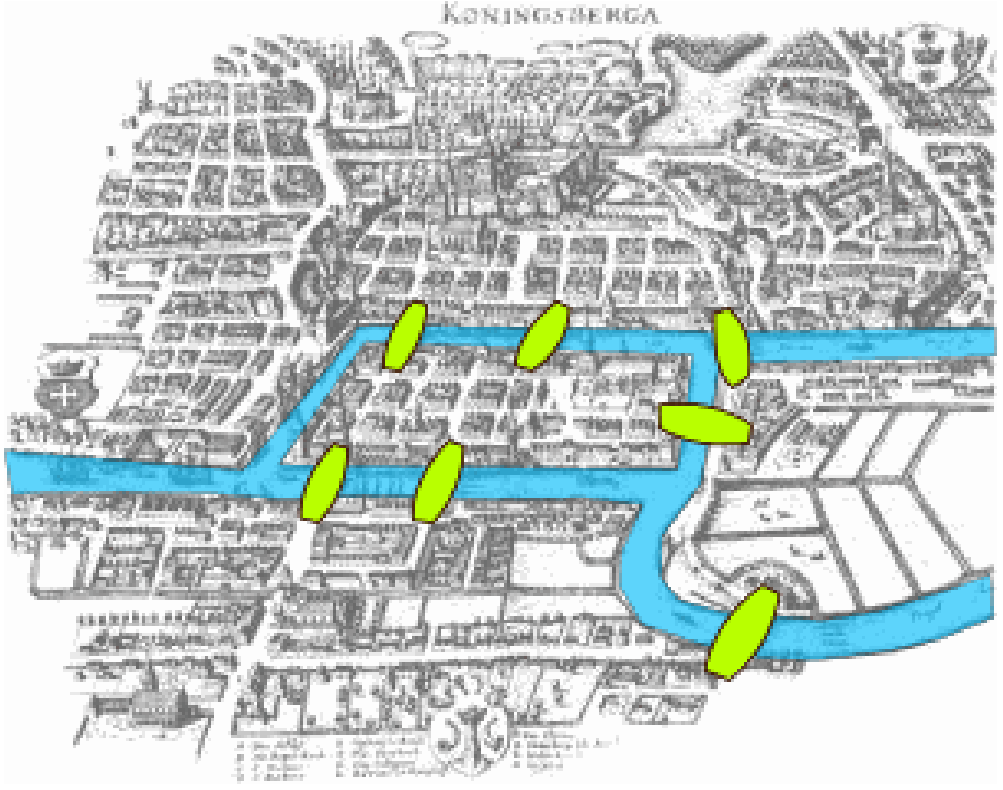


image credits: Bogdan Giuşcă (public domain)

Demo

```
$ ls demo/koenigsberg
```

Example: Intelligent Greenhouse

photo © LemnaTec GmbH

Demo

```
$ ls demo/ipc/scanalyzer-08-strips
```

Example: FreeCell

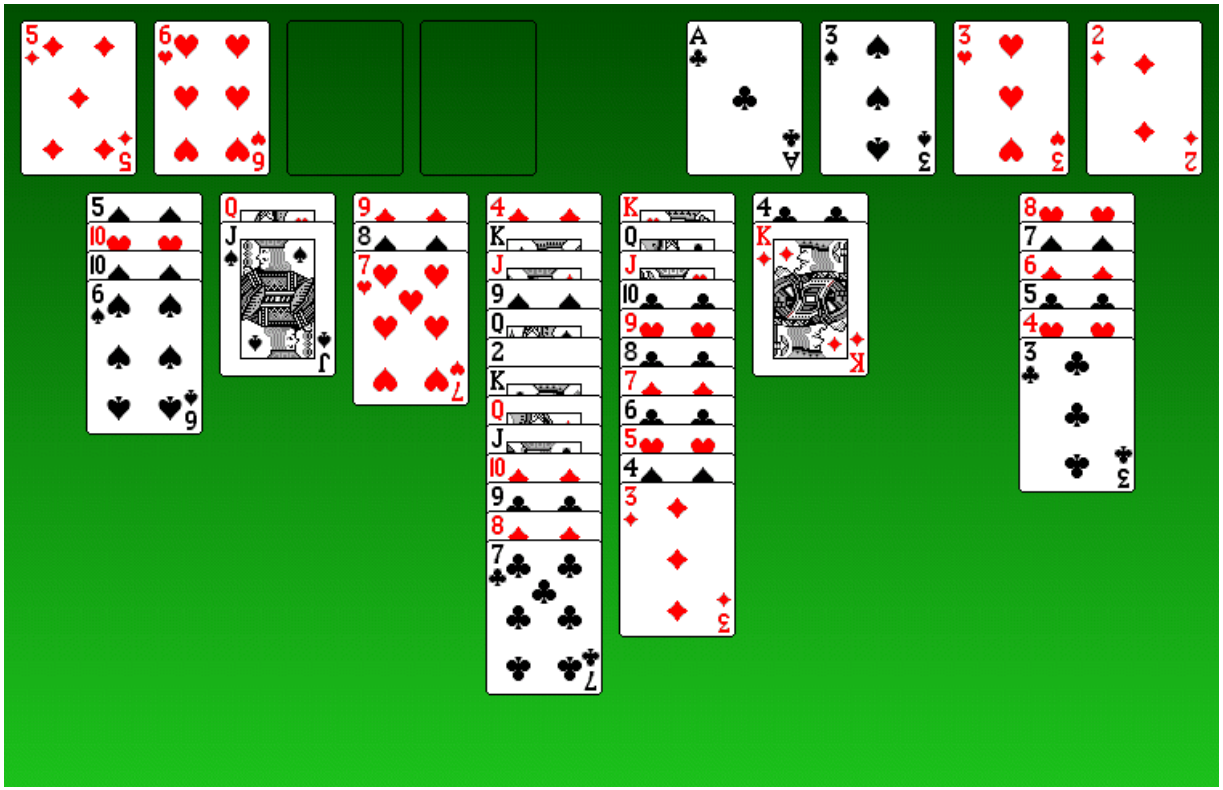


image credits: GNOME Project (GNU General Public License)

Demo Material

```
$ ls demo/ipc/freecell
```

Many More Examples

Demo

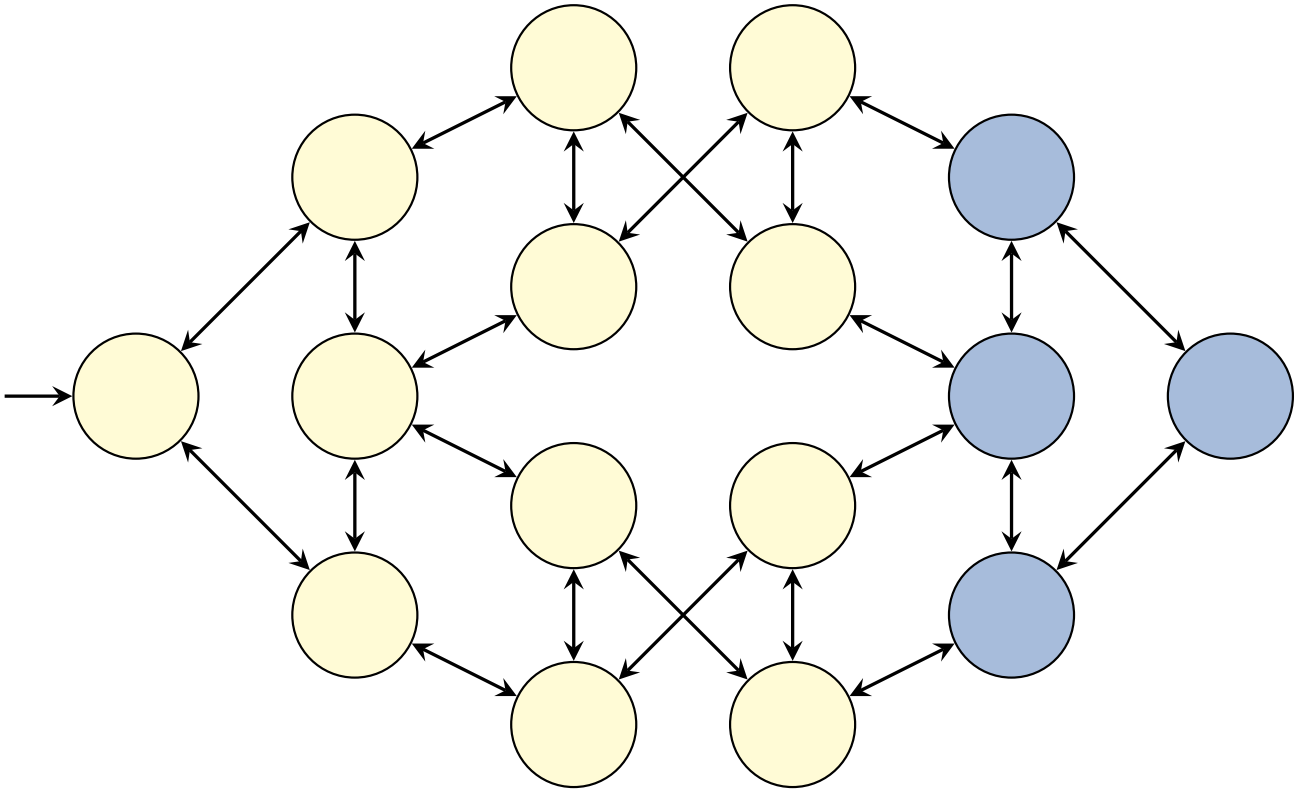
```
$ ls demo/ipc
agricola-opt18-strips
agricola-sat18-strips
airport
airport-adl
assembly
barman-mco14-strips
barman-opt11-strips
barman-opt14-strips
...
```

$\rightsquigarrow$ (most) benchmarks of planning competitions IPC since 1998

How Hard is Planning?

Classical Planning as State-Space Search

classical planning as state-space search:



$\rightsquigarrow$ much more on this later in the course

Is Planning Difficult?

Classical planning is computationally challenging:

- number of states grows exponentially with description size when using (propositional) logic-based representations
- provably hard (PSPACE-complete)

$\rightsquigarrow$ we prove this later in the course

problem sizes:

- Seven Bridges of Königsberg: 64 reachable states
- Rubik's Cube: $4.325 \cdot 10^{19}$ reachable states
  $\rightsquigarrow$ consider 2 billion/second $\rightsquigarrow$ 1 billion years
- standard benchmarks: some with $> 10^{200}$ reachable states

Summary

Summary

- planning = thinking before acting
- major subarea of Artificial Intelligence
- domain-independent planning = general problem solving
- classical planning = the "easy case"
  (deterministic, fully observable etc.)
- still hard enough!
  $\rightsquigarrow$ PSPACE-complete because of huge number of states
- often solved by state-space search
- number of states grows exponentially with input size

Planning and Optimization

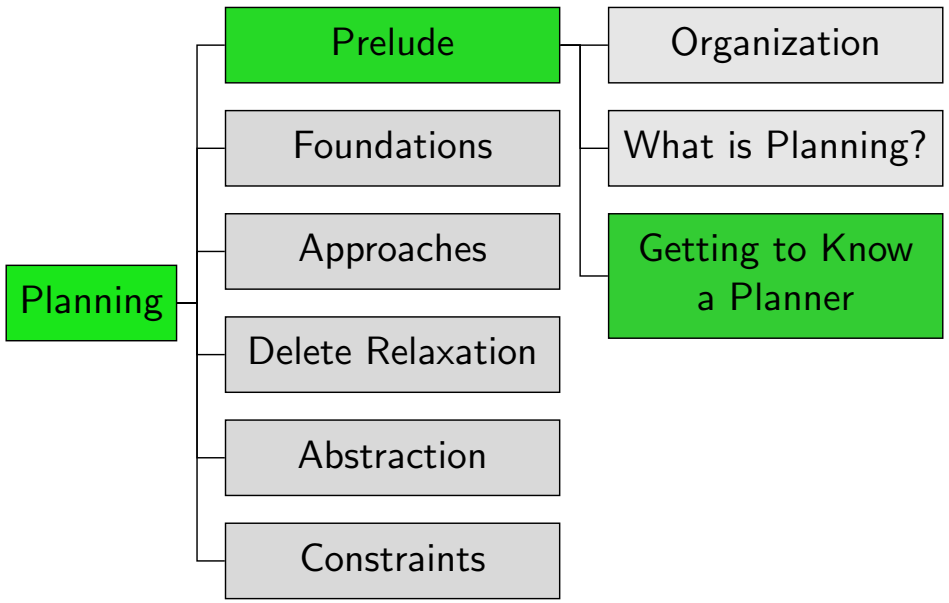
A3. Getting to Know a Planner

Malte Helmert and Gabriele Röger

Universität Basel

September 22, 2025

Content of the Course



Fast Downward and VAL

Getting to Know a Planner

We now play around a bit with a planner and its input:

- look at problem formulation
- run a planner (= planning system/planning algorithm)
- validate plans found by the planner

Planner: Fast Downward

Fast Downward

We use the Fast Downward planner in this course

- because we know it well (developed by our research group)
- because it implements many search algorithms and heuristics
- because it is the classical planner most commonly used as a basis for other planners

$\rightsquigarrow$ `https://www.fast-downward.org`

Validator: VAL

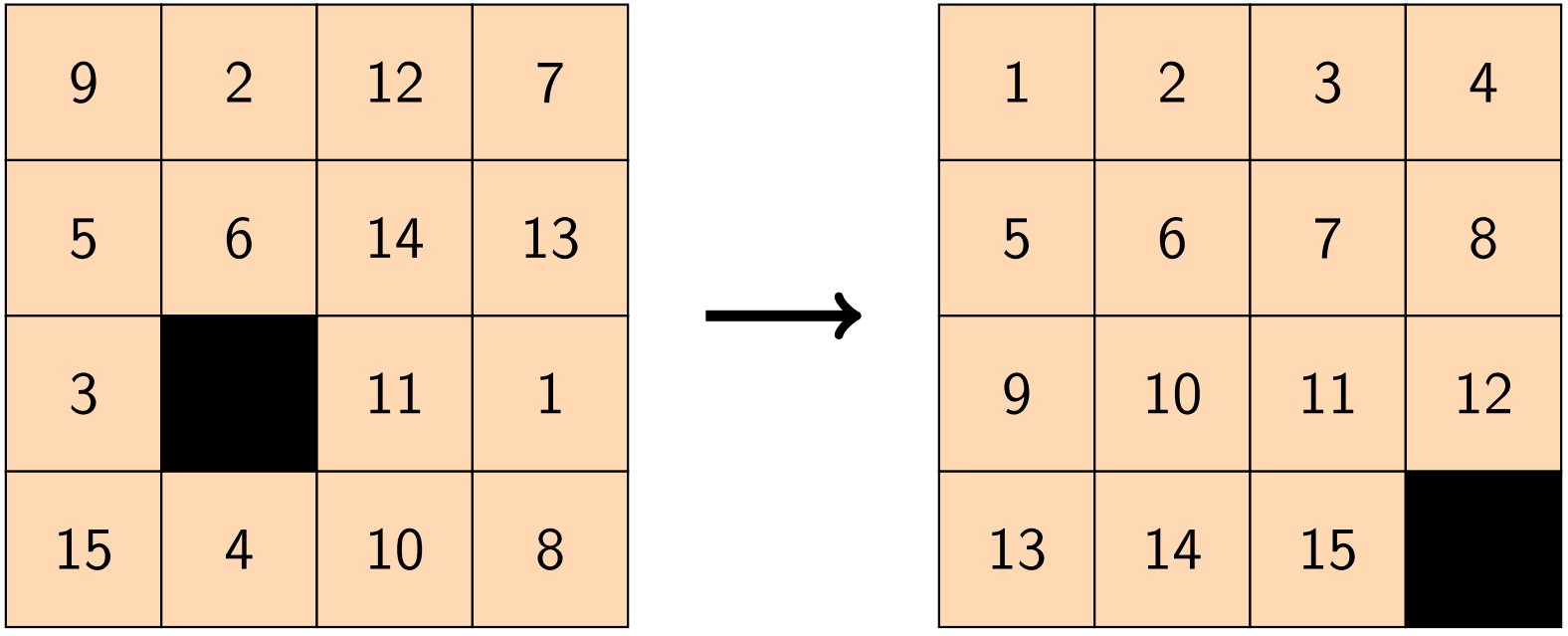
VAL

We use the VAL plan validation tool (Fox, Howey & Long)
to independently verify that the plans we generate are correct.

- very useful debugging tool
- `https://github.com/KCL-Planning/VAL`

15-Puzzle

Illustrating Example: 15-Puzzle



Solving the 15-Puzzle

Demo

```
$ cd demo
$ less tile/puzzle.pddl
$ less tile/puzzle01.pddl
$ ./fast-downward.py \
      tile/puzzle.pddl tile/puzzle01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
$ validate tile/puzzle.pddl tile/puzzle01.pddl \
      sas_plan
...
```

Variation: Weighted 15-Puzzle

Weighted 15-Puzzle:

- moving different tiles has different cost
- cost of moving tile x = number of prime factors of x

Demo

```
$ cd demo
$ meld tile/puzzle.pddl tile/weight.pddl
$ meld tile/puzzle01.pddl tile/weight01.pddl
$ ./fast-downward.py \
      tile/weight.pddl tile/weight01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Variation: Glued 15-Puzzle

Glued 15-Puzzle:

- some tiles are glued in place and cannot be moved

Demo

```
$ cd demo
$ meld tile/puzzle.pddl tile/glued.pddl
$ meld tile/puzzle01.pddl tile/glued01.pddl
$ ./fast-downward.py \
      tile/glued.pddl tile/glued01.pddl \
      --heuristic "h=cg()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Note: different heuristic used!

Variation: Cheating 15-Puzzle

Cheating 15-Puzzle:

- Can remove tiles from puzzle frame (creating more blanks) and reinsert tiles at any blank location.

Demo

```
$ cd demo
$ meld tile/puzzle.pddl tile/cheat.pddl
$ meld tile/puzzle01.pddl tile/cheat01.pddl
$ ./fast-downward.py \
      tile/cheat.pddl tile/cheat01.pddl \
      --heuristic "h=ff()" \
      --search "eager_greedy([h],preferred=[h])"
...
```

Summary

Summary

- We saw planning tasks modeled in the PDDL language.
- We ran the Fast Downward planner and VAL plan validator.
- We made some modifications to PDDL problem formulations and checked the impact on the planner.

Planning and Optimization

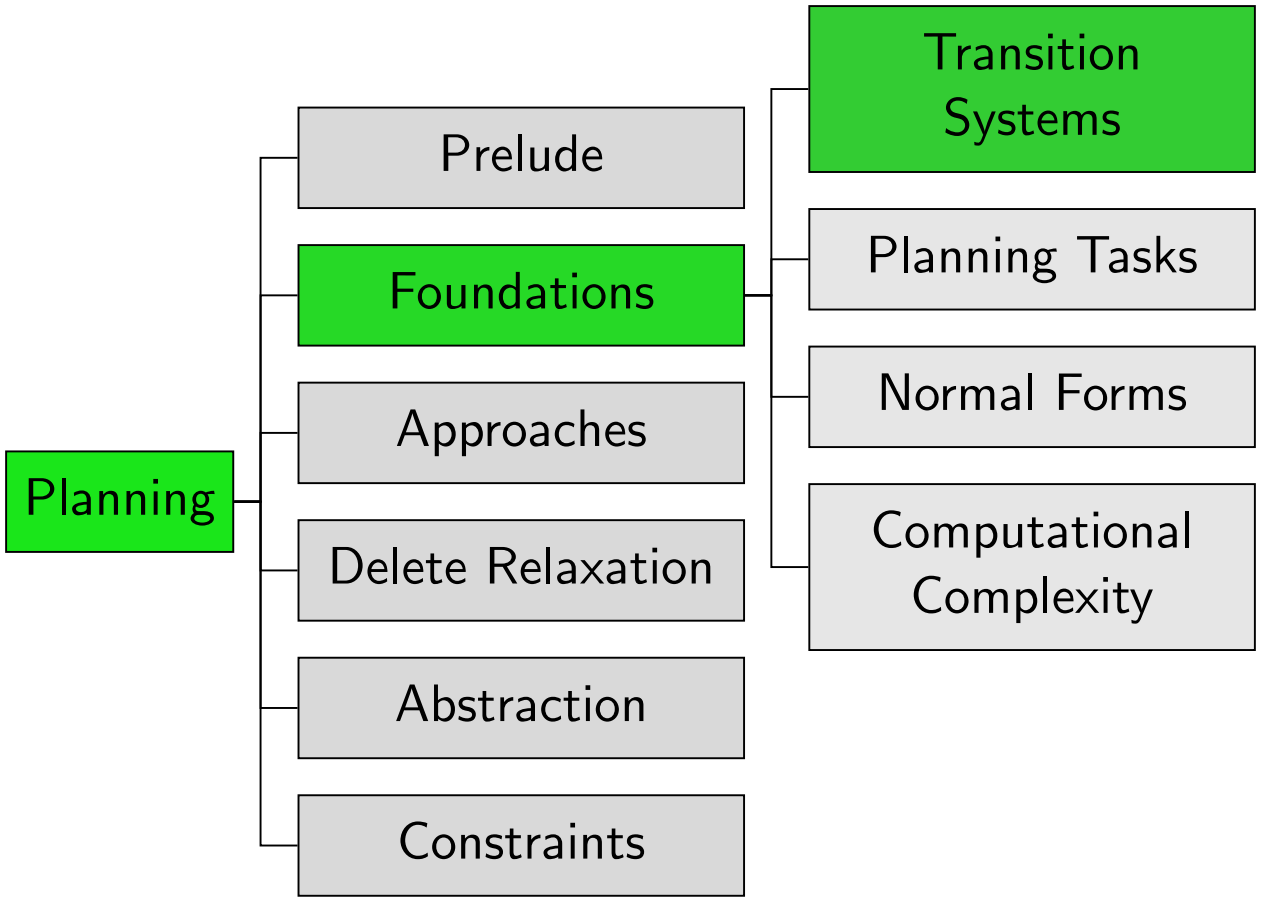
B1. Transition Systems and Propositional Logic

Malte Helmert and Gabriele Röger

Universität Basel

September 22, 2025

Content of the Course



Next Steps

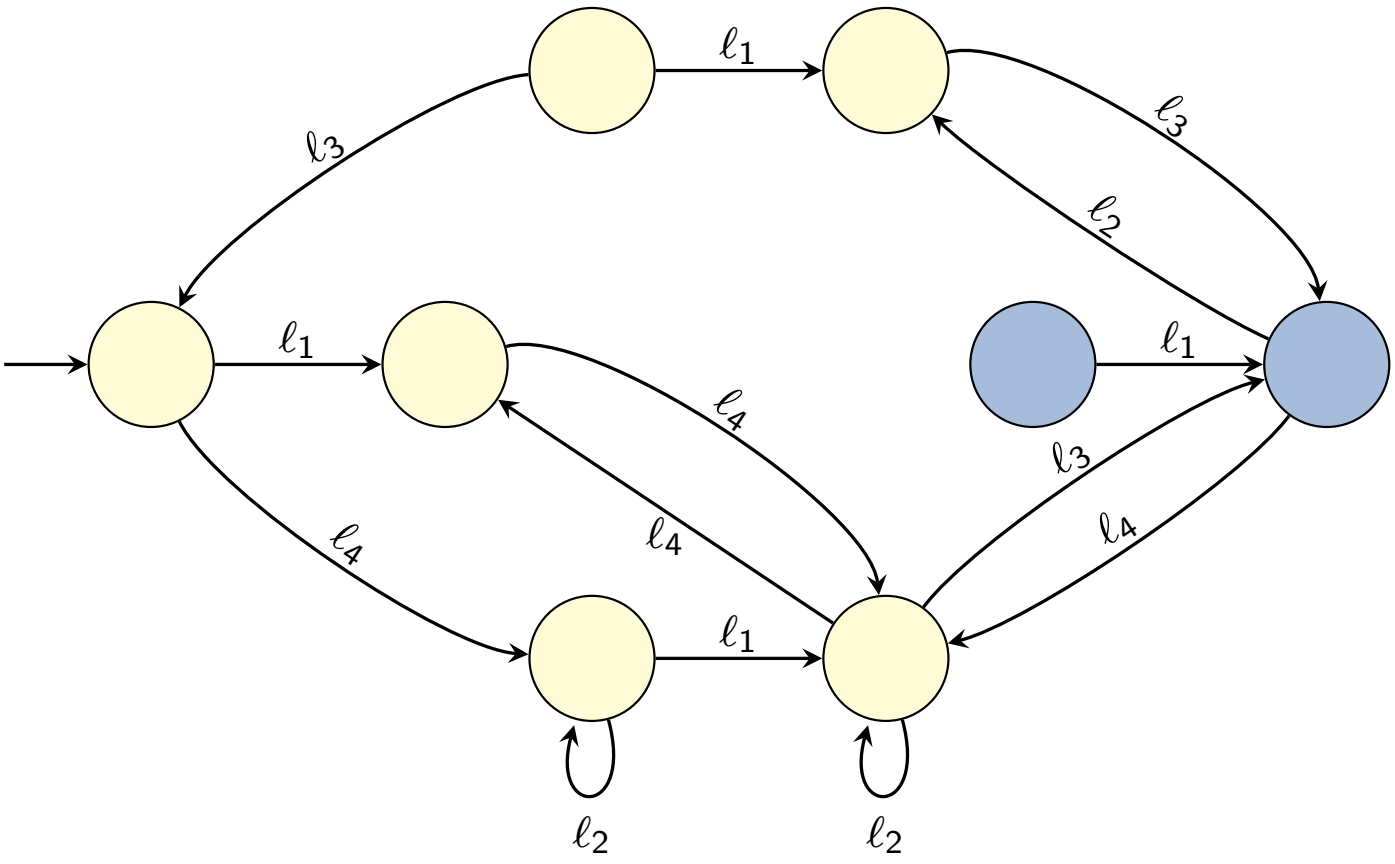
Our next steps are to formally define our problem:

- introduce a mathematical model for planning tasks: transition systems
  $\rightsquigarrow$ Chapter B1
- introduce compact representations for planning tasks suitable as input for planning algorithms
  $\rightsquigarrow$ Chapter B2

Transition Systems

Transition System Example

Transition systems are often depicted as directed arc-labeled graphs with decorations to indicate the initial state and goal states.



$$c(\ell_1) = 1,\ c(\ell_2) = 1,\ c(\ell_3) = 5,\ c(\ell_4) = 0$$

Transition Systems

Definition (Transition System)

A transition system is a 6-tuple $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$ where

- S is a finite set of states,
- L is a finite set of (transition) labels,
- $c : L \to \mathbb{R}_0^+$ is a label cost function,
- $T \subseteq S \times L \times S$ is the transition relation,
- $s_0 \in S$ is the initial state, and
- $S_\star \subseteq S$ is the set of goal states.

We say that $\mathcal{T}$ has the transition $\langle s, \ell, s' \rangle$ if $\langle s, \ell, s' \rangle \in T$.

We also write this as $s \xrightarrow{\ell} s'$, or $s \to s'$ when not interested in ℓ.

Note: Transition systems are also called state spaces.

Deterministic Transition Systems

Definition (Deterministic Transition System)

A transition system is called deterministic if for all states s and all labels ℓ, there is at most one state s' with $s \xrightarrow{\ell} s'$.

Example: previously shown transition system

Transition System Terminology (1)

We use common terminology from graph theory:

- s' successor of s if $s \to s'$
- s predecessor of s' if $s \to s'$

Transition System Terminology (2)

We use common terminology from graph theory:

- s' reachable from s if there exists a sequence of transitions $s^0 \xrightarrow{\ell_1} s^1, \ldots, s^{n-1} \xrightarrow{\ell_n} s^n$ s.t. $s^0 = s$ and $s^n = s'$
  - Note: $n = 0$ possible; then $s = s'$
  - $s^0, \ldots, s^n$ is called (state) path from s to s'
  - $\ell_1, \ldots, \ell_n$ is called (label) path from s to s'
  - $s^0 \xrightarrow{\ell_1} s^1, \ldots, s^{n-1} \xrightarrow{\ell_n} s^n$ is called trace from s to s'
  - length of path/trace is n
  - cost of label path/trace is $\sum_{i=1}^{n} c(\ell_i)$

Transition System Terminology (3)

We use common terminology from graph theory:

- s' reachable (without reference state) means reachable from initial state s_0
- solution or goal path from s: path from s to some $s' \in S_\star$
  - if s is omitted, $s = s_0$ is implied
- transition system solvable if a goal path from s_0 exists

Example: Blocks World

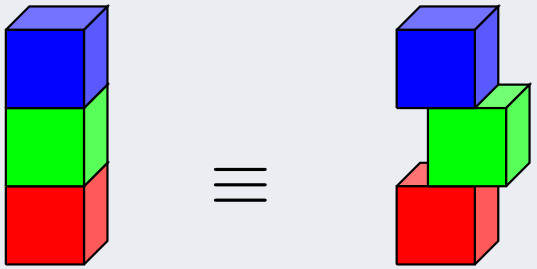
Running Example: Blocks World

- Throughout the course, we occasionally use the blocks world domain as an example.
- In the blocks world, a number of different blocks are arranged on a table.
- Our job is to rearrange them according to a given goal.

Blocks World Rules (1)

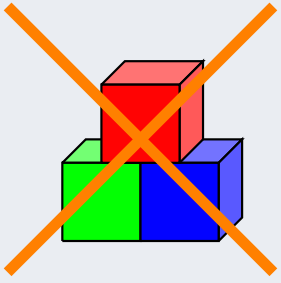
Location on the table does not matter.

Location on a block does not matter.

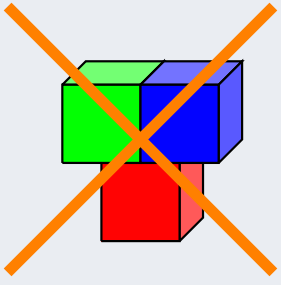


Blocks World Rules (2)

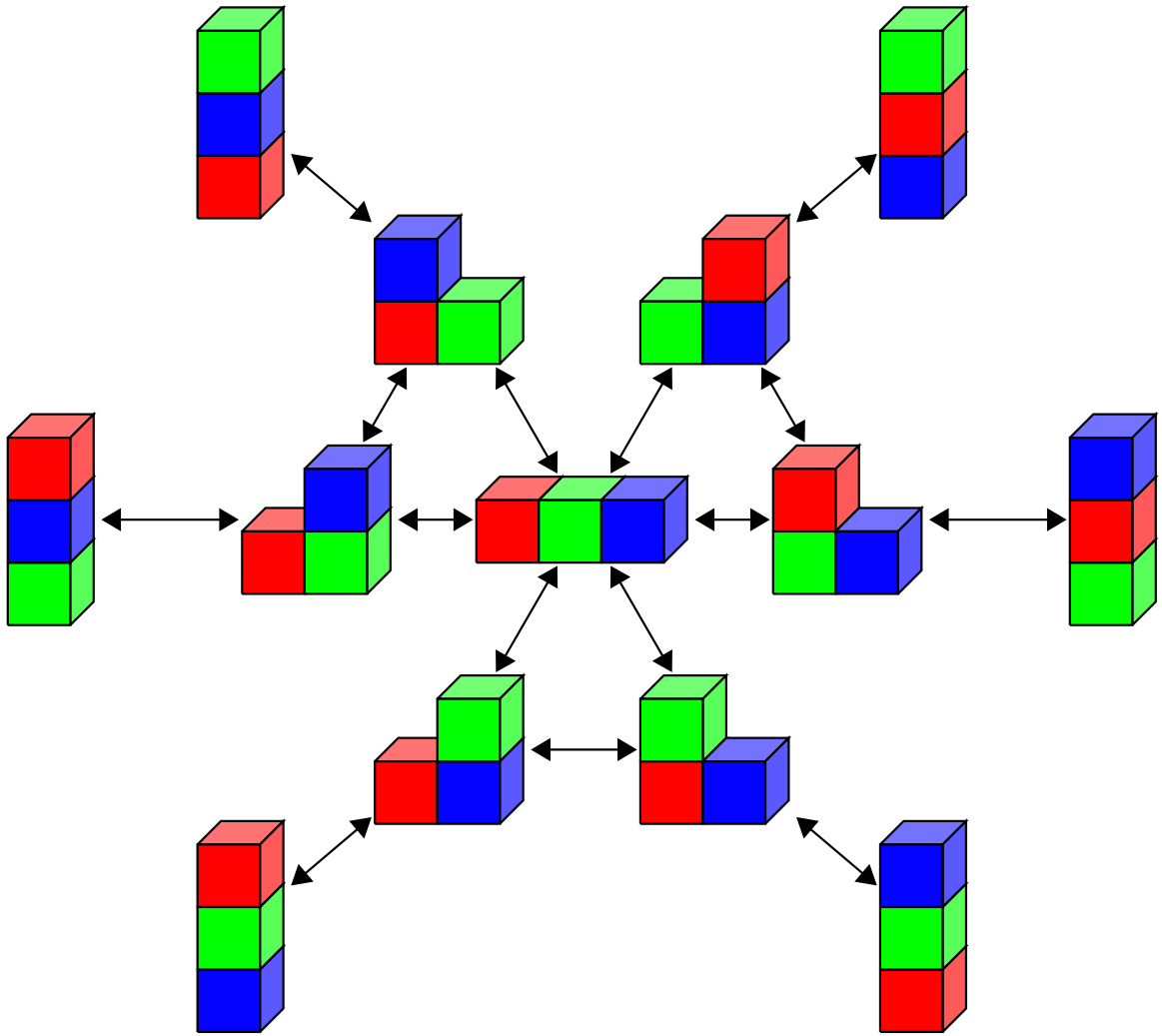
At most one block may be below a block.



At most one block may be on top of a block.



Blocks World Transition System for Three Blocks



Labels omitted for clarity. All label costs are 1. Initial/goal states not marked.

Blocks World Computational Properties

blocks	states	blocks	states
1	1	10	58941091
2	3	11	824073141
3	13	12	12470162233
4	73	13	202976401213
5	501	14	35350175244O3
6	4051	15	65573803186921
7	37633	16	1290434218669921
8	394353	17	26846616451246353
9	4596553	18	588633468315403843

- Finding solutions is possible in linear time
  in the number of blocks: move everything onto the table, then construct the goal configuration.
- Finding a shortest solution is NP-complete
  given a compact description of the problem.

The Need for Compact Descriptions

- We see from the blocks world example that transition systems are often far too large to be directly used as inputs to planning algorithms.
- We therefore need compact descriptions of transition systems.
- For this purpose, we will use propositional logic, which allows expressing information about 2^n states as logical formulas over n state variables.

Reminder: Propositional Logic

More on Propositional Logic

Need to Catch Up?

- This section is a reminder. We assume you are already well familiar with propositional logic.
- If this is not the case, we recommend Chapters D1–D4 of the Discrete Mathematics in Computer Science course: `https://dmi.unibas.ch/en/studies/computer-science/course-offer-hs24/lecture-discrete-mathematics-in-computer-science/`
  - Videos for these chapters are available on request.

Syntax of Propositional Logic

Definition (Logical Formula)

Let A be a set of atomic propositions.

The logical formulas over A are constructed by finite application of the following rules:

- $\top$ and $\bot$ are logical formulas (truth and falsity).
- For all $a \in A$, a is a logical formula (atom).
- If φ is a logical formula, then so is $\neg\varphi$ (negation).
- If φ and ψ are logical formulas, then so are $(\varphi \vee \psi)$ (disjunction) and $(\varphi \wedge \psi)$ (conjunction).

Syntactical Conventions for Propositional Logic

Abbreviations:

- $(\varphi \rightarrow \psi)$ is short for $(\neg\varphi \vee \psi)$ (implication)
- $(\varphi \leftrightarrow \psi)$ is short for $((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi))$ (equijunction)
- parentheses omitted when not necessary:
  - $(\neg)$ binds more tightly than binary connectives
  - $(\wedge)$ binds more tightly than $(\vee)$, which binds more tightly than $(\rightarrow)$, which binds more tightly than $(\leftrightarrow)$

Semantics of Propositional Logic

Definition (Interpretation, Model)

An interpretation of propositions A is a function $I : A \to \{\mathbf{T}, \mathbf{F}\}$.

Define the notation $I \models \varphi$ (I satisfies φ; I is a model of φ; φ is true under I) for interpretations I and formulas φ by

- $I \models \top$
- $I \not\models \bot$
- $I \models a$ iff $I(a) = \mathbf{T}$ (for all $a \in A$)
- $I \models \neg\varphi$ iff $I \not\models \varphi$
- $I \models (\varphi \vee \psi)$ iff ($I \models \varphi$ or $I \models \psi$)
- $I \models (\varphi \wedge \psi)$ iff ($I \models \varphi$ and $I \models \psi$)

Note: Interpretations are also called valuations or truth assignments.

Propositional Logic Terminology (1)

- A logical formula φ is satisfiable
  if there is at least one interpretation I such that $I \models \varphi$.
- Otherwise it is unsatisfiable.
- A logical formula φ is valid or a tautology
  if $I \models \varphi$ for all interpretations I.
- A logical formula ψ is a logical consequence
  of a logical formula φ, written $\varphi \models \psi$,
  if $I \models \psi$ for all interpretations I with $I \models \varphi$.
- Two logical formulas φ and ψ are logically equivalent,
  written $\varphi \equiv \psi$, if $\varphi \models \psi$ and $\psi \models \varphi$.

Question: How to phrase these in terms of models?

Propositional Logic Terminology (2)

- A logical formula that is a proposition a or a negated proposition $\neg a$ for some atomic proposition $a \in A$ is a literal.
- A formula that is a disjunction of literals is a clause.
  This includes unit clauses ℓ consisting of a single literal and the empty clause $\bot$ consisting of zero literals.
- A formula that is a conjunction of literals is a monomial.
  This includes unit monomials ℓ consisting of a single literal and the empty monomial $\top$ consisting of zero literals.

Normal forms:

- negation normal form (NNF)
- conjunctive normal form (CNF)
- disjunctive normal form (DNF)

Summary

Summary

- Transition systems are (typically huge) directed graphs that encode how the state of the world can change.
- Propositional logic allows us to compactly describe complex information about large sets of interpretations as logical formulas.

Planning and Optimization

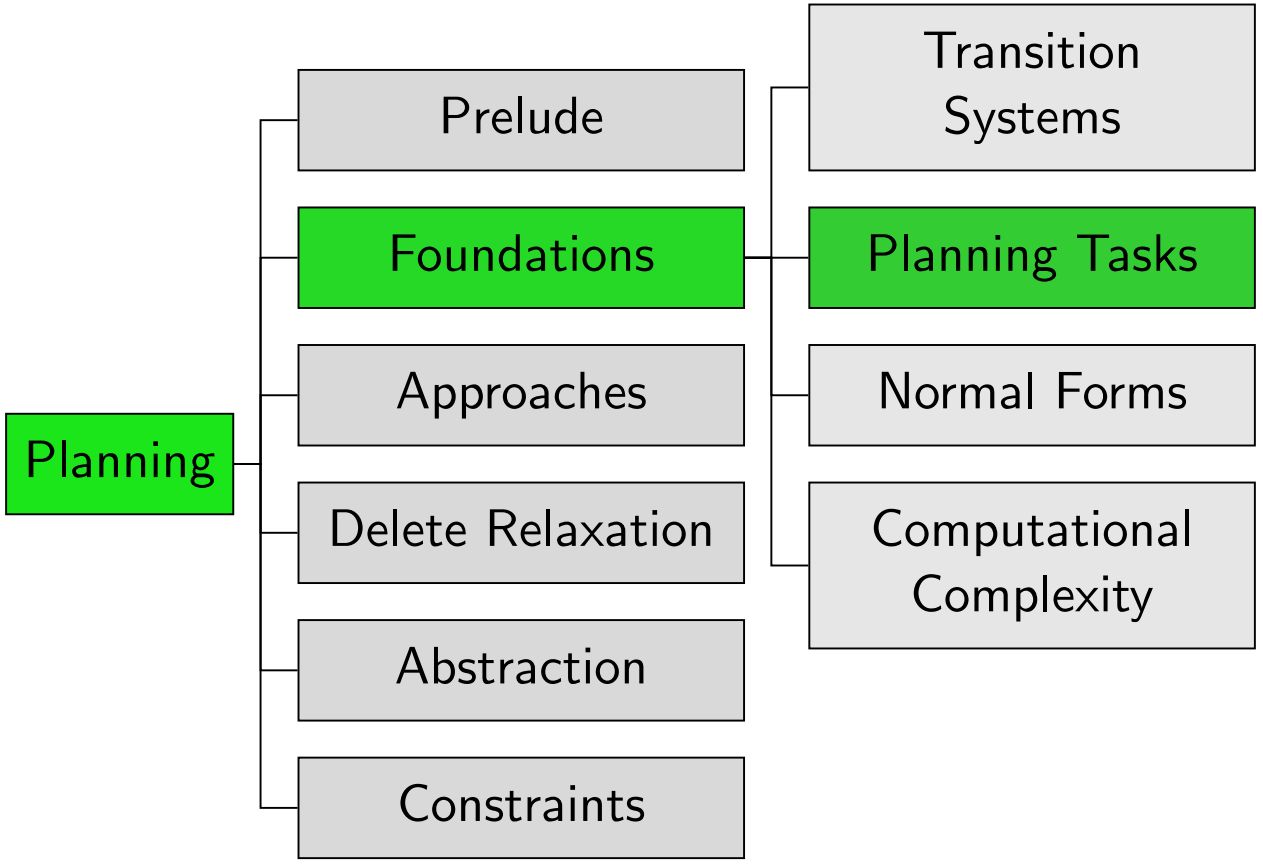
B2. Introduction to Planning Tasks

Malte Helmert and Gabriele Röger

Universität Basel

September 24, 2025

Content of the Course



Introduction

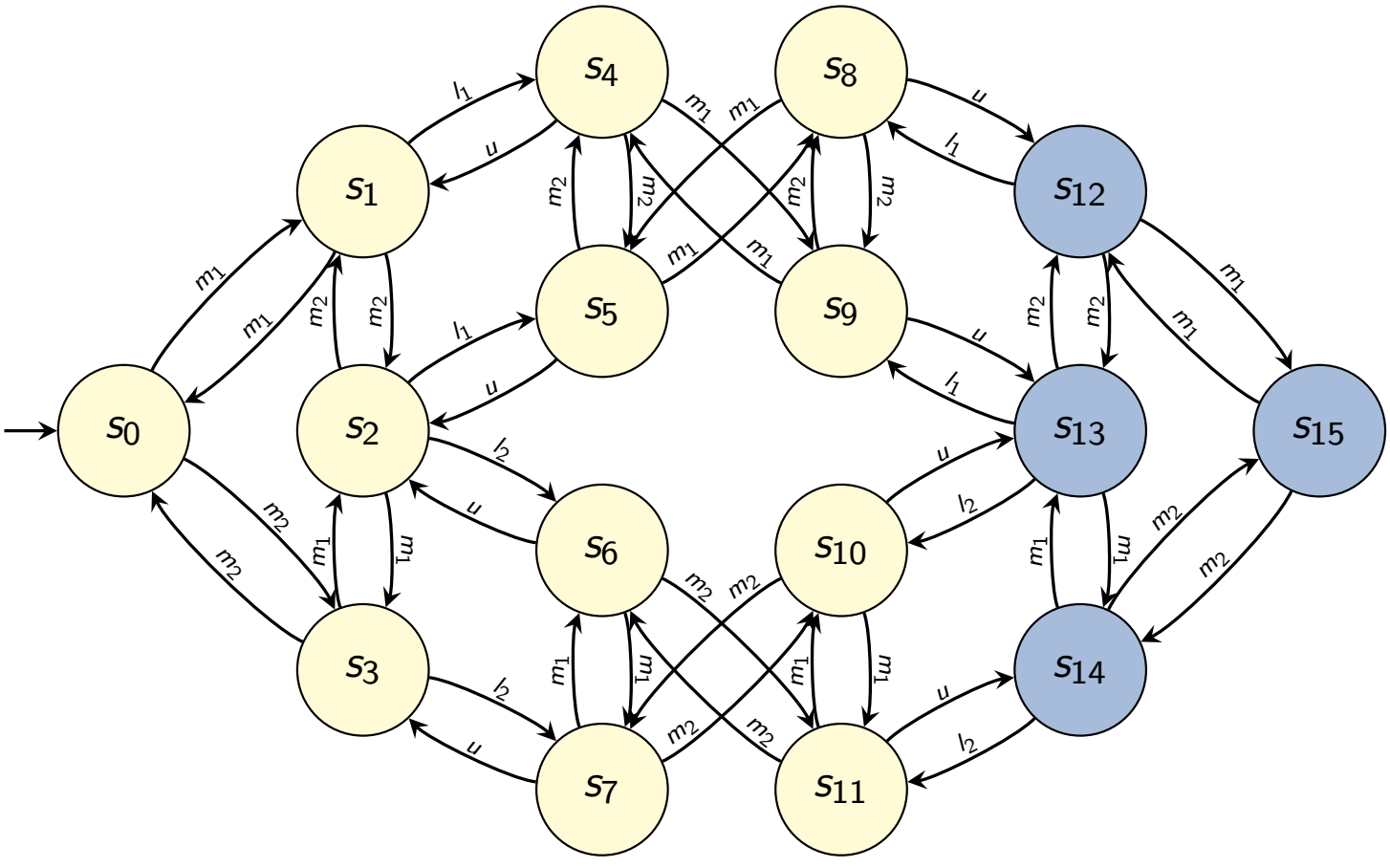
The State Explosion Problem

- We saw in blocks world:
  n blocks $\rightsquigarrow$ number of states exponential in n
- same is true everywhere we look
- known as the state explosion problem

To represent transitions systems compactly,
need to tame these exponentially growing aspects:

- states
- goal states
- transitions

Running Example: Transition System



$c(m_1) = 5,\ c(m_2) = 5,\ c(l_1) = 1,\ c(l_2) = 1,\ c(u) = 1$

State Variables

Compact Descriptions of Transition Systems

How to specify huge transition systems
without enumerating the states?

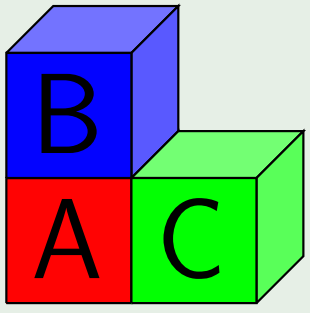
- represent different aspects of the world
  in terms of different (propositional) state variables
- individual state variables are atomic propositions
  $\rightsquigarrow$ a state is an interpretation of state variables
- n state variables induce 2^n states
  $\rightsquigarrow$ exponentially more compact than "flat" representations

Example: n^2 variables suffice for blocks world with n blocks

Blocks World State with Propositional Variables

Example

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



$\rightsquigarrow$ 9 variables for 3 blocks

Propositional State Variables

Definition (Propositional State Variable)

A propositional state variable is a symbol X.

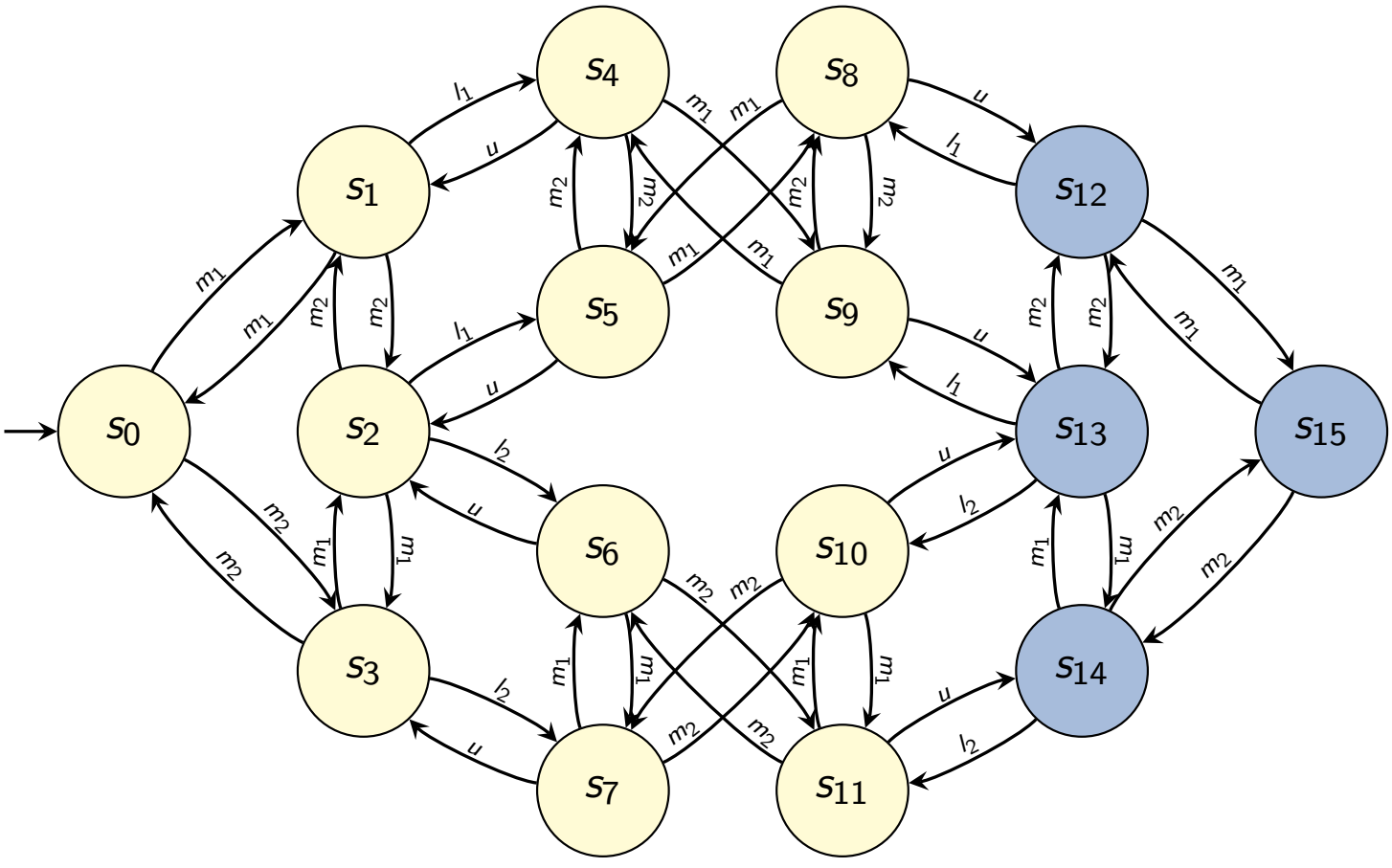
Let V be a finite set of propositional state variables.

A state s over V is an interpretation of V, i.e.,
a truth assignment $s : V \to \{\mathbf{T}, \mathbf{F}\}$.

Running Example: Compact State Descriptions

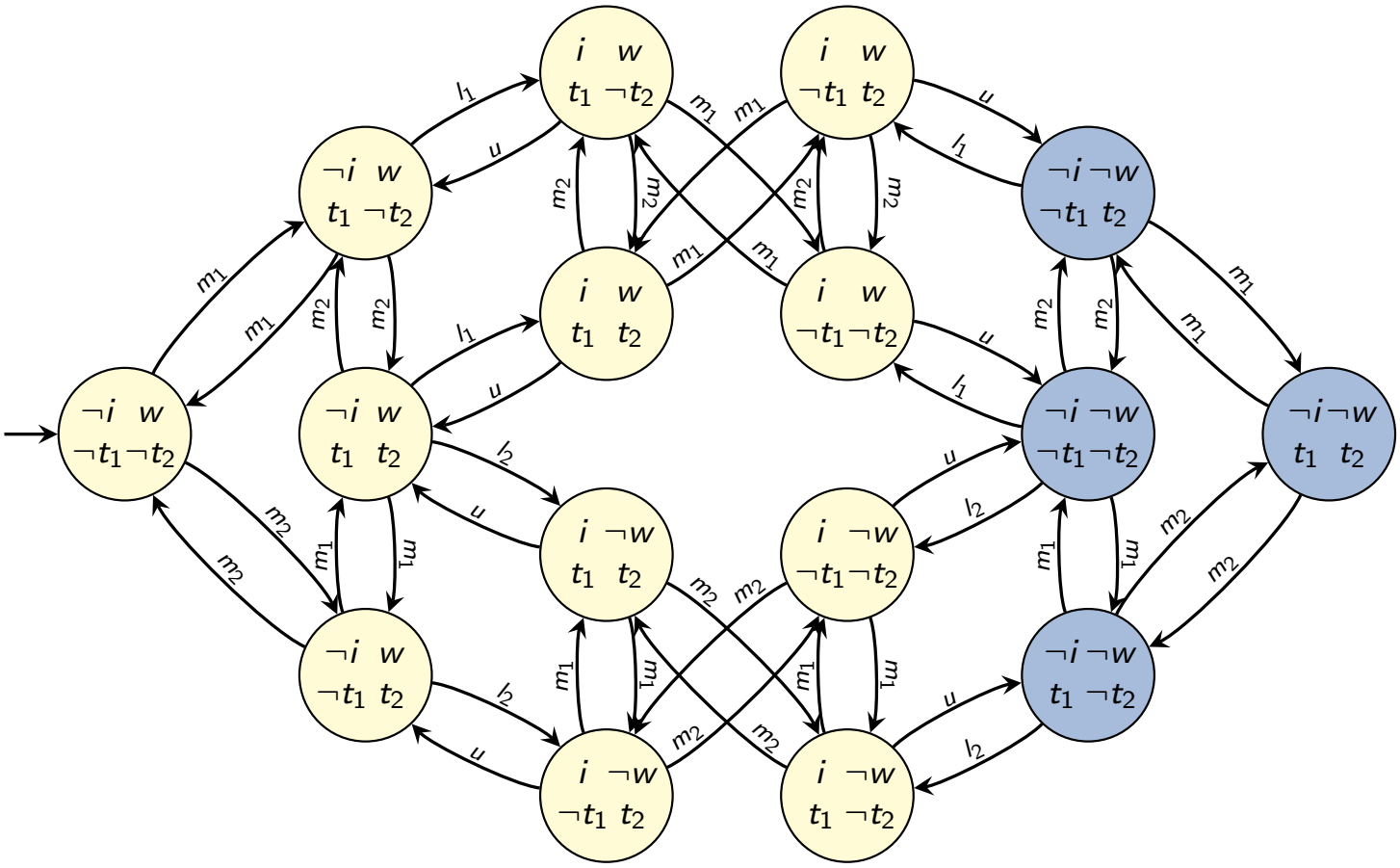
- In the running example, we describe 16 states with 4 propositional state variables ($2^4 = 16$).

Running Example: Opaque States



Running Example: Using State Variables

state variables $V = \{i, w, t_1, t_2\}$



states shown by true literals

example: $\{i \mapsto \mathbf{T}, w \mapsto \mathbf{F}, t_1 \mapsto \mathbf{T}, t_2 \mapsto \mathbf{F}\} \rightsquigarrow i\ \neg w\ t_1\ \neg t_2$

Running Example: Intuition

Intuition: delivery task with 2 trucks, 1 package, locations L and R

transition labels:

- m_1/m_2: move first/second truck
- l_1/l_2: load package into first/second truck
- u: unload package from a truck

state variables:

- t_1 true if first truck is at location L (else at R)
- t_2 true if second truck is at location L (else at R)
- i true if package is inside a truck
- w encodes where exactly the package is:
  - if i is true, w true if package in first truck
  - if i is false, w true if package at location L

State Formulas

Representing Sets of States

How do we compactly represent sets of states,
for example the set of goal states?

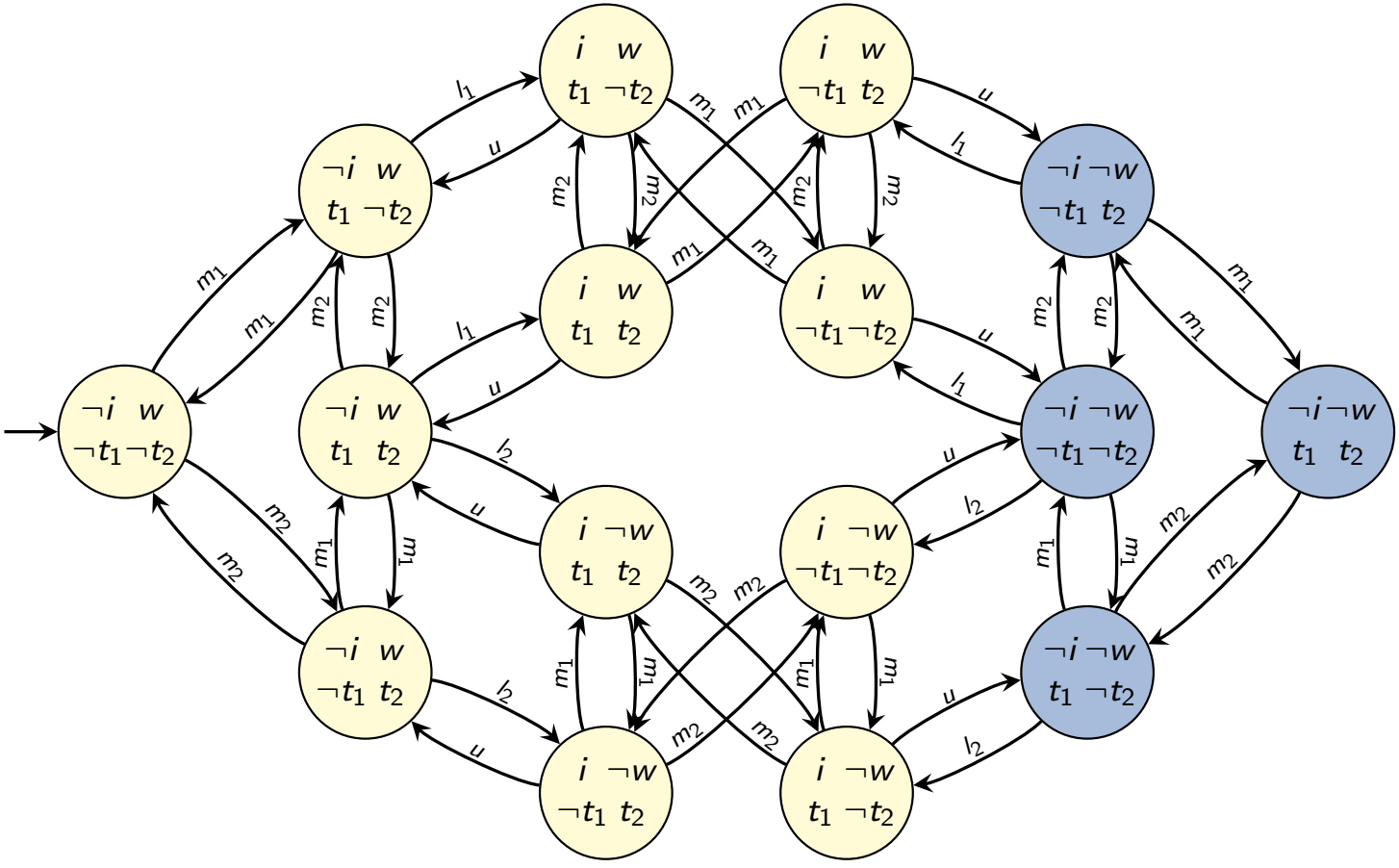
Idea: formula φ over the state variables represents the models of φ.

Definition (State Formula)

Let V be a finite set of propositional state variables.

A formula over V is a propositional logic formula using V as the set of atomic propositions.

Running Example: Representing Goal States



goal formula $\gamma = \neg i \wedge \neg w$ represents goal states $S_\star$

Operators and Effects

Operators Representing Transitions

How do we compactly represent transitions?

- most complex aspect of a planning task
- central concept: operators

Idea: one operator o for each transition label ℓ, describing

- in which states s a transition $s \xrightarrow{\ell} s'$ exists (precondition)
- how state s' differs from state s (effect)
- what the cost of ℓ is

Syntax of Operators

Definition (Operator)

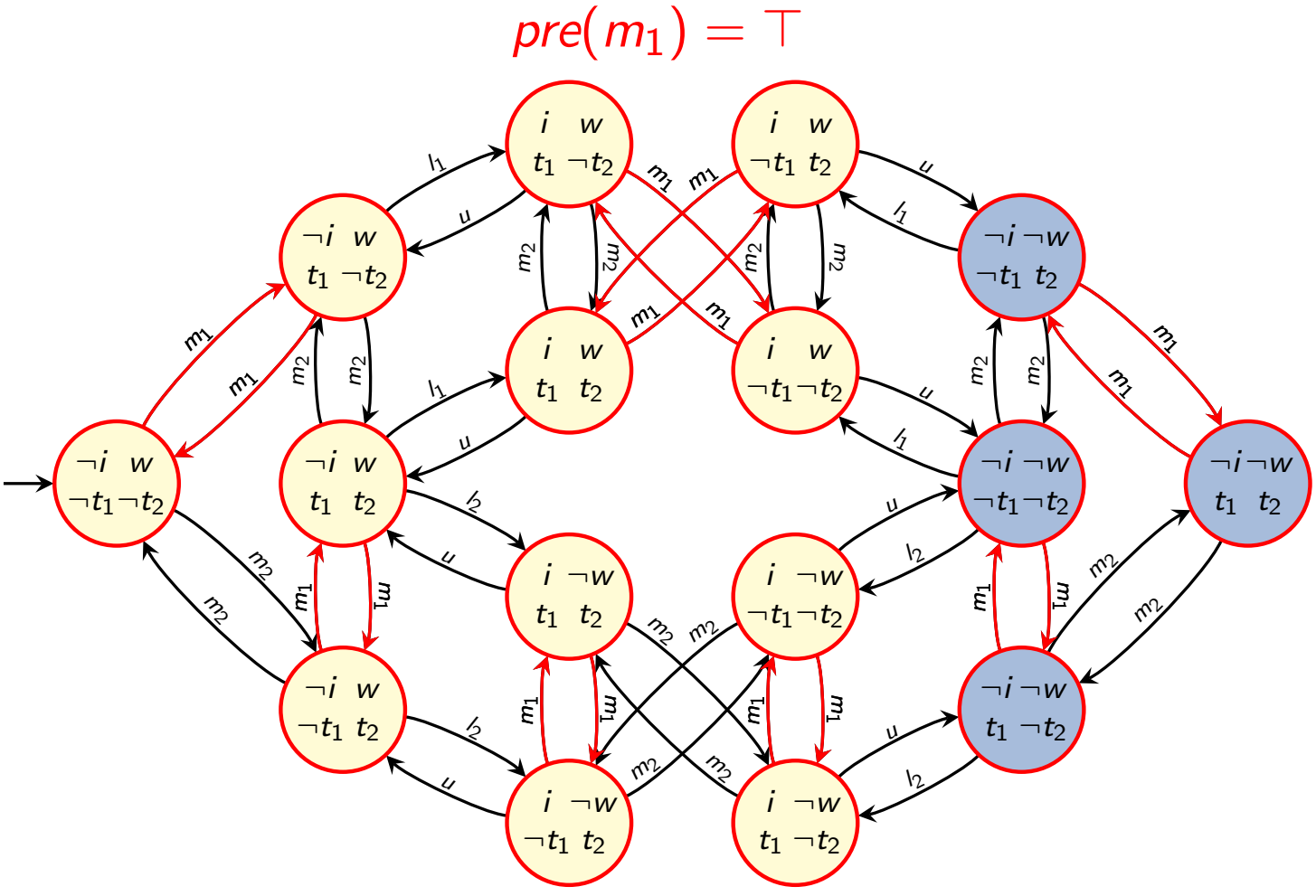
An operator o over state variables V is an object with three properties:

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

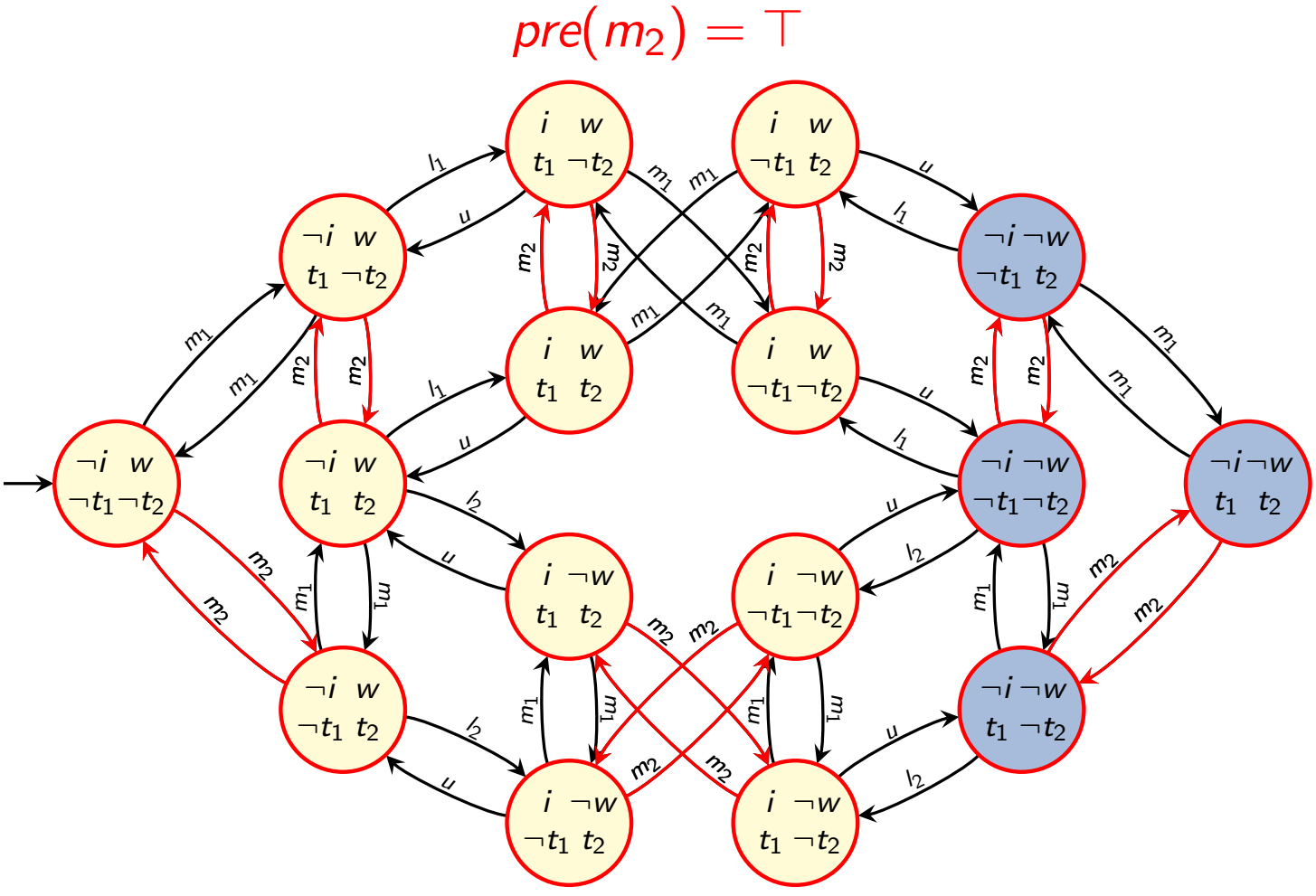
Notes:

- Operators are also called actions.
- Operators are often written as triples $\langle pre(o), eff(o), cost(o) \rangle$.
- This can be abbreviated to pairs $\langle pre(o), eff(o) \rangle$ when the cost of the operator is irrelevant.

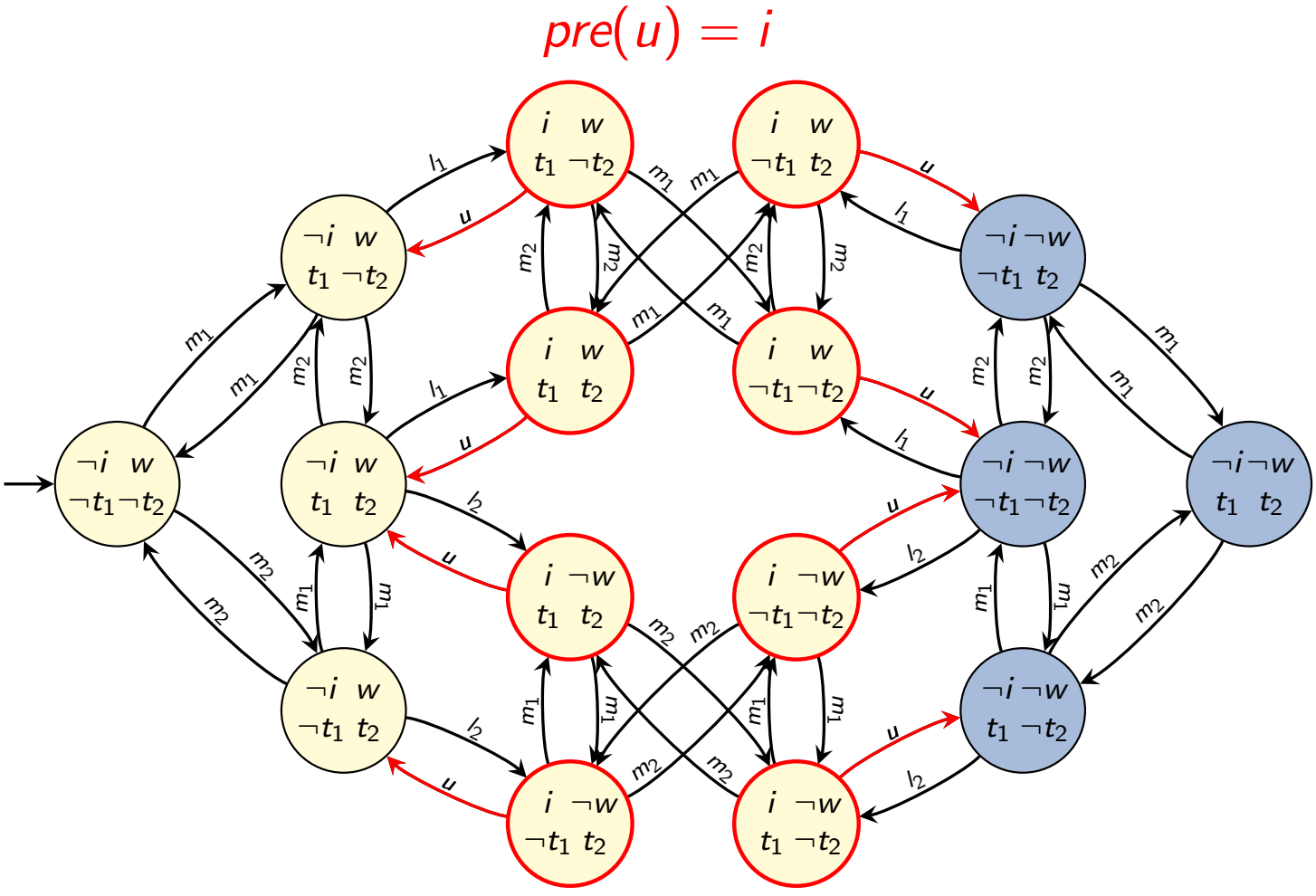
Running Example: Operator Preconditions



Running Example: Operator Preconditions

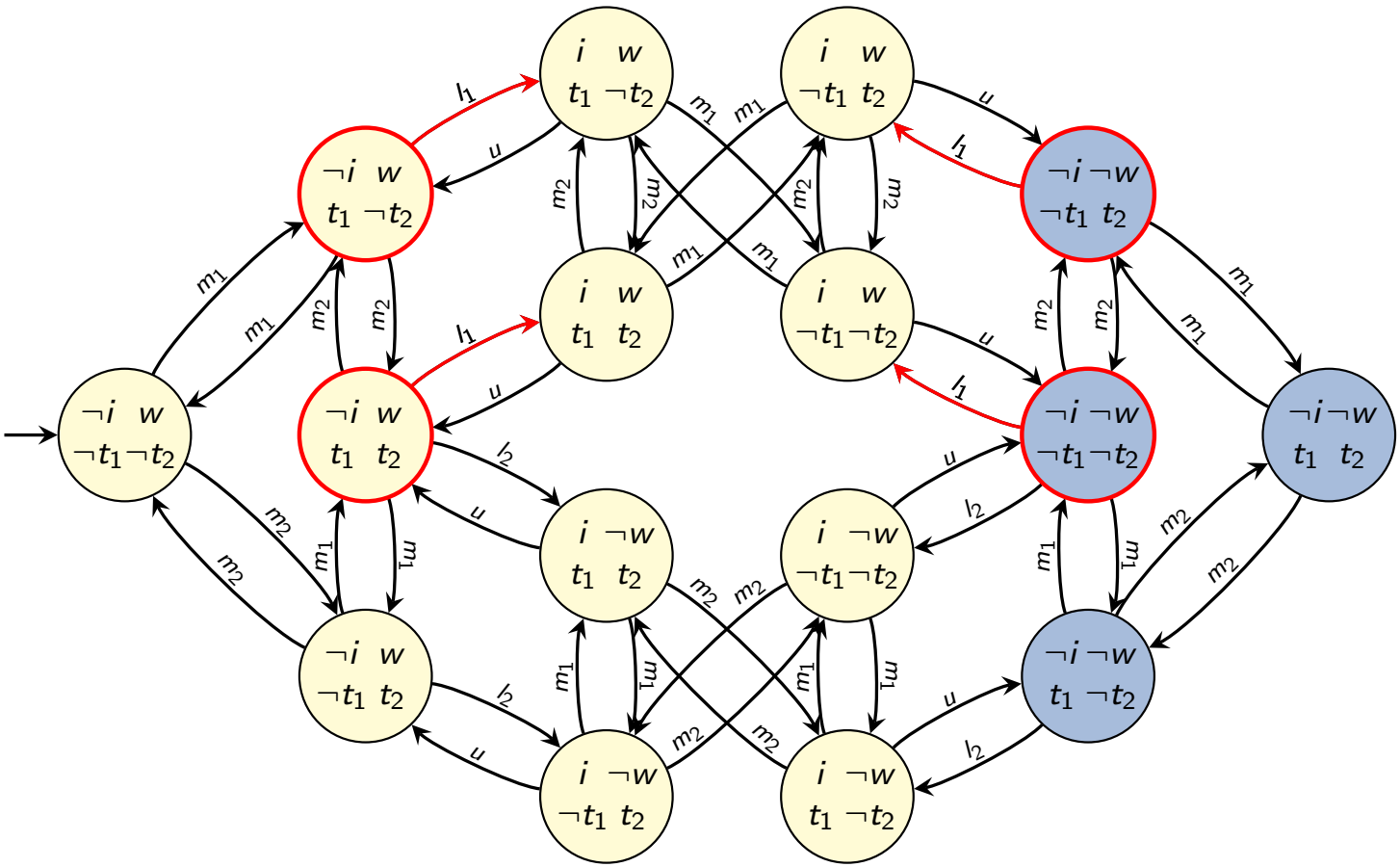


Running Example: Operator Preconditions



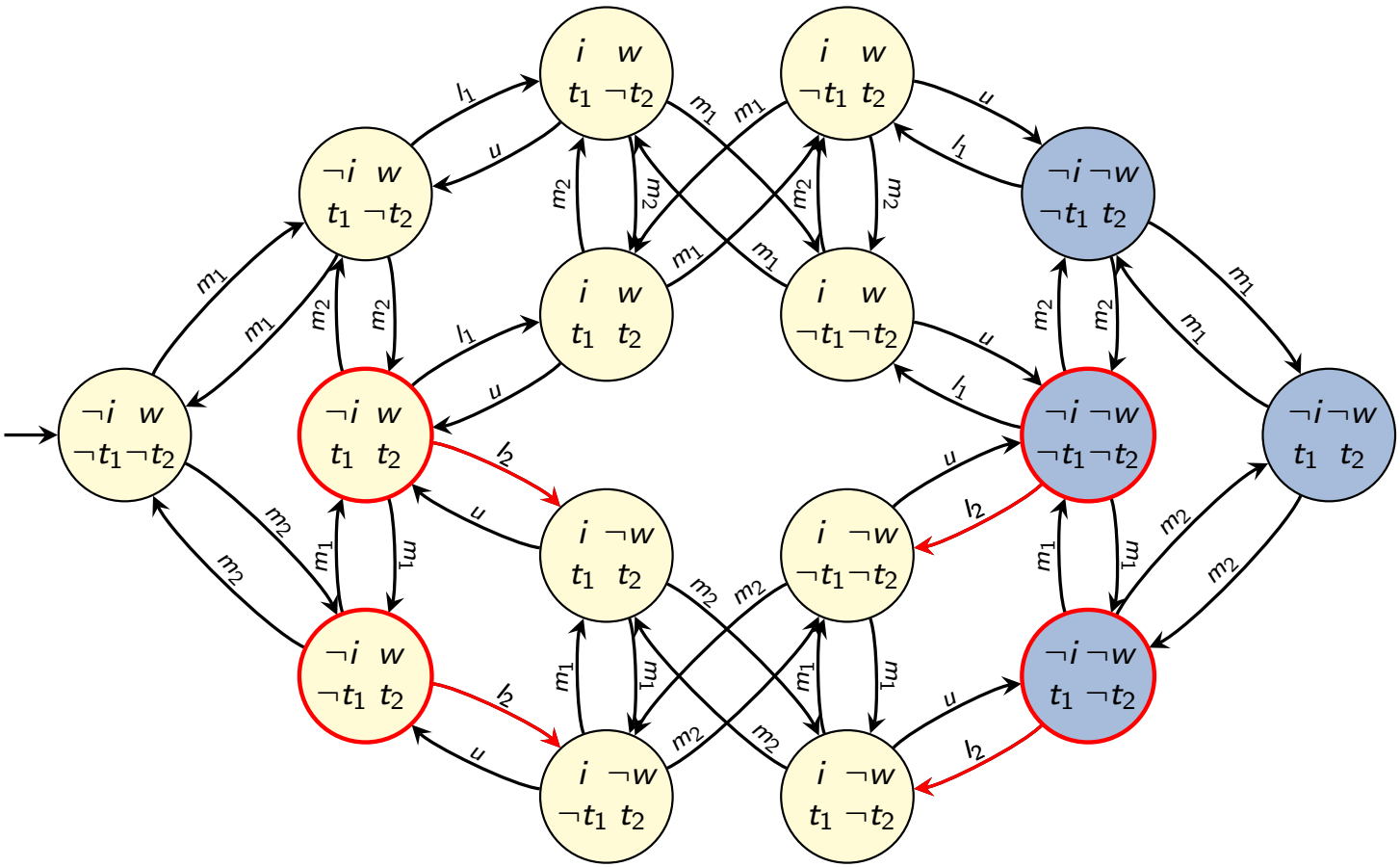
Running Example: Operator Preconditions

$$pre(l_1) = \neg i \wedge (w \leftrightarrow t_1)$$



Running Example: Operator Preconditions

$$pre(l_2) = \neg i \wedge (w \leftrightarrow t_2)$$



Syntax of Effects

Definition (Effect)

Effects over propositional state variables V are inductively defined as follows:

- $\top$ is an effect (empty effect).
- If $v \in V$ is a propositional state variable, then v and $\neg v$ are effects (atomic effect).
- If e and e' are effects, then $(e \wedge e')$ is an effect (conjunctive effect).
- If χ is a formula over V and e is an effect, then $(\chi \rhd e)$ is an effect (conditional effect).

We may omit parentheses when this does not cause ambiguity.

Example: we will later see that $((e \wedge e') \wedge e'')$ behaves identically to $(e \wedge (e' \wedge e''))$ and will write this as $e \wedge e' \wedge e''$.

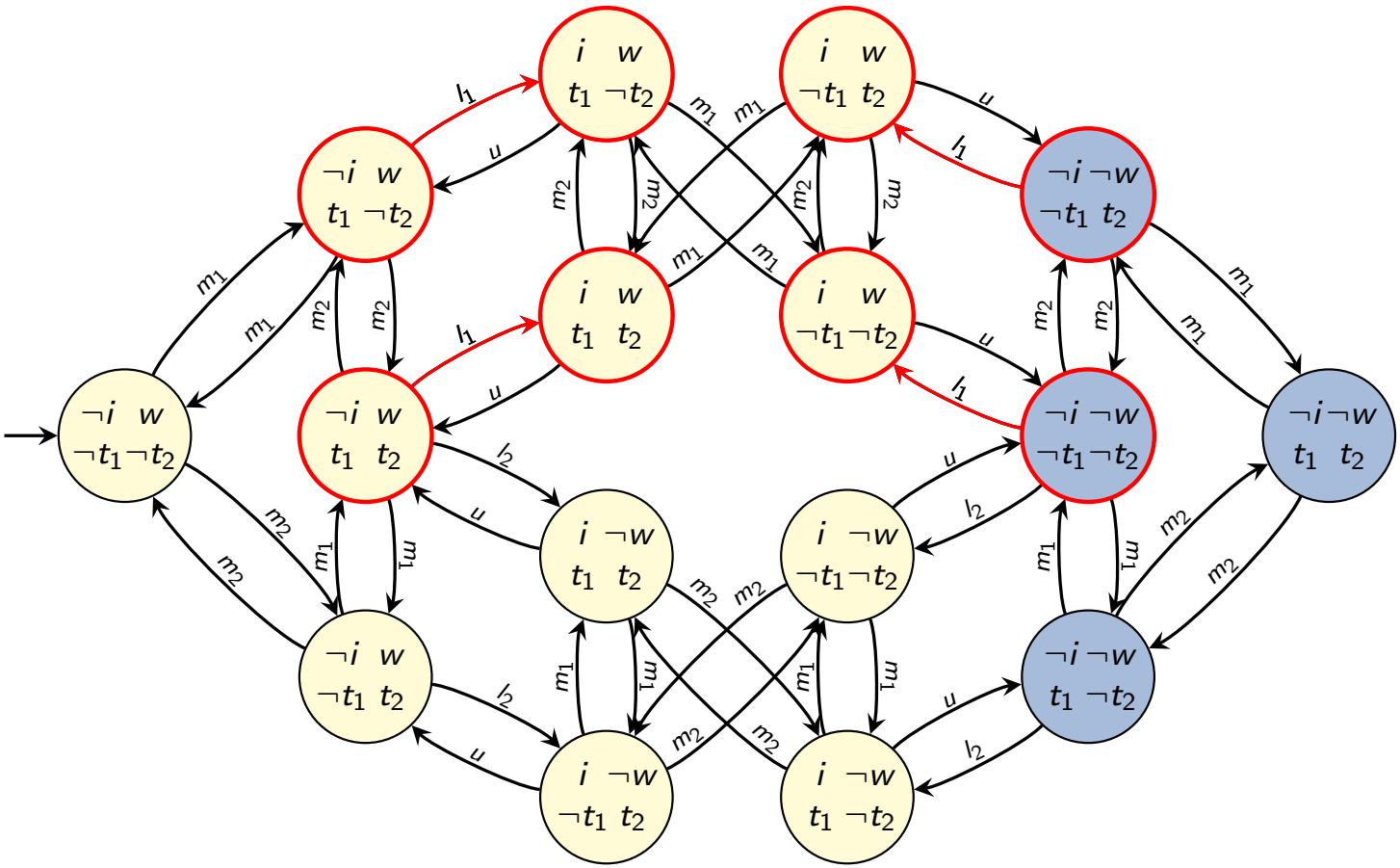
Effects: Intuition

Intuition for effects:

- The empty effect $\top$ changes nothing.
- Atomic effects can be understood as assignments that update the value of a state variable.
  - v means "$v := \mathbf{T}$"
  - $\neg v$ means "$v := \mathbf{F}$"
- A conjunctive effect $e = (e' \wedge e'')$ means that both subeffects e and e' take place simultaneously.
- A conditional effect $e = (\chi \rhd e')$ means that subeffect e' takes place iff χ is true in the state where e takes place.

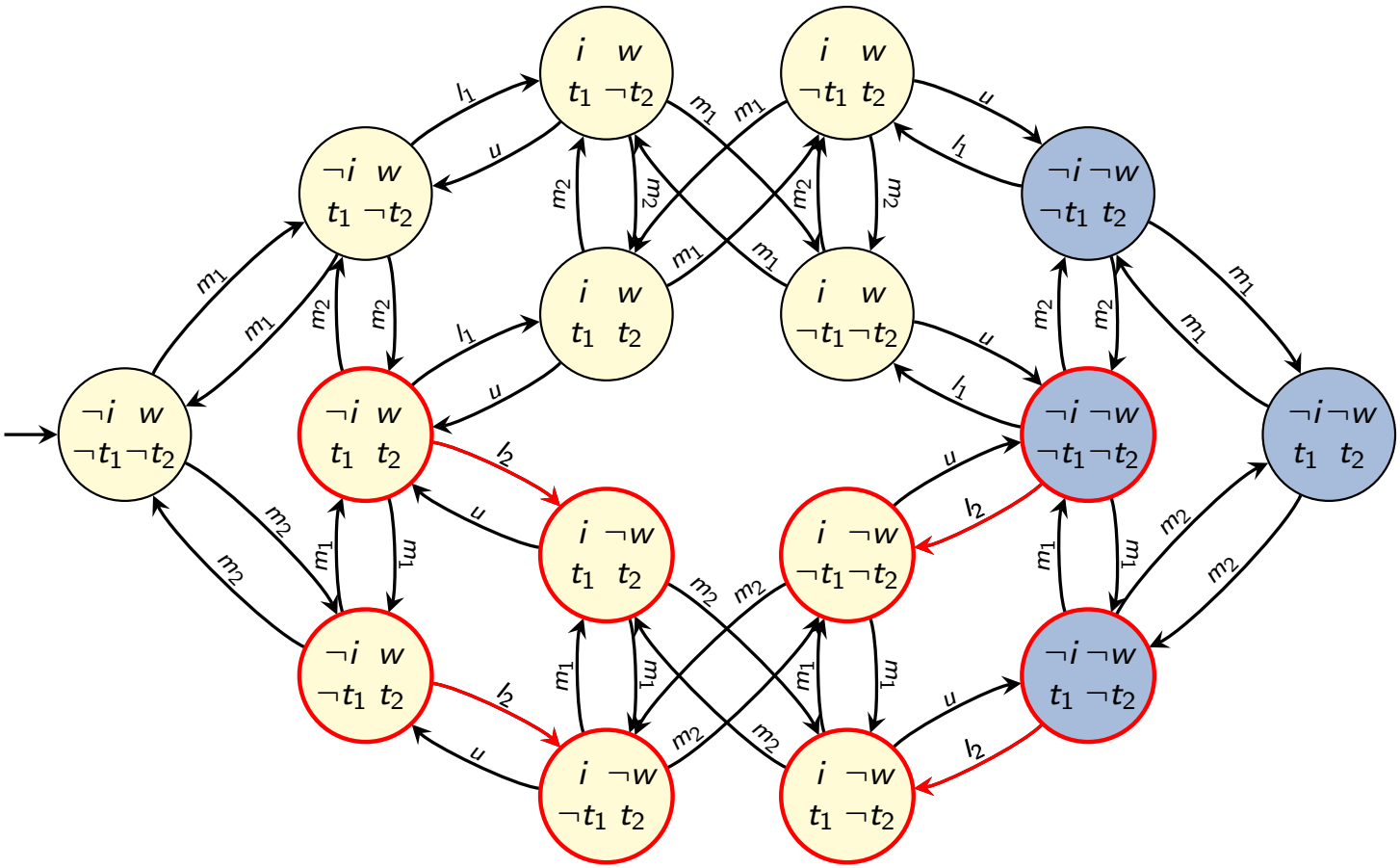
Running Example: Operator Effects

$$eff(l_1) = (i \wedge w)$$



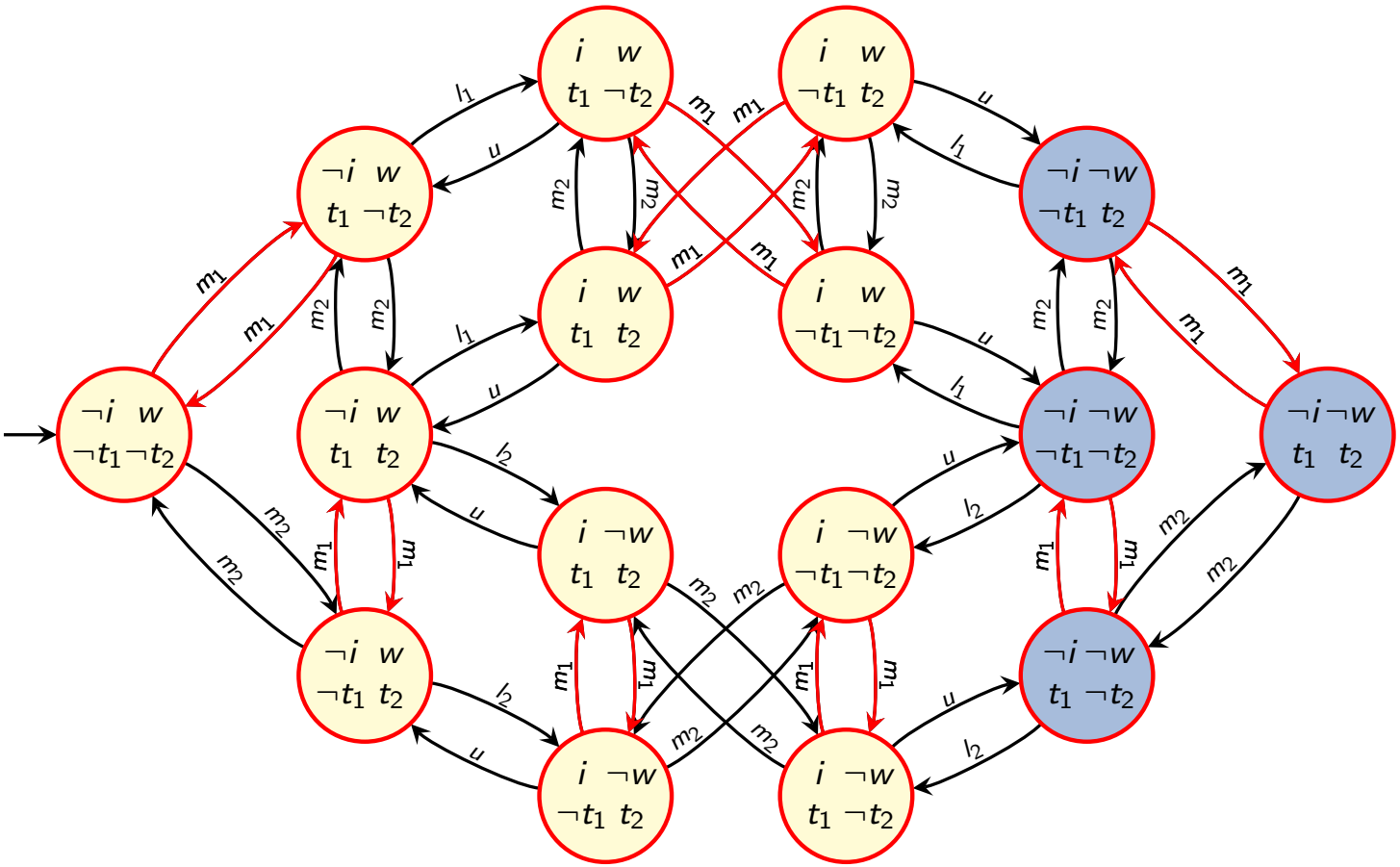
Running Example: Operator Effects

$$eff(l_2) = (i \wedge \neg w)$$



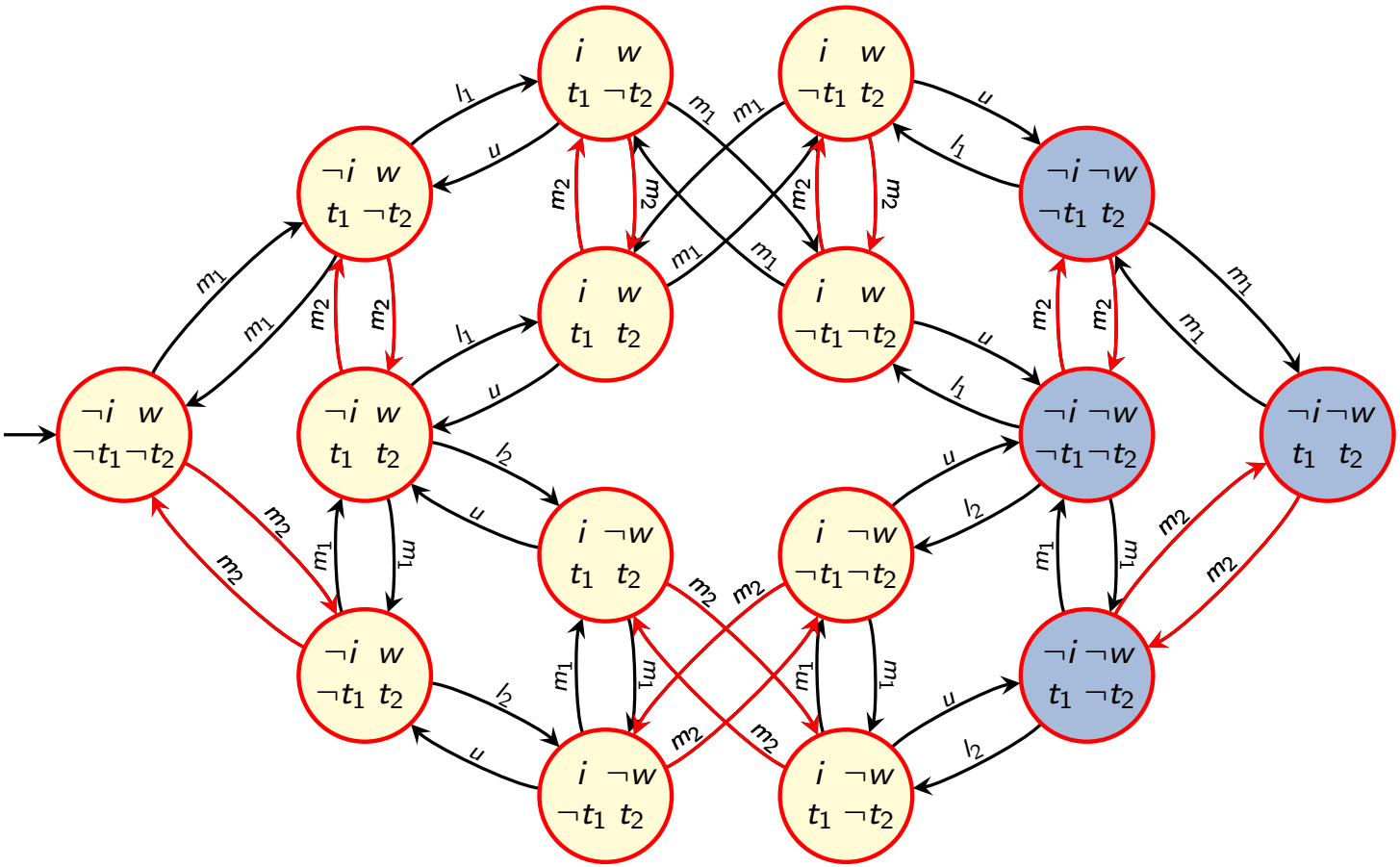
Running Example: Operator Effects

$$eff(m_1) = ((t_1 \rhd \neg t_1) \wedge (\neg t_1 \rhd t_1))$$



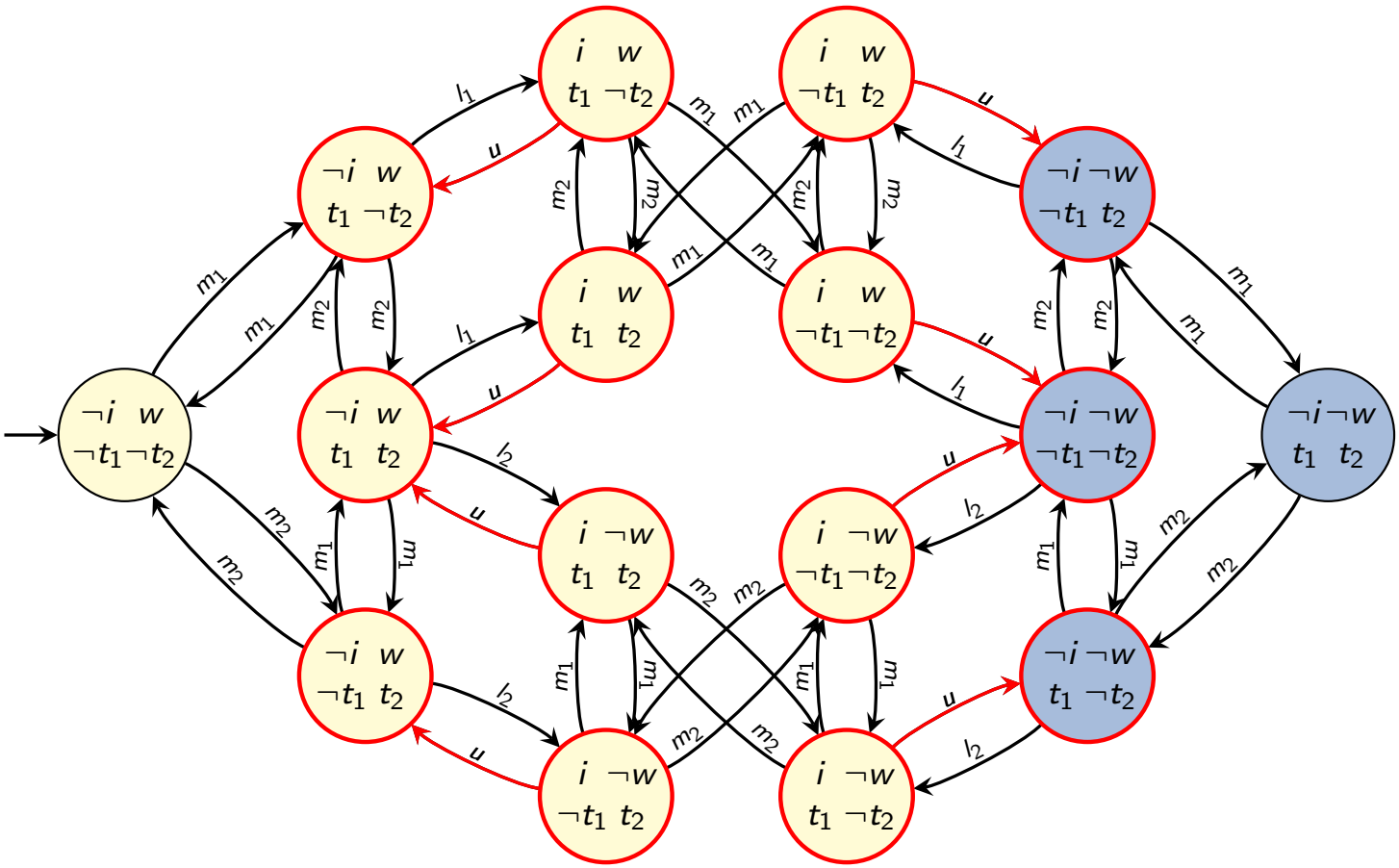
Running Example: Operator Effects

$$eff(m_2) = ((t_2 \rhd \neg t_2) \wedge (\neg t_2 \rhd t_2))$$



Running Example: Operator Effects

$$eff(u) = \neg i \wedge (w \rhd ((t_1 \rhd w) \wedge (\neg t_1 \rhd \neg w))) \\ \wedge (\neg w \rhd ((t_2 \rhd w) \wedge (\neg t_2 \rhd \neg w)))$$



Summary

Summary

- Propositional state variables let us compactly describe properties of large transition systems.
- A state is an assignment to a set of state variables.
- Sets of states are represented as formulas over state variables.
- Operators describe when (precondition), how (effect) and at which cost the state of the world can be changed.
- Effects are structured objects including empty, atomic, conjunctive and conditional effects.

Planning and Optimization

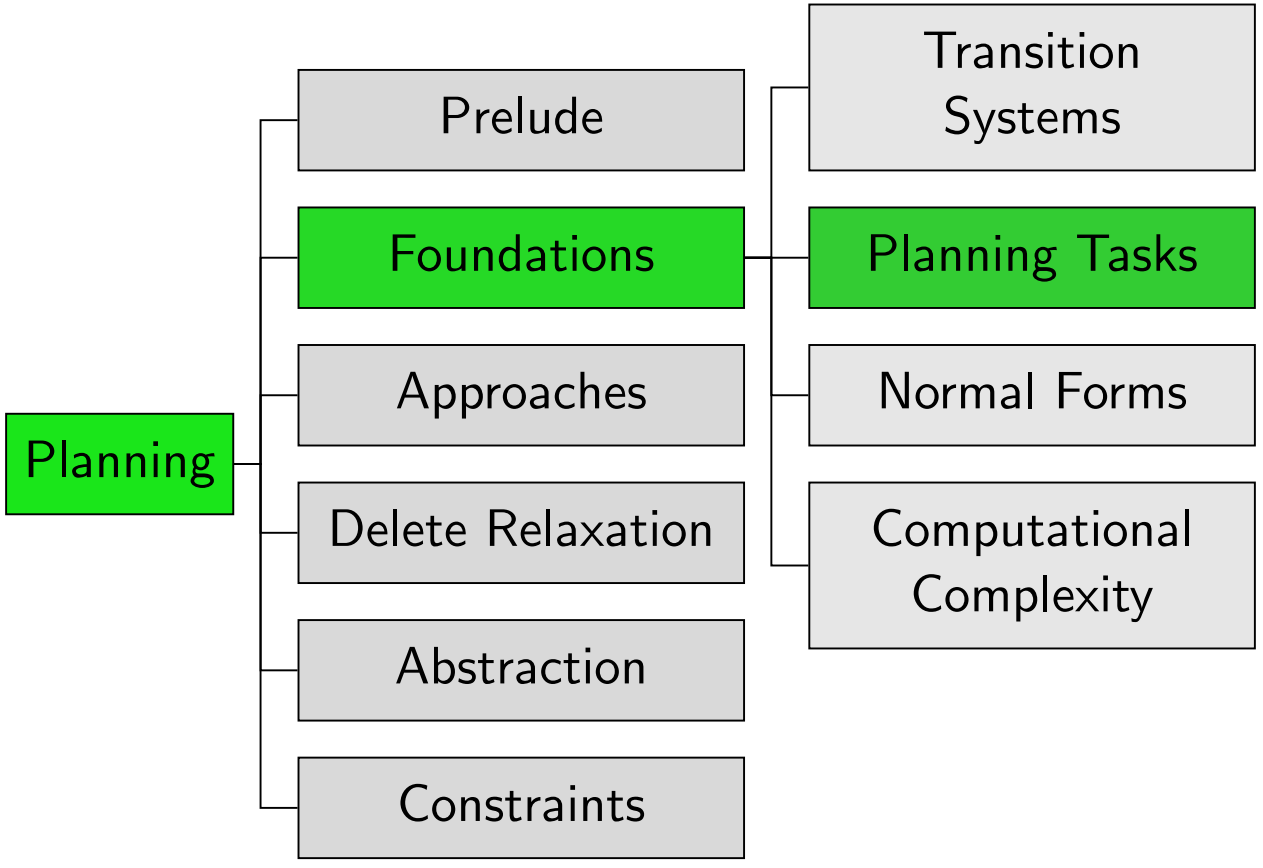
B3. Formal Definition of Planning

Malte Helmert and Gabriele Röger

Universität Basel

September 24, 2025

Content of the Course



Semantics of Effects and Operators

Semantics of Effects: Effect Conditions

Definition (Effect Condition for an Effect)

Let ℓ be an atomic effect, and let e be an effect.

The effect condition $\mathit{effcond}(\ell, e)$ under which ℓ triggers given the effect e is a propositional formula defined as follows:

- $\mathit{effcond}(\ell, \top) = \bot$
- $\mathit{effcond}(\ell, e) = \top$ for the atomic effect $e = \ell$
- $\mathit{effcond}(\ell, e) = \bot$ for all atomic effects $e = \ell' \neq \ell$
- $\mathit{effcond}(\ell, (e \wedge e')) = (\mathit{effcond}(\ell, e) \vee \mathit{effcond}(\ell, e'))$
- $\mathit{effcond}(\ell, (\chi \rhd e)) = (\chi \wedge \mathit{effcond}(\ell, e))$

Intuition: $\mathit{effcond}(\ell, e)$ represents the condition that must be true in the current state for the effect e to lead to the atomic effect ℓ

Effect Condition: Example (1)

Example

Consider the move operator m_1 from the running example:
$\mathit{eff}(m_1) = ((t_1 \rhd \neg t_1) \land (\neg t_1 \rhd t_1))$.

Under which conditions does it set t_1 to false?

$$\begin{aligned}
\mathit{effcond}(\neg t_1, \mathit{eff}(m_1)) &= \mathit{effcond}(\neg t_1, ((t_1 \rhd \neg t_1) \land (\neg t_1 \rhd t_1))) \\
&= \mathit{effcond}(\neg t_1, (t_1 \rhd \neg t_1)) \lor \\
&\quad \mathit{effcond}(\neg t_1, (\neg t_1 \rhd t_1)) \\
&= (t_1 \land \mathit{effcond}(\neg t_1, \neg t_1)) \lor \\
&\quad (\neg t_1 \land \mathit{effcond}(\neg t_1, t_1)) \\
&= (t_1 \land \top) \lor (\neg t_1 \land \bot) \\
&\equiv t_1 \lor \bot \\
&\equiv t_1
\end{aligned}$$

Effect Condition: Example (2)

Example

Consider the move operator m_1 from the running example:
$\mathit{eff}(m_1) = ((t_1 \rhd \neg t_1) \land (\neg t_1 \rhd t_1))$.

Under which conditions does it set i to true?

$$\begin{aligned}
\mathit{effcond}(i, \mathit{eff}(m_1)) &= \mathit{effcond}(i, ((t_1 \rhd \neg t_1) \land (\neg t_1 \rhd t_1))) \\
&= \mathit{effcond}(i, (t_1 \rhd \neg t_1)) \lor \\
&\quad \mathit{effcond}(i, (\neg t_1 \rhd t_1)) \\
&= (t_1 \land \mathit{effcond}(i, \neg t_1)) \lor \\
&\quad (\neg t_1 \land \mathit{effcond}(i, t_1)) \\
&= (t_1 \land \bot) \lor (\neg t_1 \land \bot) \\
&\equiv \bot \lor \bot \\
&\equiv \bot
\end{aligned}$$

Semantics of Effects: Applying an Effect

first attempt:

Definition (Applying Effects)

Let V be a set of propositional state variables.
Let s be a state over V, and let e be an effect over V.

The resulting state of applying e in s, written $s[\![e]\!]$,
is the state s' defined as follows for all $v \in V$:

$$s'(v) = \begin{cases} \mathbf{T} & \text{if } s \models \mathit{effcond}(v, e) \\ \mathbf{F} & \text{if } s \models \mathit{effcond}(\neg v, e) \\ s(v) & \text{otherwise} \end{cases}$$

What is the problem with this definition?

Semantics of Effects: Applying an Effect

correct definition:

Definition (Applying Effects)

Let V be a set of propositional state variables.
Let s be a state over V, and let e be an effect over V.

The resulting state of applying e in s, written $s[\![e]\!]$,
is the state s' defined as follows for all $v \in V$:

$$s'(v) = \begin{cases} \mathbf{T} & \text{if } s \models \mathit{effcond}(v, e) \\ \mathbf{F} & \text{if } s \models \mathit{effcond}(\neg v, e) \wedge \neg \mathit{effcond}(v, e) \\ s(v) & \text{otherwise} \end{cases}$$

Add-after-Delete Semantics

Note:

- The definition implies that if a variable is simultaneously "added" (set to **T**) and "deleted" (set to **F**), the value **T** takes precedence.
- This is called add-after-delete semantics.
- This detail of effect semantics is somewhat arbitrary, but has proven useful in applications.

Semantics of Operators

Definition (Applicable, Applying Operators, Resulting State)

Let V be a set of propositional state variables.
Let s be a state over V, and let o be an operator over V.

Operator o is applicable in s if $s \models \mathit{pre}(o)$.

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

Planning Tasks

Planning Tasks

Definition (Planning Task)

A (propositional) planning task is a 4-tuple $\Pi = \langle V, I, O, \gamma \rangle$ where

- V is a finite set of propositional state variables,
- I is an interpretation of V called the initial state,
- O is a finite set of operators over V, and
- γ is a formula over V called the goal.

Running Example: Planning Task

Example

From the previous chapter, we see that the running example can be represented by the task $\Pi = \langle V, I, O, \gamma \rangle$ with

- $V = \{i, w, t_1, t_2\}$
- $I = \{i \mapsto \mathbf{F}, w \mapsto \mathbf{T}, t_1 \mapsto \mathbf{F}, t_2 \mapsto \mathbf{F}\}$
- $O = \{m_1, m_2, l_1, l_2, u\}$ where
  - $m_1 = \langle \top, ((t_1 \rhd \neg t_1) \wedge (\neg t_1 \rhd t_1)), 5 \rangle$
  - $m_2 = \langle \top, ((t_2 \rhd \neg t_2) \wedge (\neg t_2 \rhd t_2)), 5 \rangle$
  - $l_1 = \langle \neg i \wedge (w \leftrightarrow t_1), (i \wedge w), 1 \rangle$
  - $l_2 = \langle \neg i \wedge (w \leftrightarrow t_2), (i \wedge \neg w), 1 \rangle$
  - $u = \langle i, \neg i \wedge (w \rhd ((t_1 \rhd w) \wedge (\neg t_1 \rhd \neg w))) \wedge (\neg w \rhd ((t_2 \rhd w) \wedge (\neg t_2 \rhd \neg w))), 1 \rangle$
- $\gamma = \neg i \wedge \neg w$

Mapping Planning Tasks to Transition Systems

Definition (Transition System Induced by a Planning Task)

The 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
- $S_\star = \{s \in S \mid s \models \gamma\}$.

Planning Tasks: Terminology

- Terminology for transitions systems is also applied to the planning tasks Π that induce them.
- For example, when we speak of the states of Π, we mean the states of $\mathcal{T}(\Pi)$.
- A sequence of operators that forms a solution of $\mathcal{T}(\Pi)$ is called a plan of Π.

Satisficing and Optimal Planning

By planning, we mean the following two algorithmic problems:

Definition (Satisficing Planning)

Given: a planning task Π
Output: a plan for Π, or **unsolvable** if no plan for Π exists

Definition (Optimal Planning)

Given: a planning task Π
Output: a plan for Π with minimal cost among all plans for Π, or **unsolvable** if no plan for Π exists

Summary

Summary

- Planning tasks compactly represent transition systems and are suitable as inputs for planning algorithms.
- A planning task consists of a set of state variables and an initial state, operators and goal over these state variables.
- We gave formal definitions for these concepts.
- In satisficing planning, we must find a solution for a planning task (or show that no solution exists).
- In optimal planning, we must additionally guarantee that generated solutions are of minimal cost.

Planning and Optimization

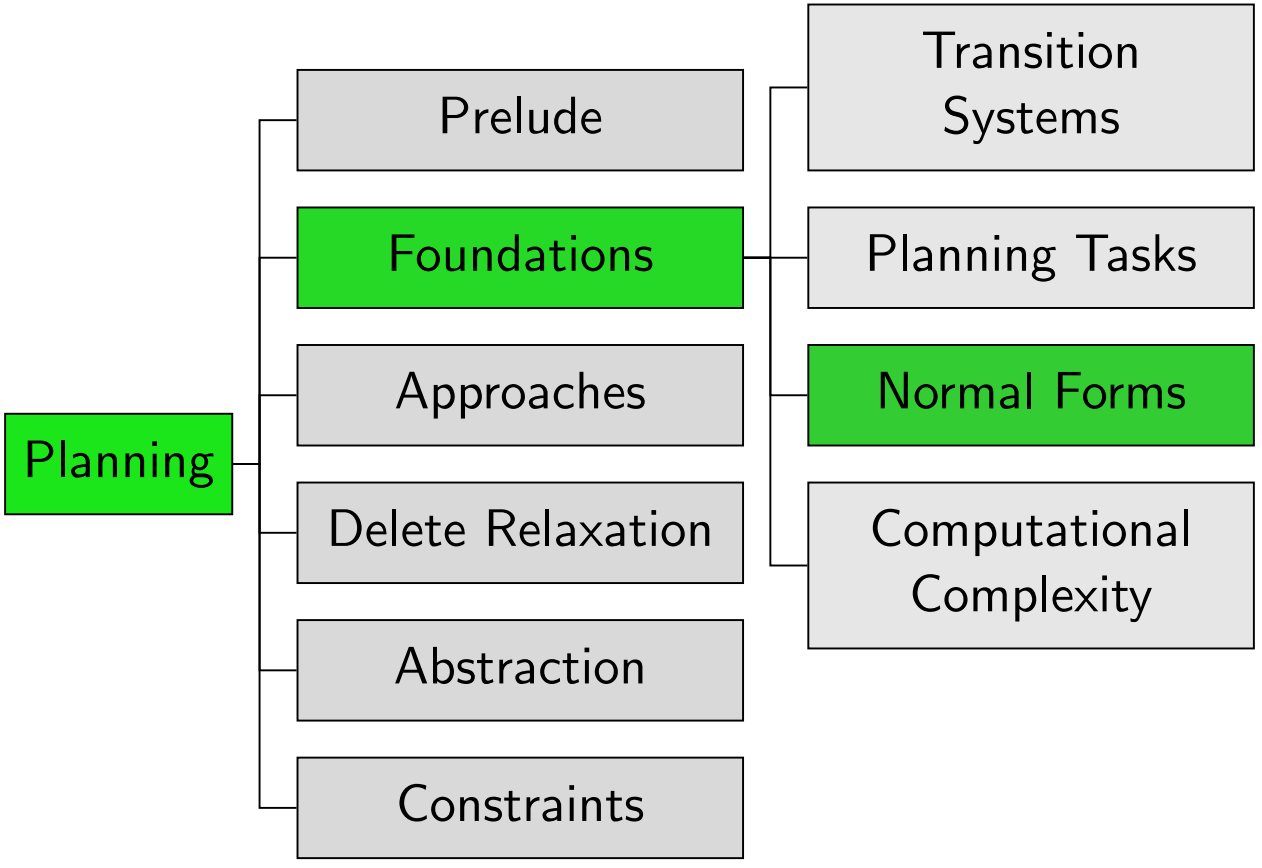
B4. Equivalent Operators and Normal Forms

Malte Helmert and Gabriele Röger

Universität Basel

September 29, 2025

Content of the Course



Reminder & Motivation

Reminder: Syntax of Effects

Definition (Effect)

Effects over propositional state variables V
are inductively defined as follows:

- $\top$ is an effect (empty effect).
- If $v \in V$ is a propositional state variable, then v and $\neg v$ are effects (atomic effect).
- If e and e' are effects, then $(e \wedge e')$ is an effect (conjunctive effect).
- If χ is a formula over V and e is an effect, then $(\chi \rhd e)$ is an effect (conditional effect).

Arbitrary nesting of conjunctive and conditional effects,
e.g. $c \wedge (a \rhd (\neg b \wedge (c \rhd (b \wedge \neg d \wedge \neg a)))) \wedge (\neg b \rhd \neg a)$
$\rightsquigarrow$ Can we make our life easier?

Reminder: Semantics of Effects

- $\mathit{effcond}(\ell, e)$: condition that must be true in the current state for the effect e to trigger the atomic effect ℓ
- add-after-delete semantics:
  if an operator with effect e is applied in state s
  and we have both $s \models \mathit{effcond}(v, e)$ and $s \models \mathit{effcond}(\neg v, e)$,
  then $s'(v) = \mathbf{T}$ in the resulting state s'.

This is a very subtle detail.

$\rightsquigarrow$ Can we make our life easier?

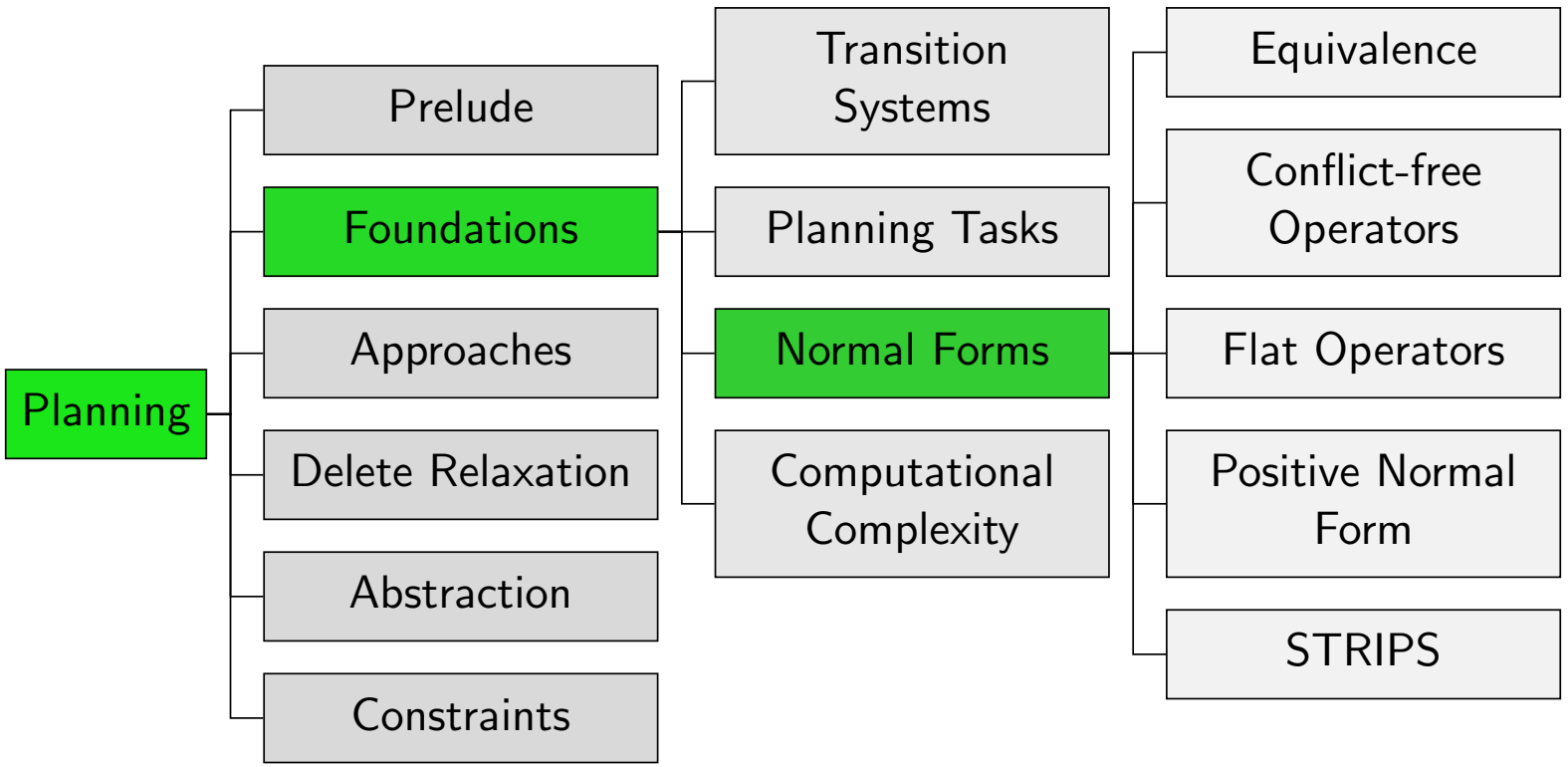
Motivation

Similarly to normal forms in propositional logic (DNF, CNF, NNF), we can define normal forms for effects, operators and planning tasks.

Among other things, we consider normal forms that avoid complicated nesting and subtleties of conflicts.

This is useful because algorithms (and proofs) then only need to deal with effects, operators and tasks in normal form.

Content of the Course



Notation: Applying Operator Sequences

Existing notation:

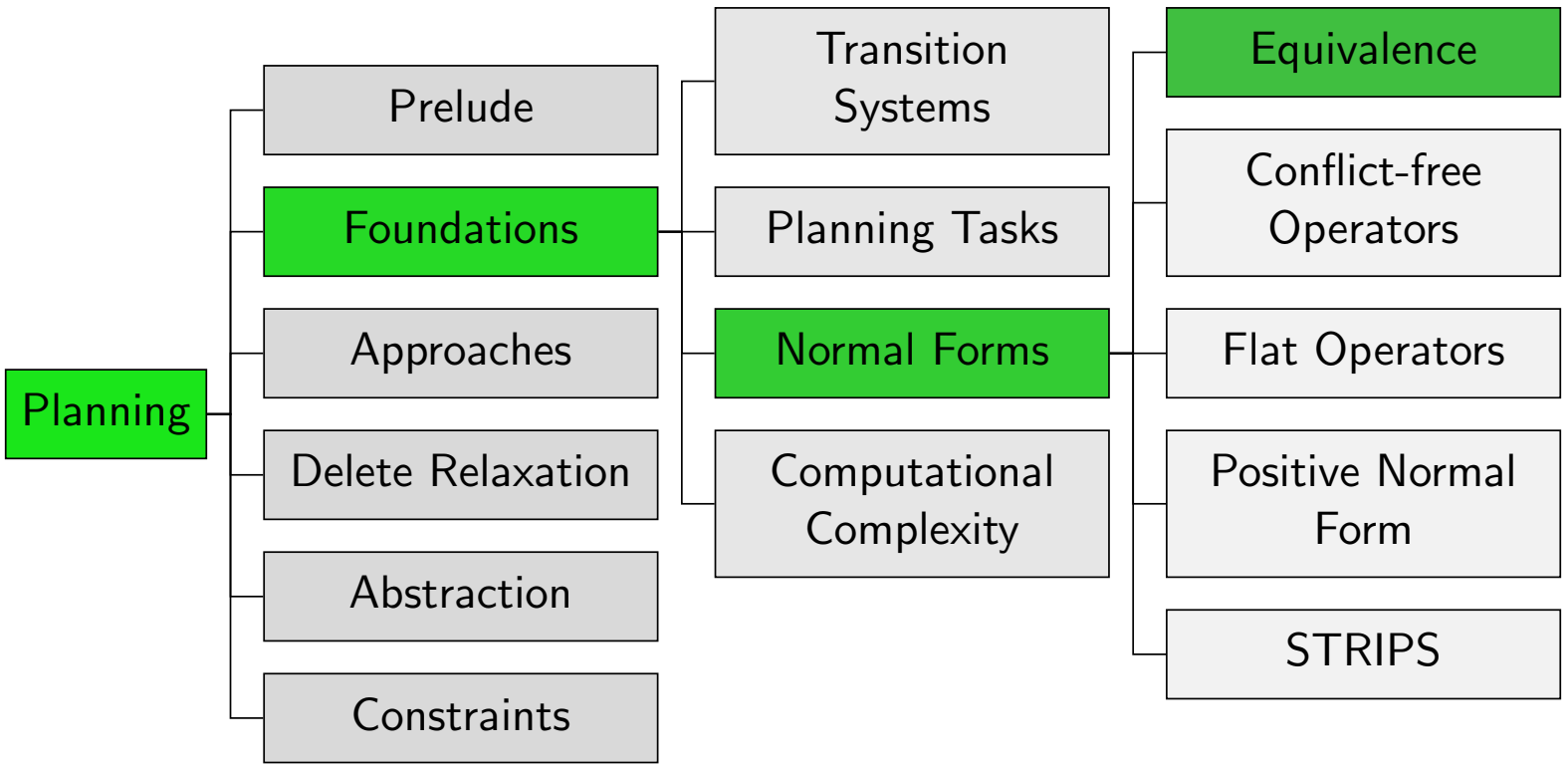
- We already write $s[\![o]\!]$ for the resulting state after applying operator o in state s.

New extended notation:

- For a sequence $\pi = \langle o_1, \ldots, o_n \rangle$ of operators that are consecutively applicable in s, we write $s[\![\pi]\!]$ for $s[\![o_1]\!][\![o_2]\!] \ldots [\![o_n]\!]$.
- This includes the case of an empty operator sequence: $s[\![\langle\rangle]\!] = s$

Equivalence Transformations

Content of the Course



Equivalence of Operators and Effects: Definition

Definition (Equivalent Effects)

Two effects e and e' over state variables V are equivalent, written $e \equiv e'$, if $s[\![e]\!] = s[\![e']\!]$ for all states s.

Definition (Equivalent Operators)

Two operators o and o' over state variables V are equivalent, written $o \equiv o'$, if $\mathit{cost}(o) = \mathit{cost}(o')$ and for all states s, s' over V, o induces the transition $s \xrightarrow{o} s'$ iff o' induces the transition $s \xrightarrow{o'} s'$.

Equivalence of Operators and Effects: Theorem

Theorem

Let o and o' be operators with $pre(o) \equiv pre(o')$, $eff(o) \equiv eff(o')$ and $cost(o) = cost(o')$. Then $o \equiv o'$.

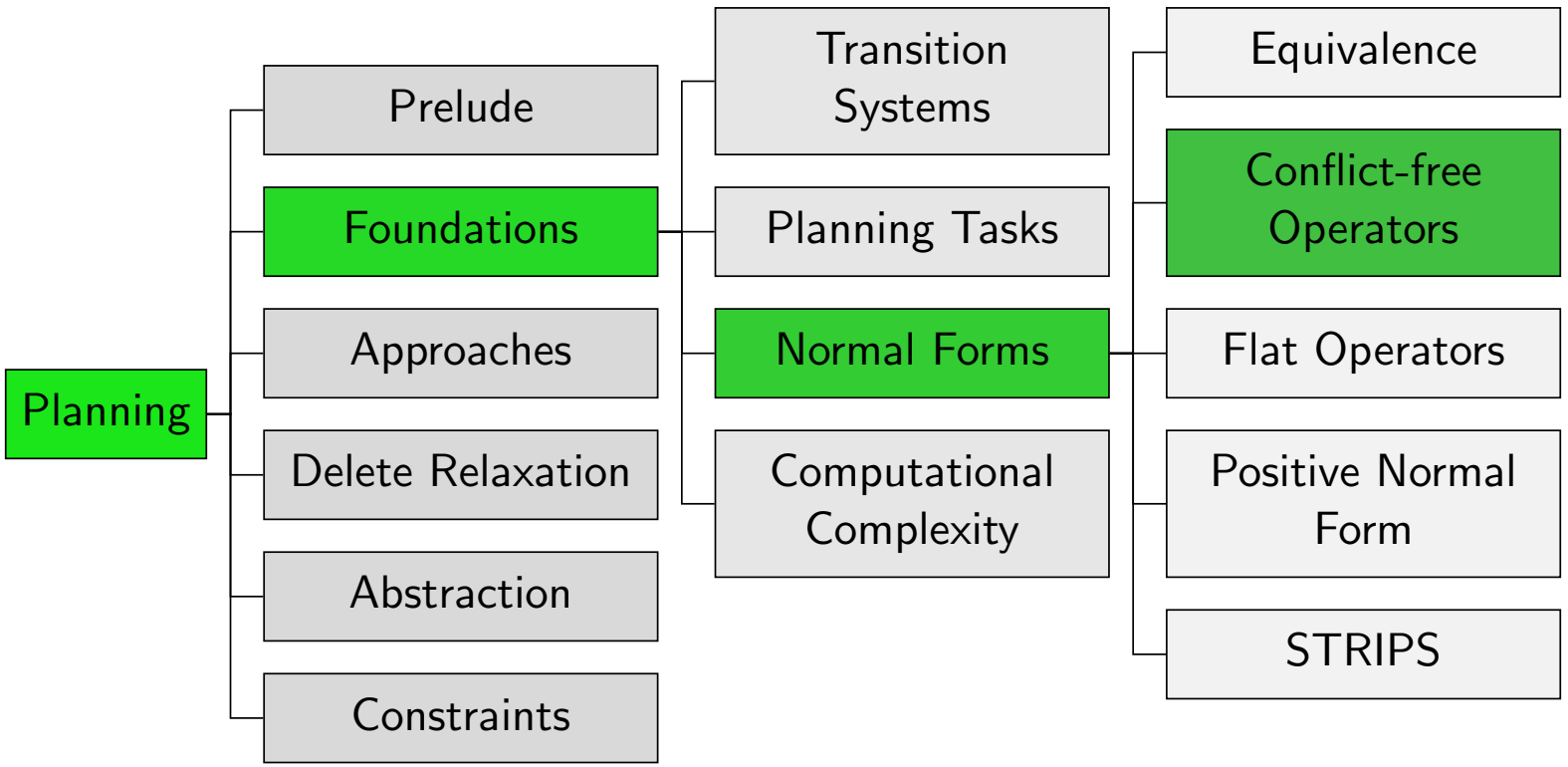
Note: The converse is not true. (Why not?)

Equivalence Transformations for Effects

$$e \wedge e' \equiv e' \wedge e \tag{1}$$
$$(e \wedge e') \wedge e'' \equiv e \wedge (e' \wedge e'') \tag{2}$$
$$\top \wedge e \equiv e \tag{3}$$
$$\chi \rhd e \equiv \chi' \rhd e \qquad \text{if } \chi \equiv \chi' \tag{4}$$
$$\top \rhd e \equiv e \tag{5}$$
$$\bot \rhd e \equiv \top \tag{6}$$
$$\chi \rhd (\chi' \rhd e) \equiv (\chi \wedge \chi') \rhd e \tag{7}$$
$$\chi \rhd (e \wedge e') \equiv (\chi \rhd e) \wedge (\chi \rhd e') \tag{8}$$
$$(\chi \rhd e) \wedge (\chi' \rhd e) \equiv (\chi \vee \chi') \rhd e \tag{9}$$

Conflict-Free Operators

Content of the Course



Conflict-Freeness: Motivation

- The add-after-delete semantics makes effects like $(a \rhd c) \wedge (b \rhd \neg c)$ somewhat unintuitive to interpret.

$\rightsquigarrow$ What happens in states where $a \wedge b$ is true?

- It would be nicer if $\mathit{effcond}(\ell, e)$ always were the condition under which the atomic effect ℓ actually materializes (because of add-after-delete, it is not)

$\rightsquigarrow$ introduce normal form where "complicated case" never arises

Conflict-Free Effects and Operators

Definition (Conflict-Free)

An effect e over propositional state variables V is called conflict-free if $\mathit{effcond}(v, e) \wedge \mathit{effcond}(\neg v, e)$ is unsatisfiable for all $v \in V$.

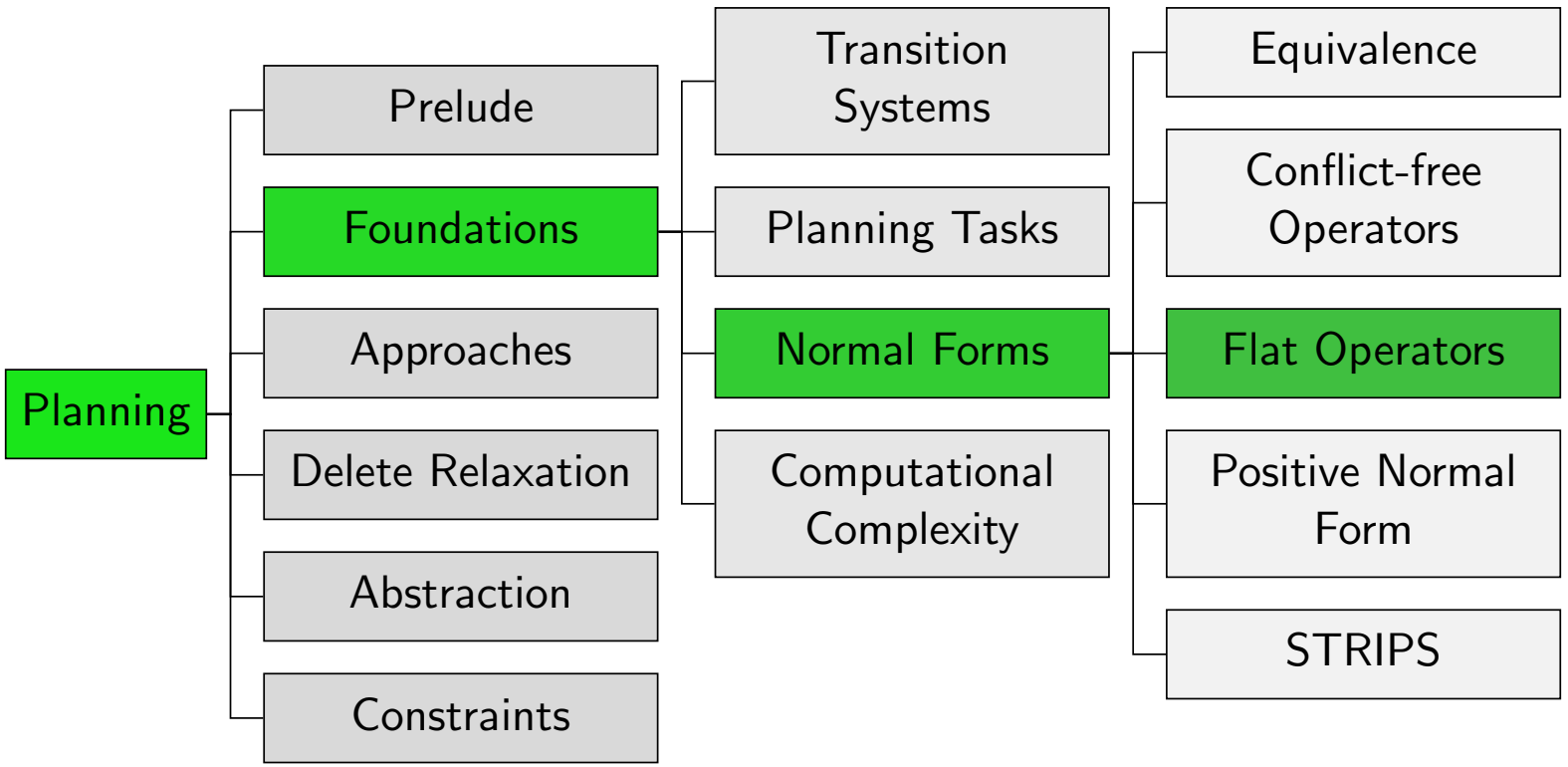
An operator o is called conflict-free if $\mathit{eff}(o)$ is conflict-free.

Making Operators Conflict-Free

- In general, testing whether an operator is conflict-free is a coNP-complete problem. (Why?)
- However, we do not necessarily need such a test. Instead, we can produce an equivalent conflict-free operator in polynomial time.
- Algorithm: given operator o, replace all atomic effects of the form $\neg v$ by $(\neg \mathit{effcond}(v, \mathit{eff}(o)) \rhd \neg v)$. The resulting operator o' is conflict-free and $o \equiv o'$. (Why?)

Flat Effects

Content of the Course



Flat Effects: Motivation

- CNF and DNF limit the nesting of connectives in propositional logic.
- For example, a CNF formula is
  - a conjunction of 0 or more subformulas,
  - each of which is a disjunction of 0 or more subformulas,
  - each of which is a literal.
- Similarly, we can define a normal form that limits the nesting of effects.
- This is useful because we then do not have to consider arbitrarily structured effects, e.g., when representing them in a planning algorithm.

Flat Effect

Definition (Flat Effect)

An effect is simple if it is either an atomic effect
or of the form $(\chi \rhd e)$, where e is an atomic effect.

An effect e is flat if it is a conjunction of 0 or more simple effects, and none of these simple effects include the same atomic effect.

An operator o is flat if $\mathit{eff}(o)$ is flat.

Notes: analogously to CNF, we consider

- a single simple effect as a conjunction of 1 simple effect
- the empty effect as a conjunction of 0 simple effects

Flat Effect: Example

Example

Consider the effect

$$c \wedge (a \rhd (\neg b \wedge (c \rhd (b \wedge \neg d \wedge \neg a)))) \wedge (\neg b \rhd \neg a)$$

An equivalent flat (and conflict-free) effect is

$$\begin{gathered} c \wedge \\ ((a \wedge \neg c) \rhd \neg b) \wedge \\ ((a \wedge c) \rhd b) \wedge \\ ((a \wedge c) \rhd \neg d) \wedge \\ ((\neg b \vee (a \wedge c)) \rhd \neg a) \end{gathered}$$

Note: if we want, we can write c as $(\top \rhd c)$ to make the structure even more uniform, with each simple effect having a condition.

Producing Flat Operators

Theorem

For every operator, an equivalent flat operator and an equivalent flat, conflict-free operator can be computed in polynomial time.

Producing Flat Operators: Proof

Proof Sketch.

Replace the effect e over variables V by

$$\begin{aligned} &\textstyle\bigwedge_{v \in V} (\mathit{effcond}(v, e) \rhd v) \\ \wedge\, &\textstyle\bigwedge_{v \in V} (\mathit{effcond}(\neg v, e) \rhd \neg v), \end{aligned}$$

which is an equivalent flat effect.

To additionally obtain conflict-freeness, use

$$\begin{aligned} &\textstyle\bigwedge_{v \in V} (\mathit{effcond}(v, e) \rhd v) \\ \wedge\, &\textstyle\bigwedge_{v \in V} ((\mathit{effcond}(\neg v, e) \wedge \neg \mathit{effcond}(v, e)) \rhd \neg v) \end{aligned}$$

instead.

(Conjuncts of the form $(\chi \rhd e)$ where $\chi \equiv \bot$ can be omitted to simplify the effect.)

Summary

Summary

- Equivalences can be used to simplify operators and effects.
- In conflict-free operators, the "complicated case" of operator semantics does not arise.
- For flat operators, the only permitted nesting is atomic effects within conditional effects within conjunctive effects, and all atomic effects must be distinct.
- For flat, conflict-free operators, it is easy to determine the condition under which a given literal is made true by applying the operator in a given state.
- Every operator can be transformed into an equivalent flat and conflict-free one in polynomial time.

Planning and Optimization

B5. Positive Normal Form and STRIPS

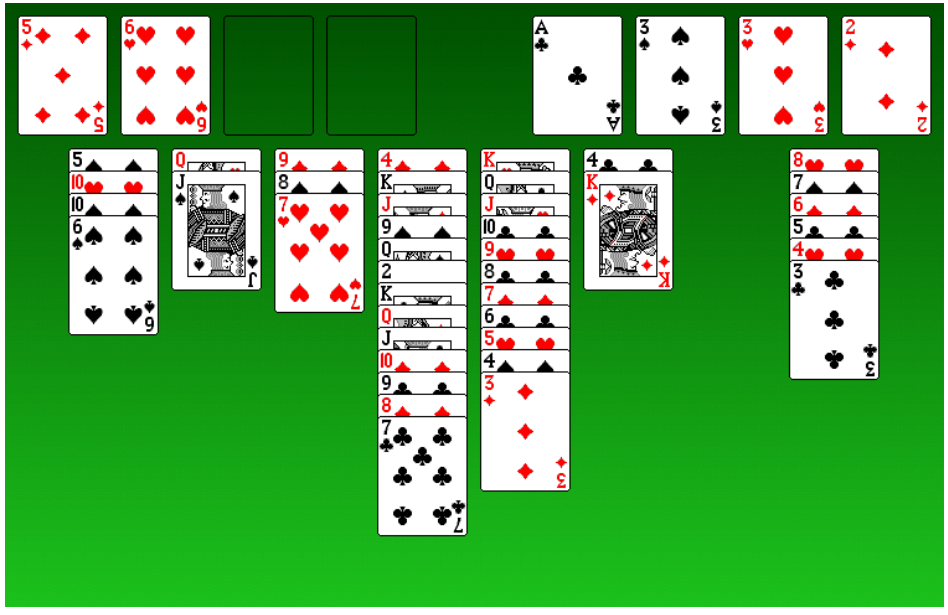
Malte Helmert and Gabriele Röger

Universität Basel

September 29, 2025

Motivation

Example: Freecell



Example (Good and Bad Effects)

If we move K♢ to a free tableau position,
the good effect is that 4♣ is now accessible.
The bad effect is that we lose one free tableau position.

What is a Good or Bad Effect?

Question: Which operator effects are good, and which are bad?

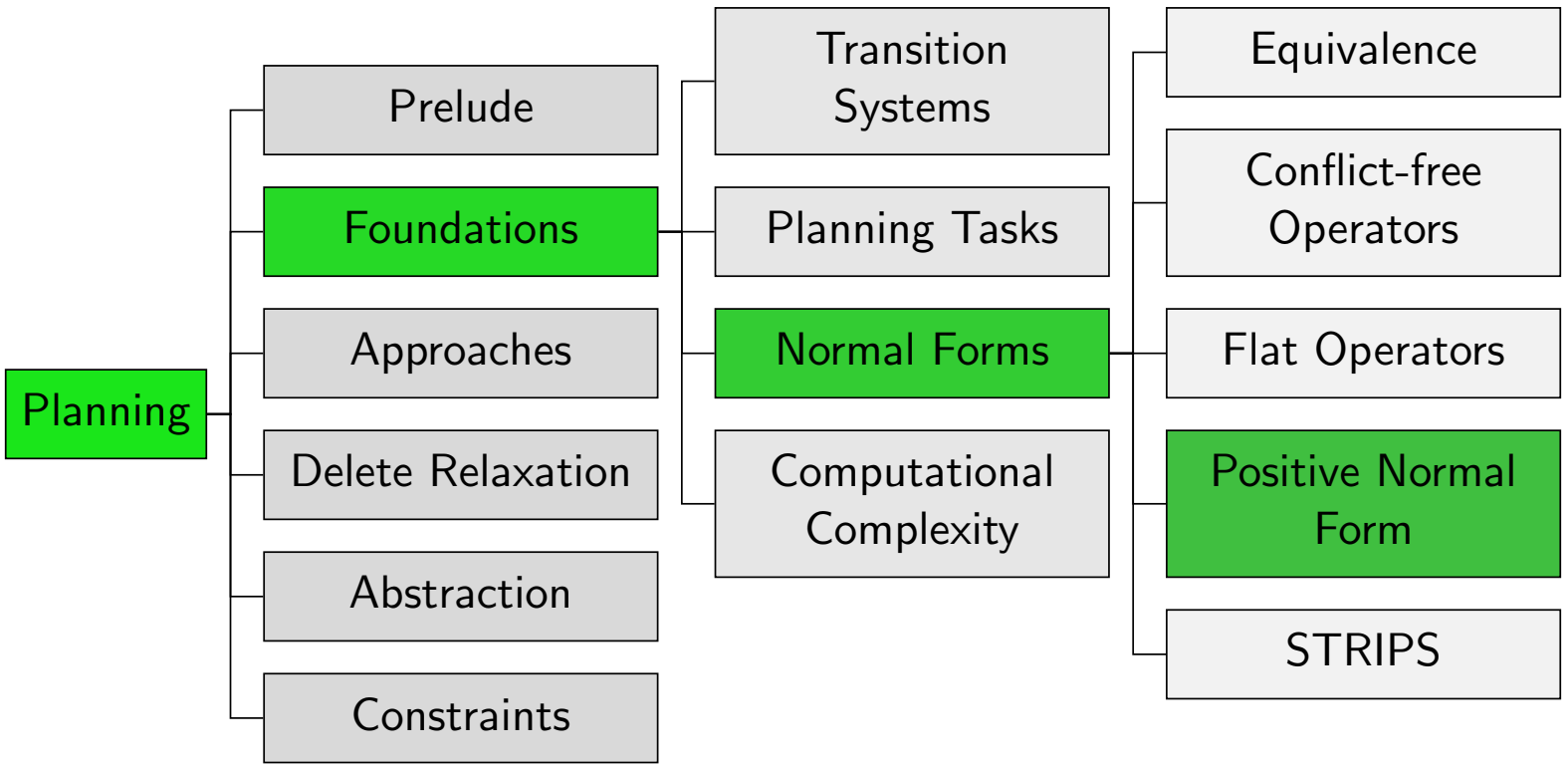
Difficult to answer in general, because it depends on context:

- Locking our door is good if we want to keep burglars out.
- Locking our door is bad if we want to enter.

We now consider a reformulation of propositional planning tasks that makes the distinction between good and bad effects obvious.

Positive Normal Form

Content of the Course



Positive Formulas, Operators and Tasks

Definition (Positive Formula)

A logical formula φ is positive if no negation symbols appear in φ.

Note: This includes the negation symbols implied by $\rightarrow$ and $\leftrightarrow$.

Definition (Positive Operator)

An operator o is positive if $\mathit{pre}(o)$ and
all effect conditions in $\mathit{eff}(o)$ are positive.

Definition (Positive Propositional Planning Task)

A propositional planning task $\langle V, I, O, \gamma \rangle$ is positive
if all operators in O and the goal γ are positive.

Positive Normal Form

Definition (Positive Normal Form)

A propositional planning task is in positive normal form
if it is positive and all operator effects are flat.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T}, \mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \neg \mathit{home} \wedge \mathit{uni} \rangle, \langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \rangle, \langle \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \mathit{bike\text{-}locked} \rangle, \langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \neg \mathit{bike\text{-}locked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T}, \mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \neg \mathit{home} \wedge \mathit{uni} \rangle, \langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \rangle, \langle \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \mathit{bike\text{-}locked} \rangle, \langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \neg \mathit{bike\text{-}locked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Identify state variable v occurring negatively in conditions.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}, \mathit{bike\text{-}unlocked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T},$$
$$\mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}, \mathit{bike\text{-}unlocked} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \neg \mathit{home} \wedge \mathit{uni} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \rangle,$$
$$\langle \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \mathit{bike\text{-}locked} \rangle,$$
$$\langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \neg \mathit{bike\text{-}locked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Introduce new variable $\hat{v}$ with complementary initial value.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}, \mathit{bike\text{-}unlocked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T},$$
$$\mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}, \mathit{bike\text{-}unlocked} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \neg \mathit{home} \wedge \mathit{uni} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \rangle,$$
$$\langle \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \mathit{bike\text{-}locked} \rangle,$$
$$\langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \neg \mathit{bike\text{-}locked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Identify effects on variable v.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}, \mathit{bike\text{-}unlocked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T},$$
$$\mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}, \mathit{bike\text{-}unlocked} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \neg \mathit{home} \wedge \mathit{uni} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \wedge \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \mathit{bike\text{-}locked} \wedge \neg \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \neg \mathit{bike\text{-}locked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Introduce complementary effects for $\hat{v}$.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}, \mathit{bike\text{-}unlocked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T},$$
$$\mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}, \mathit{bike\text{-}unlocked} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \neg \mathit{home} \wedge \mathit{uni} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \wedge \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{bike} \wedge \neg \mathit{bike\text{-}locked}, \mathit{bike\text{-}locked} \wedge \neg \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \neg \mathit{bike\text{-}locked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Identify negative conditions for v.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}, \mathit{bike\text{-}unlocked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T},$$
$$\mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}, \mathit{bike\text{-}unlocked} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \mathit{bike\text{-}unlocked}, \neg \mathit{home} \wedge \mathit{uni} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \wedge \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}unlocked}, \mathit{bike\text{-}locked} \wedge \neg \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \mathit{bike\text{-}unlocked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Replace by positive condition $\hat{v}$.

Positive Normal Form: Example

Example (Transformation to Positive Normal Form)

$$V = \{\mathit{home}, \mathit{uni}, \mathit{lecture}, \mathit{bike}, \mathit{bike\text{-}locked}, \mathit{bike\text{-}unlocked}\}$$

$$I = \{\mathit{home} \mapsto \mathbf{T}, \mathit{bike} \mapsto \mathbf{T}, \mathit{bike\text{-}locked} \mapsto \mathbf{T},$$
$$\mathit{uni} \mapsto \mathbf{F}, \mathit{lecture} \mapsto \mathbf{F}, \mathit{bike\text{-}unlocked} \mapsto \mathbf{F}\}$$

$$O = \{\langle \mathit{home} \wedge \mathit{bike} \wedge \mathit{bike\text{-}unlocked}, \neg \mathit{home} \wedge \mathit{uni} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}locked}, \neg \mathit{bike\text{-}locked} \wedge \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{bike} \wedge \mathit{bike\text{-}unlocked}, \mathit{bike\text{-}locked} \wedge \neg \mathit{bike\text{-}unlocked} \rangle,$$
$$\langle \mathit{uni}, \mathit{lecture} \wedge ((\mathit{bike} \wedge \mathit{bike\text{-}unlocked}) \rhd \neg \mathit{bike}) \rangle\}$$

$$\gamma = \mathit{lecture} \wedge \mathit{bike}$$

Positive Normal Form: Existence

Theorem (Positive Normal Form)

For every propositional planning task Π, there is an equivalent propositional planning task Π' in positive normal form. Moreover, Π' can be computed from Π in polynomial time.

Note: Equivalence here means that the transition systems induced by Π and Π', restricted to the reachable states, are isomorphic.

We prove the theorem by describing a suitable algorithm.
(However, we do not prove its correctness or complexity.)

Positive Normal Form: Algorithm

Transformation of $\langle V, I, O, \gamma \rangle$ to Positive Normal Form

Replace all operators with equivalent conflict-free operators.
Convert all conditions to negation normal form (NNF).
while any condition contains a negative literal $\neg v$:

Let v be a variable which occurs negatively in a condition.
$V := V \cup \{\hat{v}\}$ for some new propositional state variable $\hat{v}$

$$I(\hat{v}) := \begin{cases} \mathbf{F} & \text{if } I(v) = \mathbf{T} \\ \mathbf{T} & \text{if } I(v) = \mathbf{F} \end{cases}$$

Replace the effect v by $(v \wedge \neg\hat{v})$ in all operators $o \in O$.
Replace the effect $\neg v$ by $(\neg v \wedge \hat{v})$ in all operators $o \in O$.
Replace $\neg v$ by $\hat{v}$ in all conditions.

Convert all operators $o \in O$ to flat operators.

Here, all conditions refers to all operator preconditions, operator effect conditions and the goal.

Why Positive Normal Form is Interesting

In the absence of conditional effects, positive normal form allows us to distinguish good and bad effects easily:

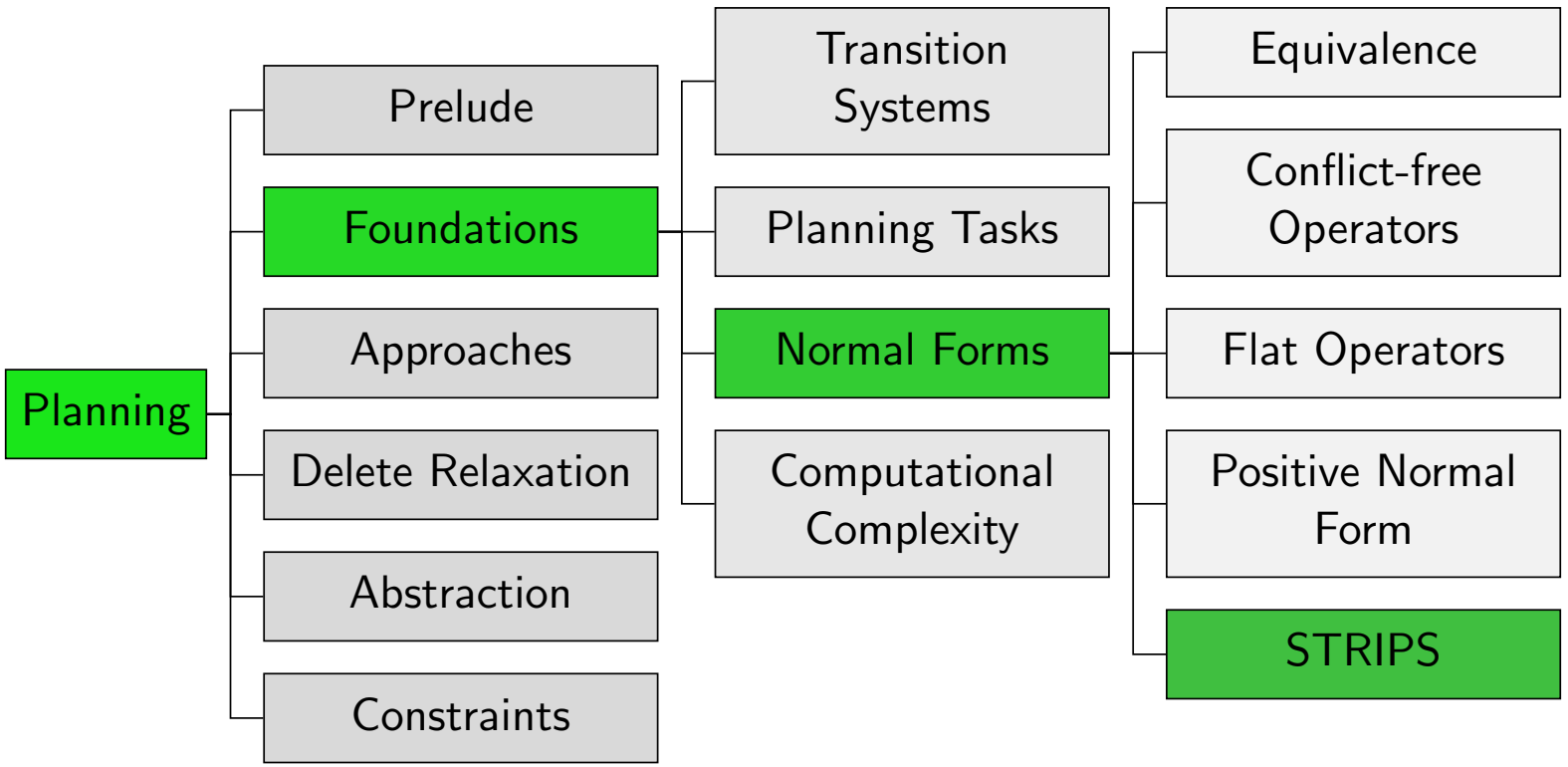
- Effects that make state variables true (add effects) are good.
- Effects that make state variables false (delete effects) are bad.

This is particularly useful for planning algorithms based on delete relaxation, which we will study in Part D.

(Why restriction "in the absence of conditional effects"?)

STRIPS

Content of the Course



STRIPS Operators and Planning Tasks

Definition (STRIPS Operator)

An operator o of a prop. planning task is a STRIPS operator if

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

Definition (STRIPS Planning Task)

A propositional planning task $\langle V, I, O, \gamma \rangle$ is a STRIPS planning task if all operators $o \in O$ are STRIPS operators and γ is a conjunction of state variables.

Note: STRIPS operators are conflict-free and flat.
STRIPS is a special case of positive normal form.

STRIPS Operators: Remarks

- Every STRIPS operator is of the form

$$\langle v_1 \wedge \cdots \wedge v_n, \quad \ell_1 \wedge \cdots \wedge \ell_m \rangle$$

  where v_i are state variables and ℓ_j are atomic effects.
- Often, STRIPS operators o are described via three sets of state variables:
  - the preconditions (state variables occurring in $pre(o)$)
  - the add effects (state variables occurring positively in $eff(o)$)
  - the delete effects (state variables occurring negatively in $eff(o)$)
- Definitions of STRIPS in the literature often do not require conflict-freeness. But it is easy to achieve and makes many things simpler.
- There exists a variant called STRIPS with negation where negative literals are also allowed in conditions.

Why STRIPS is Interesting

- STRIPS is particularly simple, yet expressive enough to capture general planning tasks.
- In particular, STRIPS planning is no easier than planning in general (as we will see in Chapters B6–B7).
- Many algorithms in the planning literature are only presented for STRIPS planning tasks (generalization is often, but not always, obvious).

STRIPS

STanford Research Institute Problem Solver
(Fikes & Nilsson, 1971)

Transformation to STRIPS

- Not every operator is equivalent to a STRIPS operator.
- However, each operator can be transformed into a set of STRIPS operators whose "combination" is equivalent to the original operator. (How?)
- However, this transformation may exponentially increase the number of operators. There are planning tasks for which such a blow-up is unavoidable.
- There are polynomial transformations of propositional planning tasks to STRIPS, but these do not lead to isomorphic transition systems (auxiliary states are needed). (They are, however, equivalent in a weaker sense.)

Summary

Summary

- A positive task helps distinguish good and bad effects.
  The notion of positive tasks only exists for propositional tasks.
- A positive task with flat operators is in positive normal form.
- STRIPS is even more restrictive than positive normal form,
  forbidding complex preconditions and conditional effects.
- Both forms are expressive enough to capture
  general propositional planning tasks.
- Transformation to positive normal form is possible
  with polynomial size increase.
- Isomorphic transformations of propositional planning tasks to STRIPS can increase the number of operators exponentially;
  non-isomorphic polynomial transformations exist.

Planning and Optimization

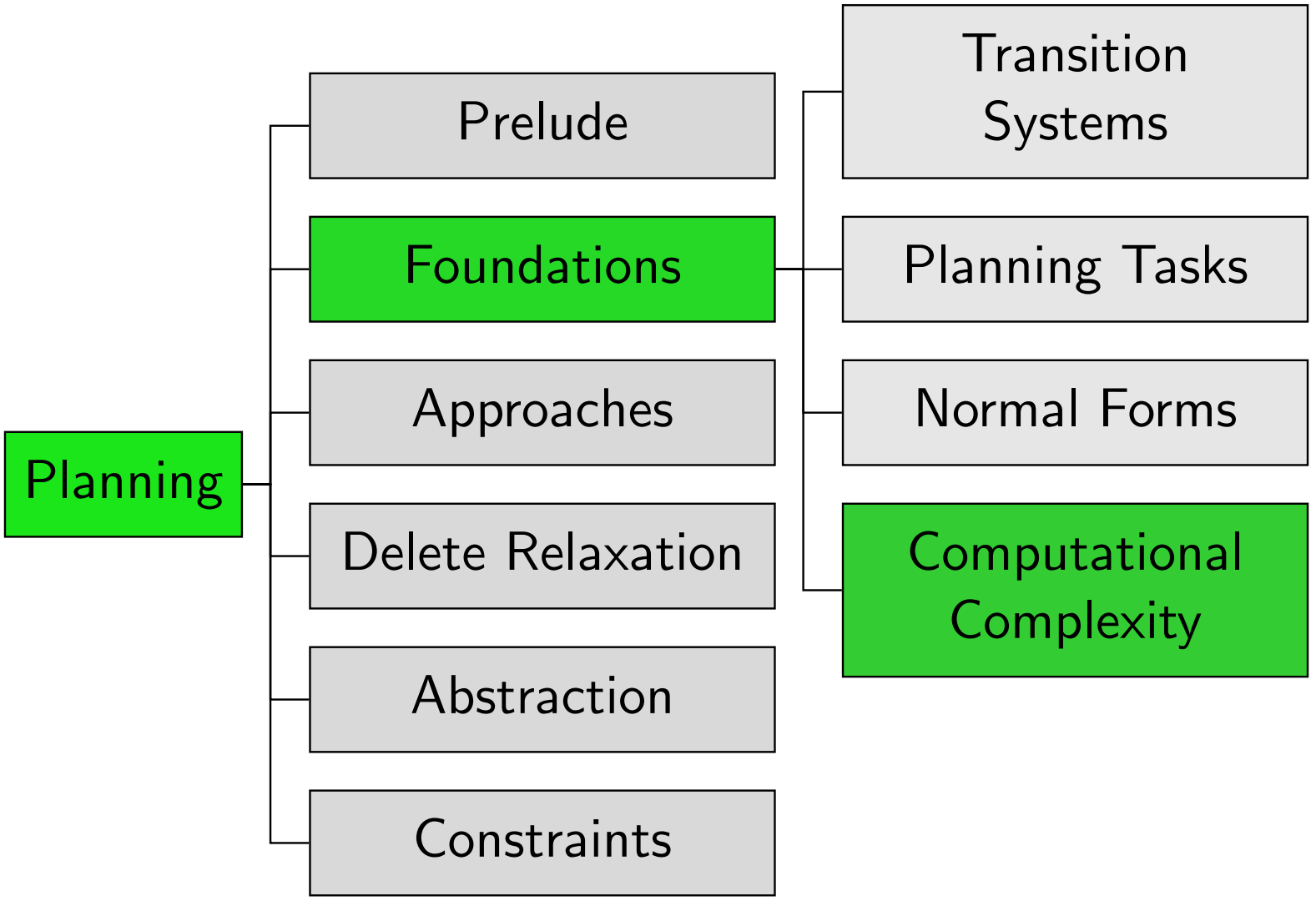
B6. Computational Complexity of Planning: Background

Malte Helmert and Gabriele Röger

Universität Basel

October 1, 2025

Content of the Course



Motivation

How Difficult is Planning?

- Using state-space search (e.g., using Dijkstra's algorithm on the transition system), planning can be solved in polynomial time in the number of states.
- However, the number of states is exponential in the number of state variables, and hence in general exponential in the size of the input to the planning algorithm.

$\rightsquigarrow$ Do non-exponential planning algorithms exist?

$\rightsquigarrow$ What is the precise computational complexity of planning?

Why Computational Complexity?

- understand the problem
- know what is not possible
- find interesting subproblems that are easier to solve
- distinguish essential features from syntactic sugar
  - Is STRIPS planning easier than general planning?

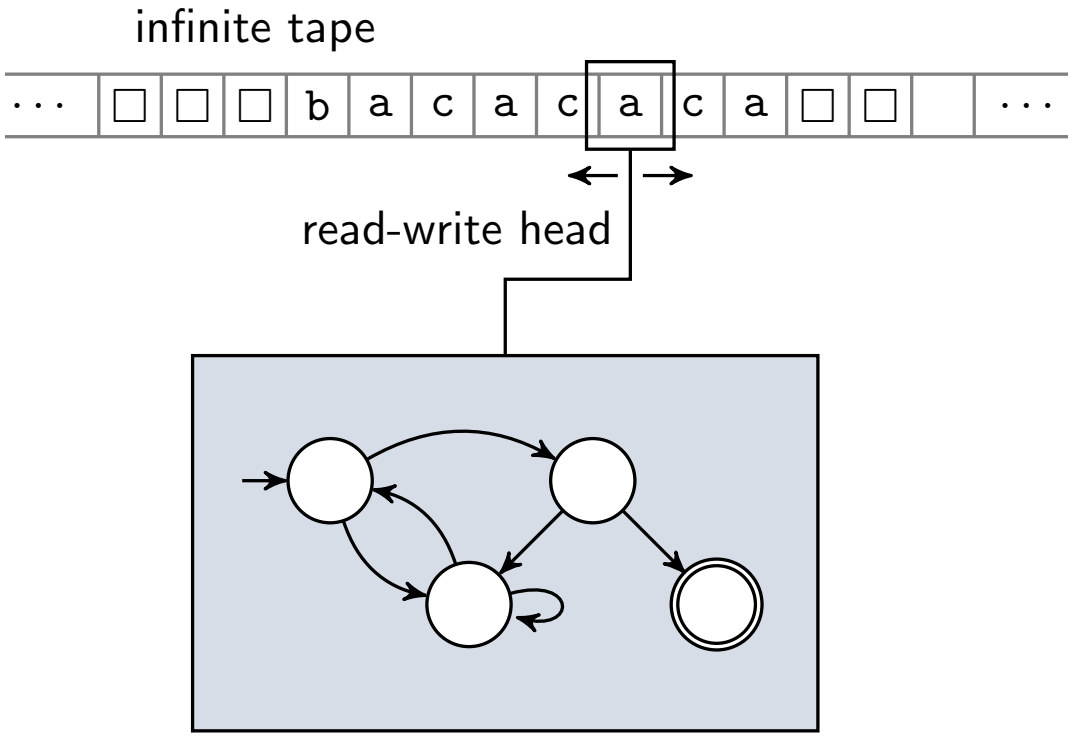
Reminder: Complexity Theory

Need to Catch Up?

- We assume knowledge of complexity theory:
  - languages and decision problems
  - Turing machines: NTMs and DTMs;
    polynomial equivalence with other models of computation
  - complexity classes: P, NP, PSPACE
  - polynomial reductions
- If you are not familiar with these topics, we recommend Chapters B11, D1–D3, D6 of the Theory of Computer Science course at `https://dmi.unibas.ch/en/studium/computer-science-informatik/lehrangebot-fs25/10948-main-lecture-theory-of-computer-science/`

Turing Machines

Turing Machines: Conceptually



Turing Machines

Definition (Nondeterministic Turing Machine)

A nondeterministic Turing machine (NTM) is a 6-tuple $\langle \Sigma, \square, Q, q_0, q_Y, \delta \rangle$ with the following components:

- input alphabet Σ and blank symbol $\square \notin \Sigma$
  - alphabets always nonempty and finite
  - tape alphabet $\Sigma_\square = \Sigma \cup \{\square\}$
- finite set Q of internal states with initial state $q_0 \in Q$ and accepting state $q_Y \in Q$
  - nonterminal states $Q' := Q \setminus \{q_Y\}$
- transition relation $\delta : (Q' \times \Sigma_\square) \to 2^{Q \times \Sigma_\square \times \{-1,+1\}}$

Deterministic Turing machine (DTM):
$|\delta(q, s)| = 1$ for all $\langle q, s \rangle \in Q' \times \Sigma_\square$

Turing Machines: Accepted Words

- Initial configuration
  - state q_0
  - input word on tape, all other tape cells contain $\square$
  - head on first symbol of input word
- Step
  - If in state q, reading symbol s, and $\langle q', s', d \rangle \in \delta(q, s)$ then
  - the NTM can transition to state q', replacing s with s' and moving the head one cell to the left/right ($d = -1/+1$).
- Input word ($\in \Sigma^*$) is accepted if some sequence of transitions brings the NTM from the initial configuration into state q_Y.

Complexity Classes

Acceptance in Time and Space

Definition (Acceptance of a Language in Time/Space)

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$.

A NTM accepts language $L \subseteq \Sigma^*$ in time f if it accepts each $w \in L$ within $f(|w|)$ steps and does not accept any $w \notin L$ (in any time).

It accepts language $L \subseteq \Sigma^*$ in space f if it accepts each $w \in L$ using at most $f(|w|)$ tape cells and does not accept any $w \notin L$.

Time and Space Complexity Classes

Definition (DTIME, NTIME, DSPACE, NSPACE)

Let $f : \mathbb{N}_0 \to \mathbb{N}_0$.

Complexity class DTIME(f) contains all languages accepted in time f by some DTM.

Complexity class NTIME(f) contains all languages accepted in time f by some NTM.

Complexity class DSPACE(f) contains all languages accepted in space f by some DTM.

Complexity class NSPACE(f) contains all languages accepted in space f by some NTM.

Polynomial Time and Space Classes

Let $\mathcal{P}$ be the set of polynomials $p : \mathbb{N}_0 \to \mathbb{N}_0$ whose coefficients are natural numbers.

Definition (P, NP, PSPACE, NPSPACE)

$$\begin{aligned} \mathrm{P} &= \bigcup_{p \in \mathcal{P}} \mathrm{DTIME}(p) \\ \mathrm{NP} &= \bigcup_{p \in \mathcal{P}} \mathrm{NTIME}(p) \\ \mathrm{PSPACE} &= \bigcup_{p \in \mathcal{P}} \mathrm{DSPACE}(p) \\ \mathrm{NPSPACE} &= \bigcup_{p \in \mathcal{P}} \mathrm{NSPACE}(p) \end{aligned}$$

Polynomial Complexity Class Relationships

Theorem (Complexity Class Hierarchy)

$P \subseteq NP \subseteq PSPACE = NPSPACE$

Proof.

$P \subseteq NP$ and $PSPACE \subseteq NPSPACE$ are obvious because deterministic Turing machines are a special case of nondeterministic ones.

$NP \subseteq NPSPACE$ holds because a Turing machine can only visit polynomially many tape cells within polynomial time.

$PSPACE = NPSPACE$ is a special case of a classical result known as Savitch's theorem (Savitch 1970). □

Summary

Summary

- We recalled the definitions of the most important complexity classes from complexity theory:
  - P: decision problems solvable in polynomial time
  - NP: decision problems solvable in polynomial time by nondeterministic algorithms
  - PSPACE: decision problems solvable in polynomial space
  - NPSPACE: decision problems solvable in polynomial space by nondeterministic algorithms
- These classes are related by $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE}$.

Planning and Optimization

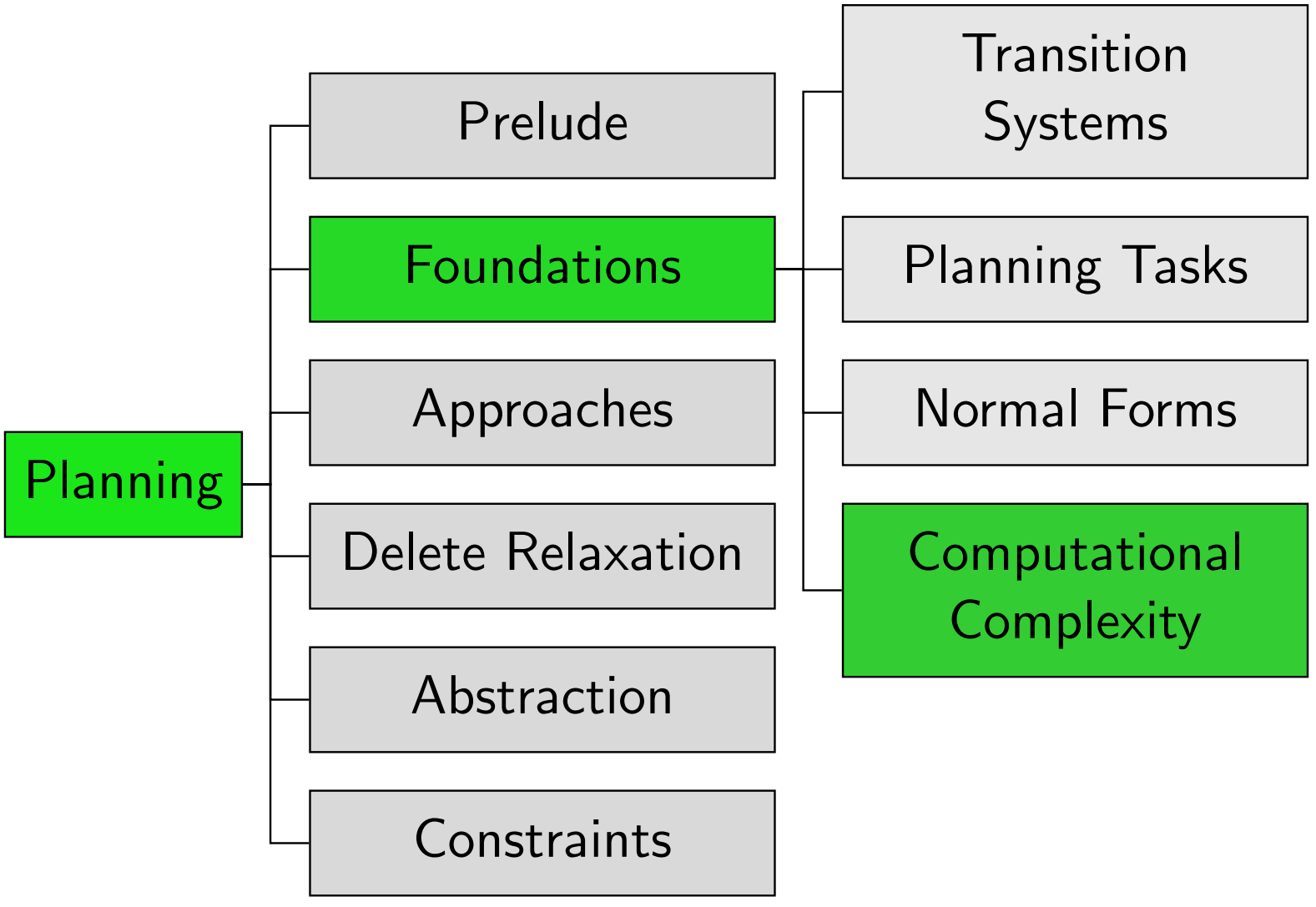
B7. Computational Complexity of Planning: Results

Malte Helmert and Gabriele Röger

Universität Basel

October 1, 2025

Content of the Course



(Bounded-Cost) Plan Existence

Decision Problems for Planning

Definition (Plan Existence)

Plan existence (PlanEx) is the following decision problem:

Given: planning task Π

Question: Is there a plan for Π?

$\rightsquigarrow$ decision problem analogue of satisficing planning

Definition (Bounded-Cost Plan Existence)

Bounded-cost plan existence (BCPlanEx)
is the following decision problem:

Given: planning task Π, cost bound $K \in \mathbb{N}_0$

Question: Is there a plan for Π with cost at most K?

$\rightsquigarrow$ decision problem analogue of optimal planning

Plan Existence vs. Bounded-Cost Plan Existence

Theorem (Reduction from PlanEx to BCPlanEx)

$\textsc{PlanEx} \leq_p \textsc{BCPlanEx}$

Proof.

Consider a planning task Π with state variables V.

Let $c_{\max}$ be the maximal cost of all operators of Π.

Compute the number of states of Π as $N = 2^{|V|}$.

Π is solvable iff there is solution with cost at most $c_{\max} \cdot (N - 1)$ because a solution need not visit any state twice.

$\rightsquigarrow$ map instance Π of PlanEx to instance $\langle \Pi, c_{\max} \cdot (N - 1) \rangle$ of BCPlanEx

$\rightsquigarrow$ polynomial reduction $\square$

PSPACE-Completeness of Planning

Membership in PSPACE

Theorem

$\textsc{BCPlanEx} \in \text{PSPACE}$

Proof.

Show $\textsc{BCPlanEx} \in \text{NPSPACE}$ and use Savitch's theorem.

Nondeterministic algorithm:

def plan($\langle V, I, O, \gamma \rangle$, K):
    $s := I$
    $k := K$
    loop forever:
        if $s \models \gamma$: **accept**
        guess $o \in O$
        if o is not applicable in s: **fail**
        if $cost(o) > k$: **fail**
        $s := s[\![o]\!]$
        $k := k - cost(o)$

□

PSPACE-Hardness

Idea: generic reduction

- For an arbitrary fixed DTM M with space bound polynomial p and input w, generate propositional planning task which is solvable iff M accepts w in space $p(|w|)$.
- Without loss of generality, we assume $p(n) \geq n$ for all n.

Reduction: State Variables

Let $M = \langle \Sigma, \square, Q, q_0, q_Y, \delta \rangle$ be the fixed DTM, and let p be its space-bound polynomial.

Given input $w_1 \dots w_n$, define relevant tape positions $X := \{-p(n), \dots, p(n)\}$

State Variables

- state_q for all $q \in Q$
- head_i for all $i \in X \cup \{-p(n) - 1, p(n) + 1\}$
- $\text{content}_{i,a}$ for all $i \in X$, $a \in \Sigma_\square$

$\rightsquigarrow$ allows encoding a Turing machine configuration

Reduction: Initial State

Let $M = \langle \Sigma, \square, Q, q_0, q_Y, \delta \rangle$ be the fixed DTM, and let p be its space-bound polynomial.

Given input $w_1 \dots w_n$, define relevant tape positions $X := \{-p(n), \dots, p(n)\}$

Initial State

Initially true:

- state_{q_0}
- head_1
- $\text{content}_{i,w_i}$ for all $i \in \{1, \dots, n\}$
- $\text{content}_{i,\square}$ for all $i \in X \setminus \{1, \dots, n\}$

Initially false:

- all others

Reduction: Operators

Let $M = \langle \Sigma, \square, Q, q_0, q_Y, \delta \rangle$ be the fixed DTM,
and let p be its space-bound polynomial.

Given input $w_1 \ldots w_n$, define relevant tape positions
$X := \{-p(n), \ldots, p(n)\}$

Operators

One operator for each transition rule $\delta(q, a) = \langle q', a', d \rangle$
and each cell position $i \in X$:

- precondition: $\text{state}_q \wedge \text{head}_i \wedge \text{content}_{i,a}$
- effect: $\neg\text{state}_q \wedge \neg\text{head}_i \wedge \neg\text{content}_{i,a}$
  $\wedge\, \text{state}_{q'} \wedge \text{head}_{i+d} \wedge \text{content}_{i,a'}$

Note that add-after-delete semantics are important here!

Reduction: Goal

Let $M = \langle \Sigma, \square, Q, q_0, q_Y, \delta \rangle$ be the fixed DTM,
and let p be its space-bound polynomial.

Given input $w_1 \dots w_n$, define relevant tape positions
$X := \{-p(n), \dots, p(n)\}$

Goal

state_{q_Y}

PSPACE-Completeness of STRIPS Plan Existence

Theorem (PSPACE-Completeness; Bylander, 1994)

PlanEx *and* BCPlanEx *are* PSPACE*-complete.*
This is true even if only STRIPS tasks are allowed.

Proof.

Membership for BCPlanEx was already shown.

Hardness for PlanEx follows because we just presented a polynomial reduction from an arbitrary problem in PSPACE to PlanEx. (Note that the reduction only generates STRIPS tasks, after trivial cleanup to make them conflict-free.)

Membership for PlanEx and hardness for BCPlanEx follow from the polynomial reduction from PlanEx to BCPlanEx. □

More Complexity Results

More Complexity Results

In addition to the basic complexity result presented in this chapter, there are many special cases, generalizations, variations and related problems studied in the literature:

- different planning formalisms
  - e.g., nondeterministic effects, partial observability, schematic operators, numerical state variables
- syntactic restrictions of planning tasks
  - e.g., without preconditions, without conjunctive effects, STRIPS without delete effects
- semantic restrictions of planning task
  - e.g., restricting variable dependencies ("causal graphs")
- particular planning domains
  - e.g., Blocksworld, Logistics, FreeCell

Complexity Results for Different Planning Formalisms

Some results for different planning formalisms:

- nondeterministic effects:
  - fully observable: EXP-complete (Littman, 1997)
  - unobservable: EXPSPACE-complete (Haslum & Jonsson, 1999)
  - partially observable: 2-EXP-complete (Rintanen, 2004)
- schematic operators:
  - usually adds one exponential level to PLANEX complexity
  - e.g., classical case EXPSPACE-complete (Erol et al., 1995)
- numerical state variables:
  - undecidable in most variations (Helmert, 2002)
  - decidable in restricted setting with at most two numeric state variables (Helal and Lakemeyer, 2025)

Summary

Summary

- Classical planning is PSPACE-complete.
- This is true both for satisficing and optimal planning (rather, the corresponding decision problems).
- The hardness proof is a polynomial reduction that translates an arbitrary polynomial-space DTM into a STRIPS task:
  - DTM configurations are encoded by state variables.
  - Operators simulate transitions between DTM configurations.
  - The DTM accepts an input iff there is a plan for the corresponding STRIPS task.
- This implies that there is no polynomial algorithm for classical planning unless P = PSPACE.
- It also means that planning is not polynomially reducible to any problem in NP unless NP = PSPACE.

Planning and Optimization

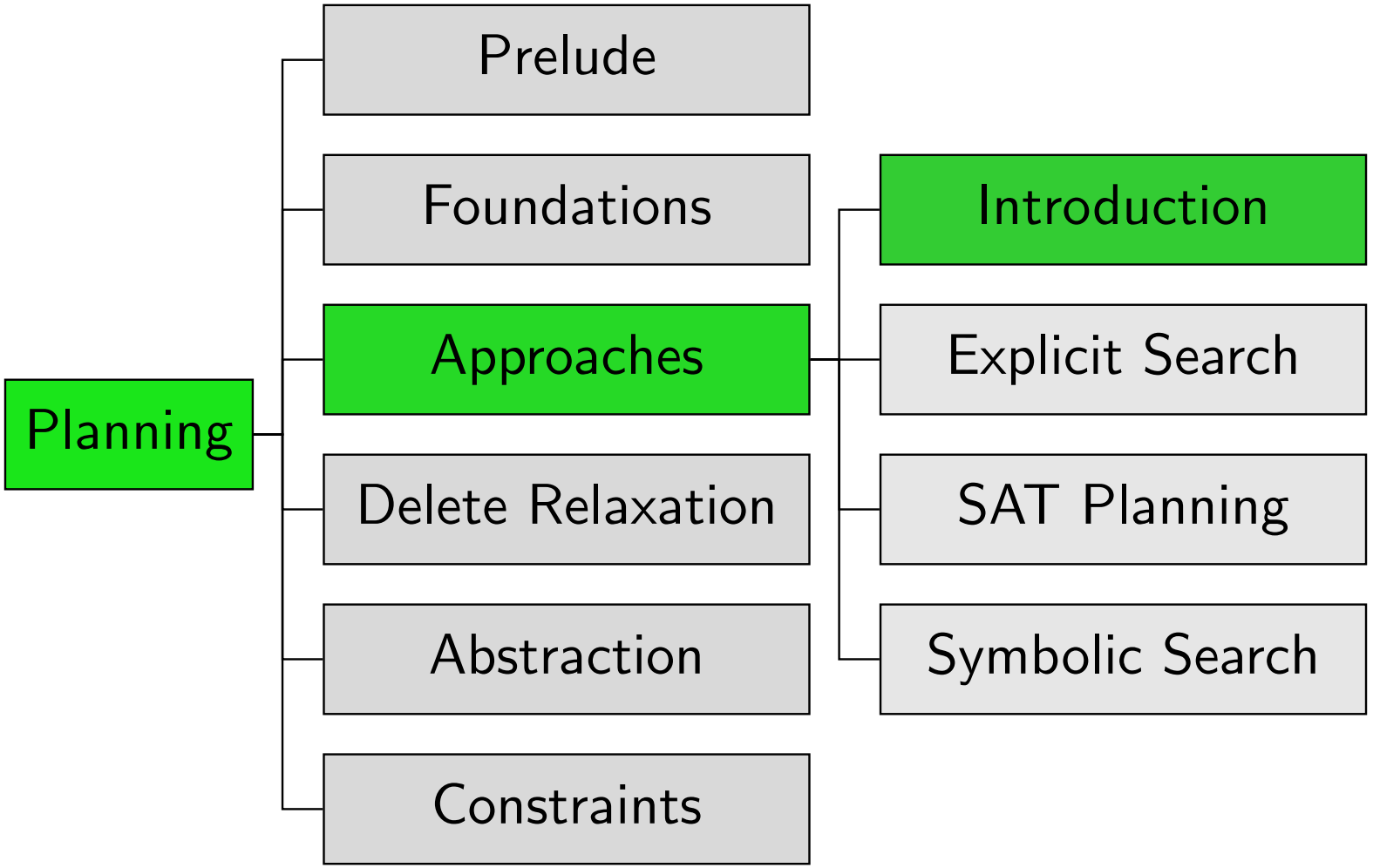
C1. Overview of Classical Planning Algorithms (Part 1)

Malte Helmert and Gabriele Röger

Universität Basel

October 6, 2025

Content of the Course



The Big Three

Classical Planning Algorithms

Let's start solving planning tasks!

This Chapter and the Next

very high-level overview of classical planning algorithms

- bird's eye view: no details, just some very brief ideas

The Big Three

Of the many planning approaches, three techniques stand out:

- explicit search ⇝ Chapters C3–C4, Parts D–F
- SAT planning ⇝ Chapters C5–C6
- symbolic search ⇝ Chapters C7–C8

also: many algorithm portfolios

Satisficing or Optimal Planning?

must carefully distinguish:

- satisficing planning: any plan is OK (cheaper ones preferred)
- optimal planning: plans must have minimum cost

solved by similar techniques, but:

- details very different
- almost no overlap between best techniques for satisficing planning and best techniques for optimal planning
- many tasks that are trivial for satisficing planners are impossibly hard for optimal planners

Explicit Search

Explicit Search

You know this one already! (Hopefully.)

Reminder: State-Space Search

Need to Catch Up?

- We assume prior knowledge of basic search algorithms:
  - uninformed vs. informed (heuristic)
  - satisficing vs. optimal
  - heuristics and their properties
  - specific algorithms: e.g., breadth-first search, greedy best-first search, A*
- If you are not familiar with them, we recommend Part B of the Foundations of Artificial Intelligence course:
  `https://dmi.unibas.ch/en/studium/computer-science-informatik/lehrangebot-fs25/13548-lecture-foundations-of-artificial-intelligence/`

Reminder: Interface for Heuristic Search Algorithms

Abstract Interface Needed for Heuristic Search Algorithms

- init() $\rightsquigarrow$ returns initial state
- is_goal(s) $\rightsquigarrow$ tests if s is a goal state
- succ(s) $\rightsquigarrow$ returns all pairs $\langle a, s' \rangle$ with $s \xrightarrow{a} s'$
- cost(a) $\rightsquigarrow$ returns cost of action a
- h(s) $\rightsquigarrow$ returns heuristic value for state s

$\rightsquigarrow$ Foundations of Artificial Intelligence course, Chap. B2 and B9

State Space vs. Search Space

- Planning tasks induce transition systems (a.k.a. state spaces) with an initial state, labeled transitions and goal states.
- State-space search searches state spaces with an initial state, a successor function and goal states.

$\rightsquigarrow$ looks like an obvious correspondence

- However, in planning as search, the state space being searched can be different from the state space of the planning task.
- When we need to make a distinction, we speak of
  - the state space of the planning task whose states are called world states vs.
  - the search space of the search algorithm whose states are called search states.

Design Choice: Search Direction

How to apply explicit search to planning? $\rightsquigarrow$ many design choices!

Design Choice: Search Direction

- progression: forward from initial state to goal
- regression: backward from goal states to initial state
- bidirectional search

$\rightsquigarrow$ Chapters C3–C4

Design Choice: Search Algorithm

How to apply explicit search to planning? $\rightsquigarrow$ many design choices!

Design Choice: Search Algorithm

- uninformed search:
  depth-first, breadth-first, iterative depth-first, ...
- heuristic search (systematic):
  greedy best-first, A*, weighted A*, IDA*, ...
- heuristic search (local):
  hill-climbing, simulated annealing, beam search, ...

Design Choice: Search Control

How to apply explicit search to planning? $\rightsquigarrow$ many design choices!

Design Choice: Search Control

- heuristics for informed search algorithms
- pruning techniques: invariants, symmetry elimination, partial-order reduction, helpful actions pruning, ...

How do we find good heuristics in a domain-independent way?

$\rightsquigarrow$ one of the main focus areas of classical planning research

$\rightsquigarrow$ Parts D–F

Summary

Summary

(Joint summary follows after next chapter.)

Planning and Optimization

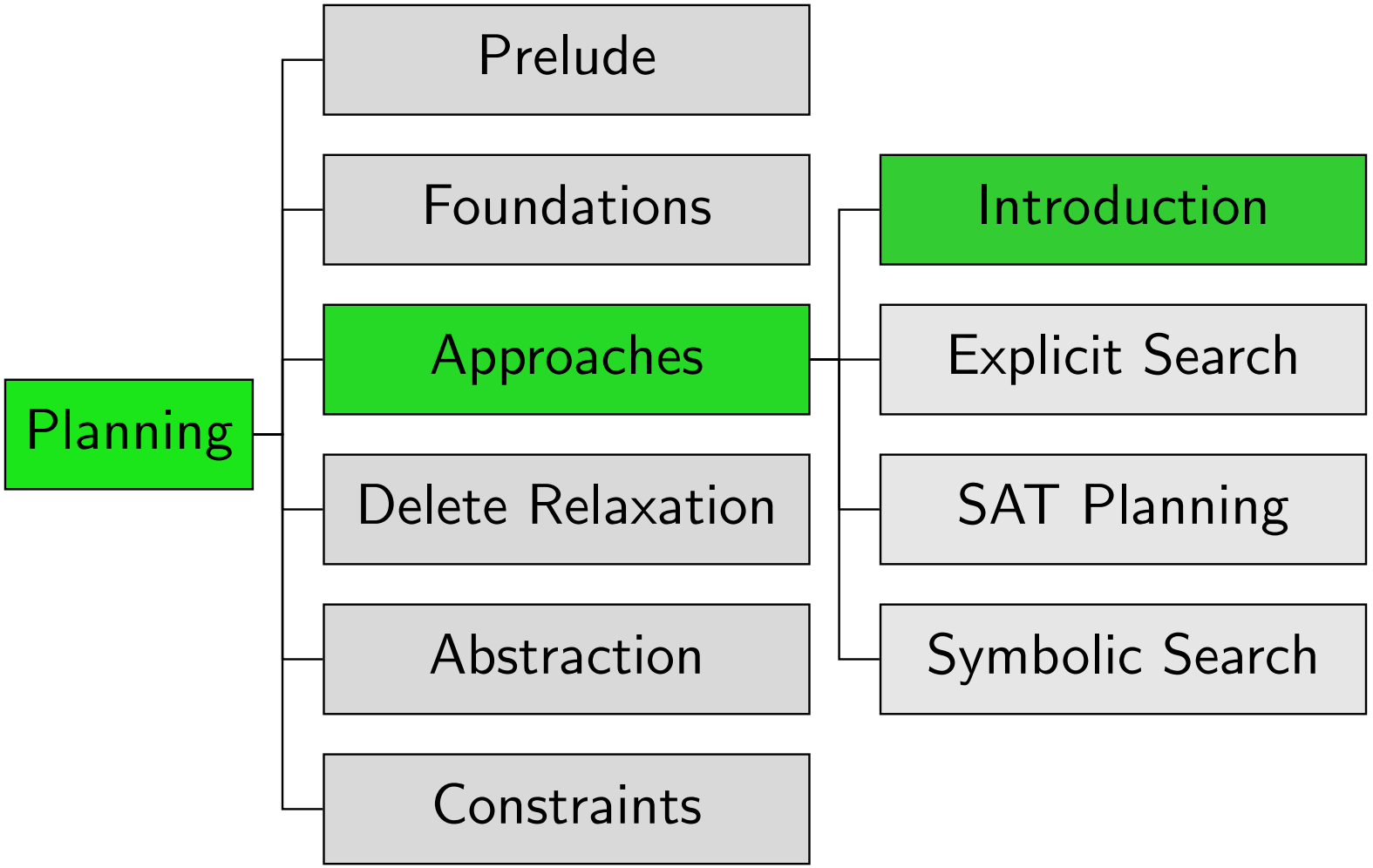
C2. Overview of Classical Planning Algorithms (Part 2)

Malte Helmert and Gabriele Röger

Universität Basel

October 6, 2025

Content of the Course



The Big Three (Repeated from Last Chapter)

Of the many planning approaches, three techniques stand out:

- explicit search ⇝ Chapters C3–C4, Parts D–F
- SAT planning ⇝ Chapters C5–C6
- symbolic search ⇝ Chapters C7–C8

also: many algorithm portfolios

SAT Planning

SAT Planning: Basic Idea

- formalize problem of finding plan with a given horizon (length bound) as a propositional satisfiability problem and feed it to a generic SAT solver
- to obtain a (semi-) complete algorithm, try with increasing horizons until a plan is found (= the formula is satisfiable)
- important optimization: allow applying several non-conflicting operators "at the same time" so that a shorter horizon suffices

SAT Encodings: Variables

- given propositional planning task $\langle V, I, O, \gamma \rangle$
- given horizon $T \in \mathbb{N}_0$

Variables of SAT Encoding

- propositional variables v^i for all $v \in V$, $0 \leq i \leq T$
  encode state after i steps of the plan
- propositional variables o^i for all $o \in O$, $1 \leq i \leq T$
  encode operator(s) applied in i-th step of the plan

Design Choice: SAT Encoding

Again, there are several important design choices.

Design Choice: SAT Encoding

- sequential or parallel
- many ways of modeling planning semantics in logic

$\rightsquigarrow$ main focus of research on SAT planning

Design Choice: SAT Solver

Again, there are several important design choices.

Design Choice: SAT Solver

- out-of-the-box like Glucose, CaDiCal, MiniSAT
- planning-specific modifications

Design Choice: Evaluation Strategy

Again, there are several important design choices.

Design Choice: Evaluation Strategy

- always advance horizon by $+1$ or more aggressively
- possibly probe multiple horizons concurrently

Symbolic Search

Symbolic Search Planning: Basic Ideas

- search processes sets of states at a time
- operators, goal states, state sets reachable with a given cost etc. represented by binary decision diagrams (BDDs) (or similar data structures)
- hope: exponentially large state sets can be represented as polynomially sized BDDs, which can be efficiently processed
- perform symbolic breadth-first search (or something more sophisticated) on these set representations

Symbolic Breadth-First Progression Search

prototypical algorithm:

Symbolic Breadth-First Progression Search

def bfs-progression(V, I, O, γ):
    $goal\_states := models(\gamma)$
    $reached_0 := \{I\}$
    $i := 0$
    loop:
        if $reached_i \cap goal\_states \neq \emptyset$:
            return solution found
        $reached_{i+1} := reached_i \cup apply(reached_i, O)$
        if $reached_{i+1} = reached_i$:
            return no solution exists
        $i := i + 1$

Symbolic Breadth-First Progression Search

prototypical algorithm:

Symbolic Breadth-First Progression Search

def bfs-progression(V, I, O, γ):
    $goal\_states := models(\gamma)$
    $reached_0 := \{I\}$
    $i := 0$
    loop:
        if $reached_i \cap goal\_states \neq \emptyset$:
            return solution found
        $reached_{i+1} := reached_i \cup apply(reached_i, O)$
        if $reached_{i+1} = reached_i$:
            return no solution exists
        $i := i + 1$

$\rightsquigarrow$ If we can implement operations $models$, $\{I\}$, $\cap$, $\neq \emptyset$, $\cup$, $apply$ and $=$ efficiently, this is a reasonable algorithm.

Design Choice: Symbolic Data Structure

Again, there are several important design choices.

Design Choice: Symbolic Data Structure

- BDDs
- ADDs
- EVMDDs
- SDDs

Other Design Choices

- additionally, same design choices as for explicit search:
  - search direction
  - search algorithm
  - search control (incl. heuristics)
- in practice, hard to make heuristics and other advanced search control efficient for symbolic search
  $\rightsquigarrow$ rarely used

Planning System Examples

Planning Systems: FF

FF (Hoffmann & Nebel, 2001)

- problem class: satisficing
- algorithm class: explicit search
- search direction: forward search
- search algorithm: enforced hill-climbing
- heuristic: FF heuristic (inadmissible)
- other aspects: helpful action pruning; goal agenda manager

$\rightsquigarrow$ breakthrough for heuristic search planning;
winner of IPC 2000

Planning Systems: LAMA

LAMA (Richter & Westphal, 2008)

- problem class: satisficing
- algorithm class: explicit search
- search direction: forward search
- search algorithm: restarting Weighted A* (anytime)
- heuristic: FF heuristic and landmark heuristic (inadmissible)
- other aspects: preferred operators; deferred heuristic evaluation; multi-queue search

$\rightsquigarrow$ still one of the leading satisficing planners;
winner of IPC 2008 and IPC 2011 (satisficing tracks)

Planning Systems: Madagascar-pC

Madagascar (Rintanen, 2014)

- problem class: satisficing
- algorithm class: SAT planning
- encoding: parallel $\exists$-step encoding
- SAT solver: using planning-specific action variable selection
- evaluation strategy: exponential horizons, parallelized probing
- other aspects: invariants

$\rightsquigarrow$ second place at IPC 2014 (agile track)

Planning Systems: SymBA*

SymBA* (Torralba, 2015)

- problem class: optimal
- algorithm class: symbolic search
- symbolic data structure: BDDs
- search direction: bidirectional
- search algorithm: mixture of (symbolic) Dijkstra and A*
- heuristic: perimeter abstractions/blind

$\rightsquigarrow$ winner of IPC 2014 (optimal track)

Planning Systems: Scorpion

Scorpion 2023 (Seipp, 2023)

- problem class: optimal
- algorithm class: explicit search
- search direction: forward search
- search algorithm: A^*
- heuristic: abstraction heuristics and cost partitioning

$\rightsquigarrow$ runner-up of IPC 2023 (optimal track)

Planning Systems: Fast Downward Stone Soup

Fast Downward Stone Soup 2023, optimal version (Büchner et al., 2023)

- problem class: optimal
- algorithm class: (portfolio of) explicit search
- search direction: forward search
- search algorithm: A^*
- heuristic: all admissible heuristics considered in the course

$\rightsquigarrow$ winner of IPC 2011 (optimal track);
various awards in IPC 2011–2023

Planning Systems: SymK

SymK (Speck et al., 2025)

- problem class: optimal
- algorithm class: symbolic search
- symbolic data structure: BDDs
- search direction: bidirectional
- search algorithm: symbolic Dijkstra algorithm
- heuristic: blind

Summary

Summary

big three classes of algorithms for classical planning:

- explicit search
  - design choices: search direction, search algorithm, search control (incl. heuristics)
- SAT planning
  - design choices: SAT encoding, SAT solver, evaluation strategy
- symbolic search
  - design choices: symbolic data structure + same ones as for explicit search

Planning and Optimization

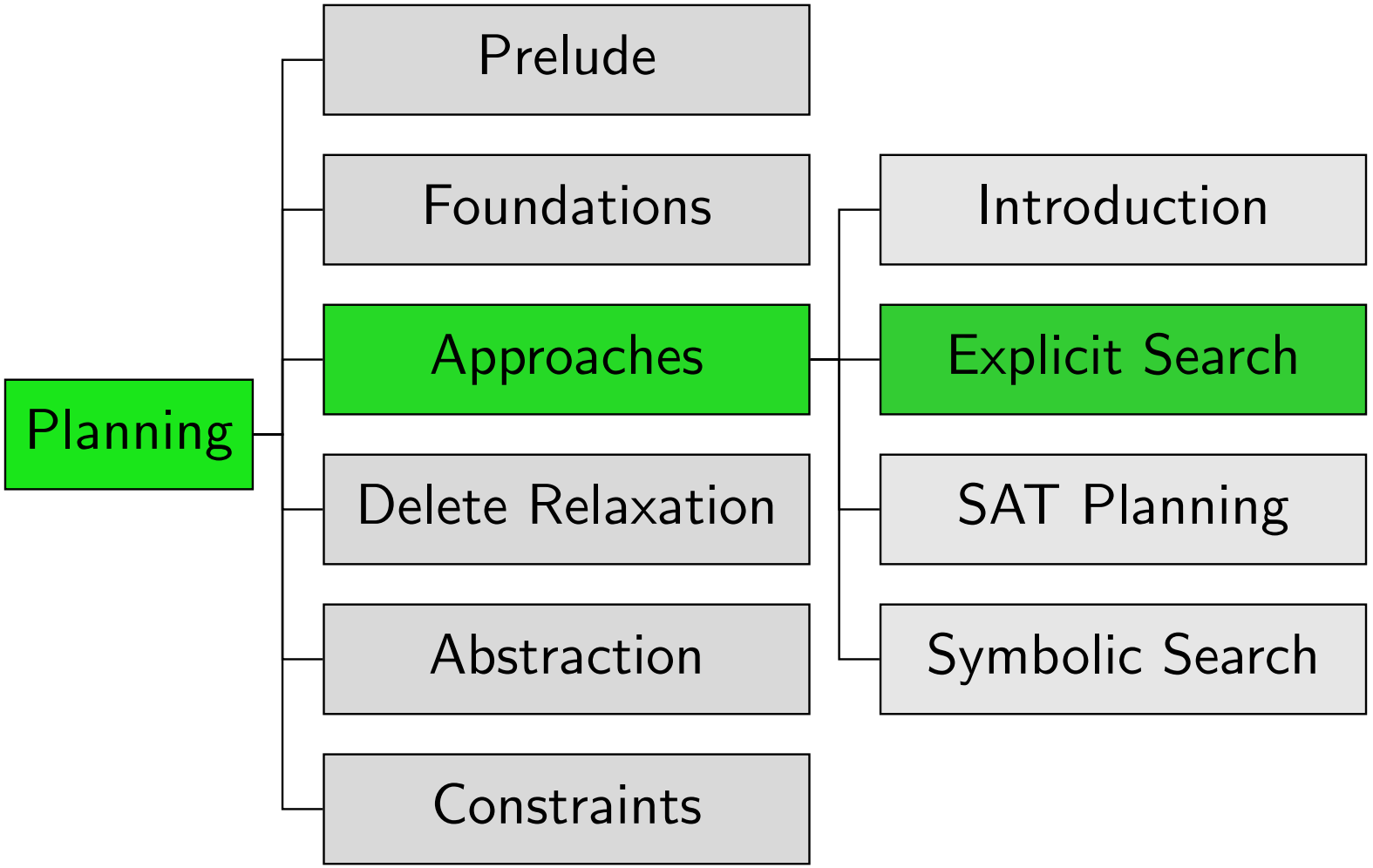
C3. Progression and Regression Search

Malte Helmert and Gabriele Röger

Universität Basel

October 8, 2025

Content of the Course



Introduction

Search Direction

Search direction

- one dimension for classifying search algorithms
- forward search from initial state to goal based on progression
- backward search from goal to initial state based on regression
- bidirectional search

In this chapter we look into progression and regression planning.

Reminder: Interface for Heuristic Search Algorithms

Abstract Interface Needed for Heuristic Search Algorithms

- init() $\rightsquigarrow$ returns initial state
- is_goal(s) $\rightsquigarrow$ tests if s is a goal state
- succ(s) $\rightsquigarrow$ returns all pairs $\langle a, s' \rangle$ with $s \xrightarrow{a} s'$
- cost(a) $\rightsquigarrow$ returns cost of action a
- h(s) $\rightsquigarrow$ returns heuristic value for state s

Progression

Planning by Forward Search: Progression

Progression: Computing the successor state $s[\![o]\!]$ of a state s with respect to an operator o.

Progression planners find solutions by forward search:

- start from initial state
- iteratively pick a previously generated state and progress it through an operator, generating a new state
- solution found when a goal state generated

pro: very easy and efficient to implement

Search Space for Progression

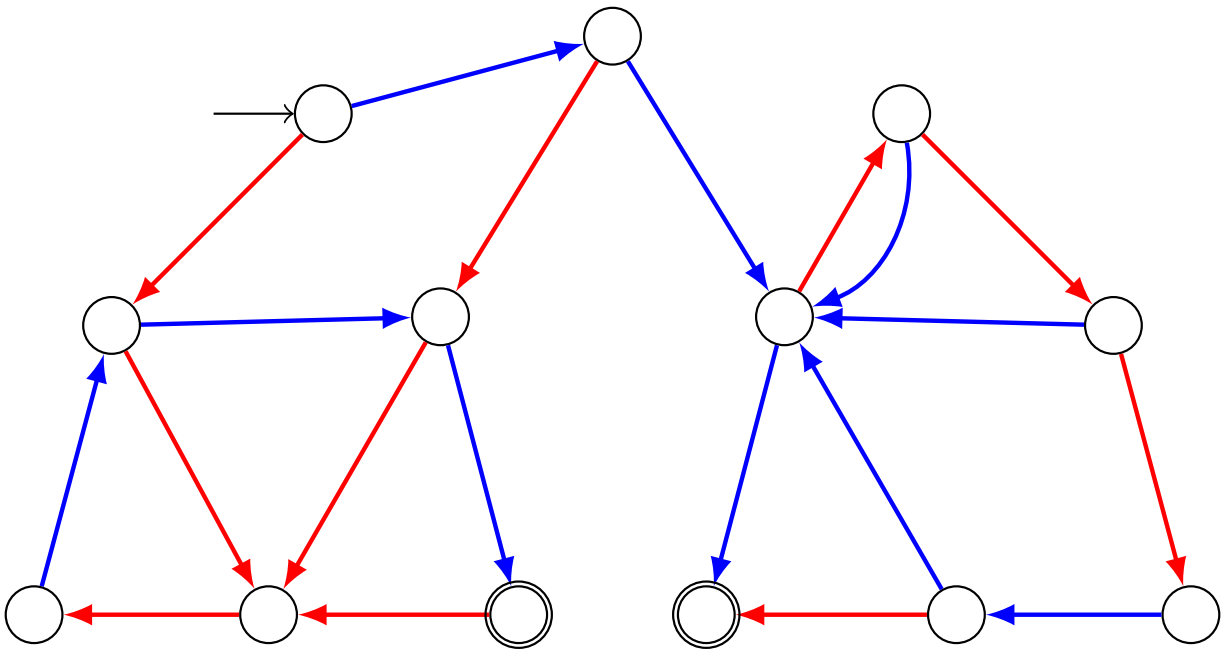
Search Space for Progression

search space for progression in a planning task $\Pi = \langle V, I, O, \gamma \rangle$
(search states are world states s of Π;
actions of search space are operators $o \in O$)

- init() $\rightsquigarrow$ returns I
- is_goal(s) $\rightsquigarrow$ tests if $s \models \gamma$
- succ(s) $\rightsquigarrow$ returns all pairs $\langle o, s[\![o]\!] \rangle$
  where $o \in O$ and o is applicable in s
- cost(o) $\rightsquigarrow$ returns $cost(o)$ as defined in Π
- h(s) $\rightsquigarrow$ estimates cost from s to γ ($\rightsquigarrow$ Parts D–F)

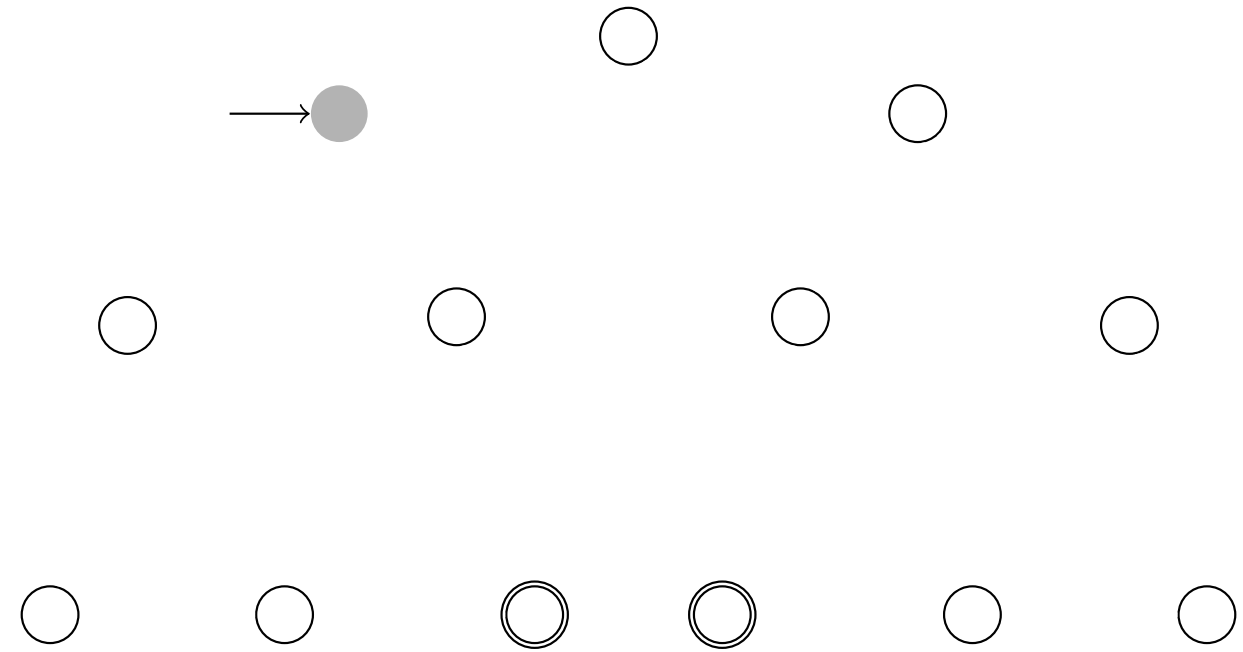
Progression Planning Example

Example of a progression search



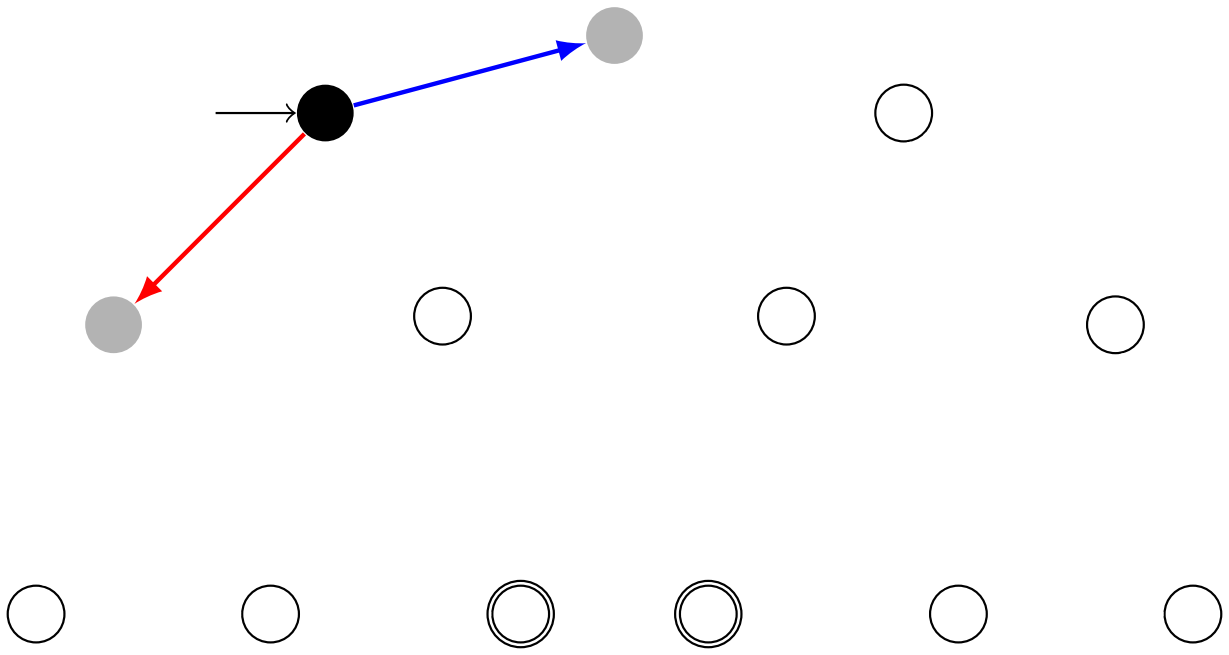
Progression Planning Example

Example of a progression search



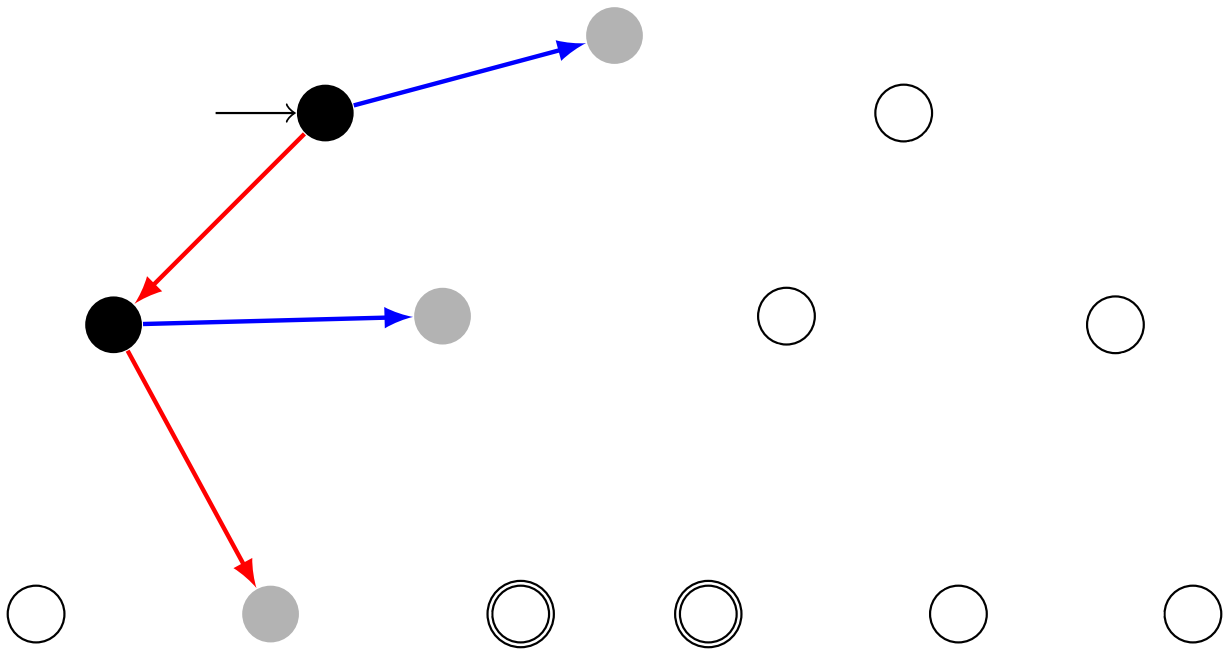
Progression Planning Example

Example of a progression search



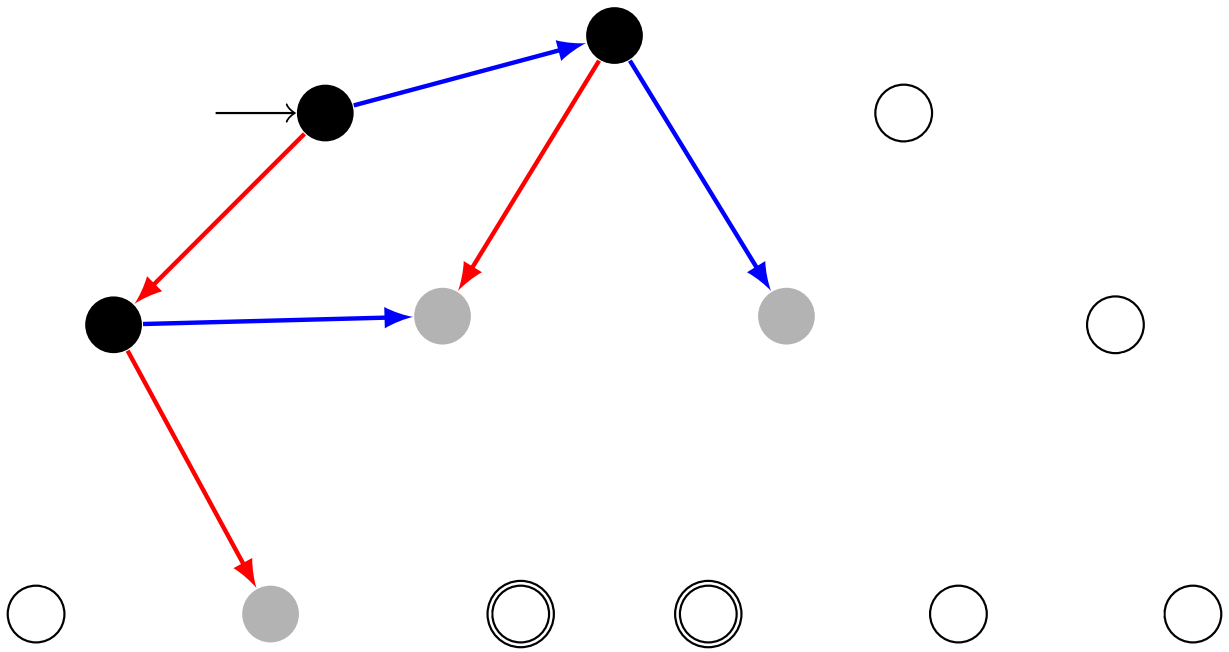
Progression Planning Example

Example of a progression search



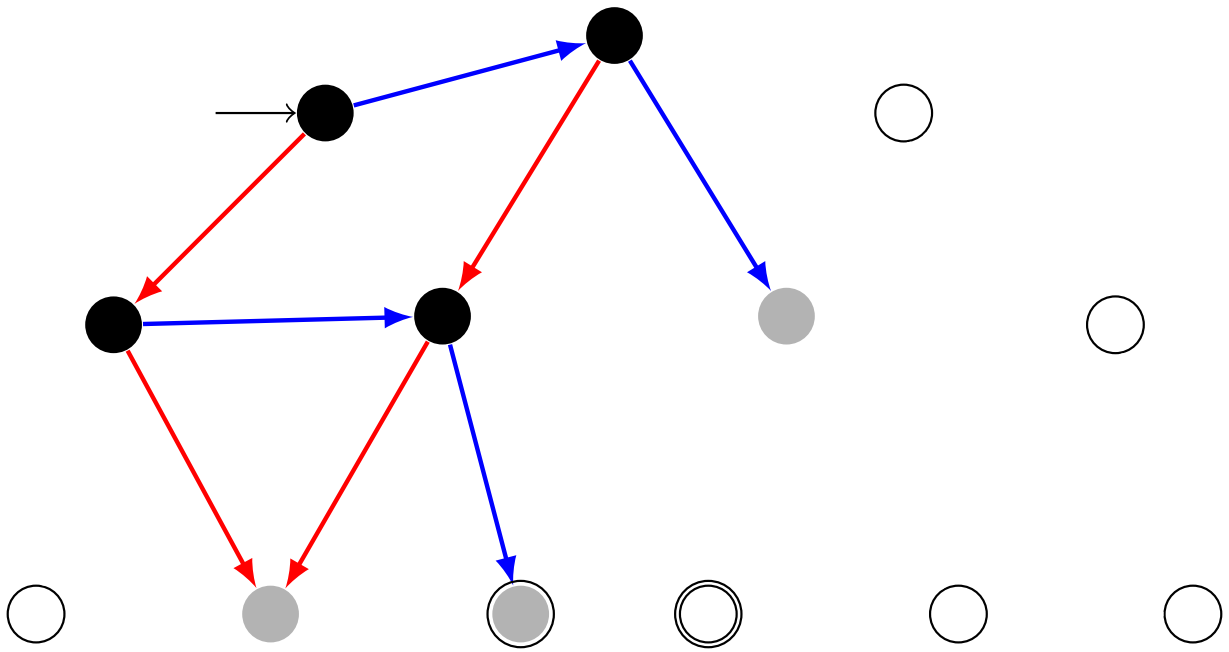
Progression Planning Example

Example of a progression search



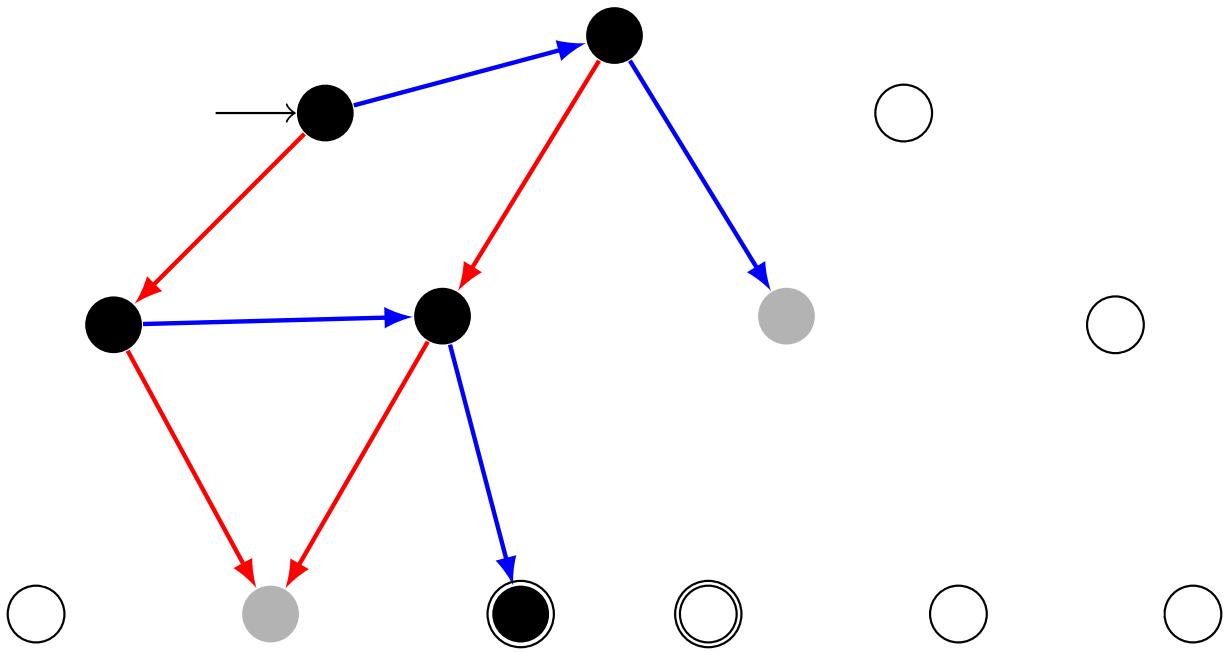
Progression Planning Example

Example of a progression search



Progression Planning Example

Example of a progression search



Regression

Forward Search vs. Backward Search

Searching planning tasks in forward vs. backward direction is not symmetric:

- forward search starts from a single initial state;
  backward search starts from a set of goal states
- when applying an operator o in a state s in forward direction,
  there is a unique successor state s';
  if we just applied operator o and ended up in state s',
  there can be several possible predecessor states s

$\rightsquigarrow$ in most natural representation for backward search in planning, each search state corresponds to a set of world states

Planning by Backward Search: Regression

Regression: Computing the possible predecessor states $regr(S', o)$ of a set of states S' ("subgoal") given the last operator o that was applied.

$\rightsquigarrow$ formal definition in next chapter

Regression planners find solutions by backward search:

- start from set of goal states
- iteratively pick a previously generated subgoal (state set) and regress it through an operator, generating a new subgoal
- solution found when a generated subgoal includes initial state

pro: can handle many states simultaneously
con: basic operations complicated and expensive

Search Space Representation in Regression Planners

identify state sets with logical formulas (again):

- each search state corresponds to a set of world states ("subgoal")
- each search state is represented by a logical formula: φ represents $\{s \in S \mid s \models \varphi\}$
- many basic search operations like detecting duplicates are NP-complete or coNP-complete

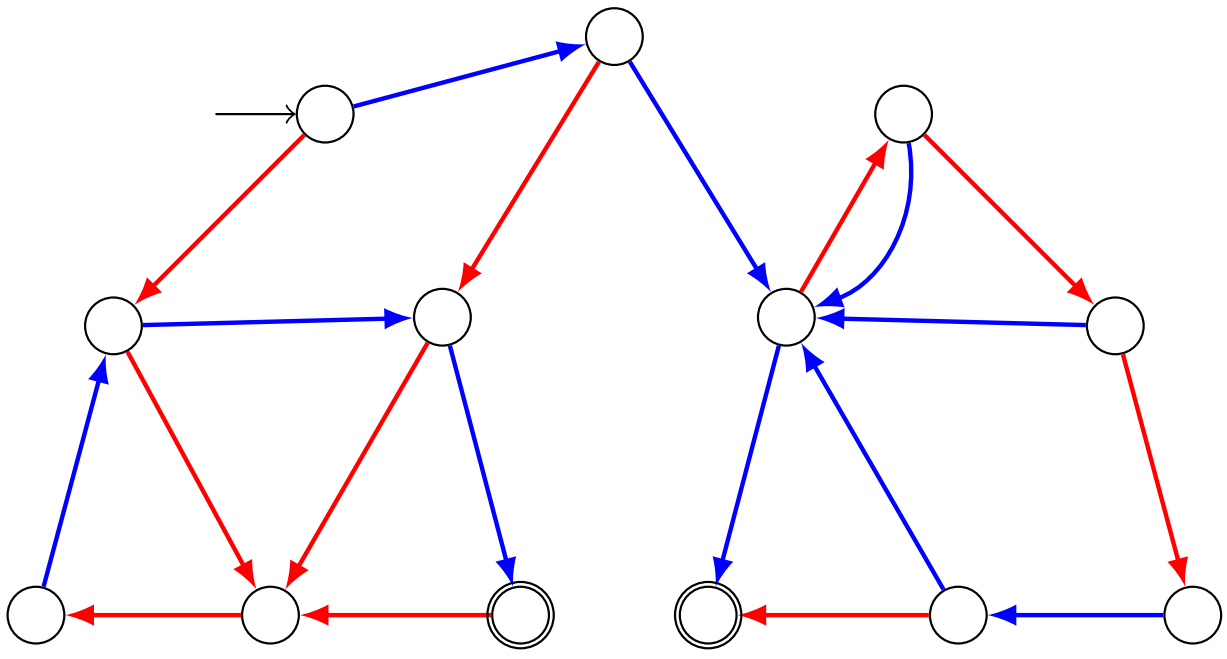
Search Space for Regression

Search Space for Regression

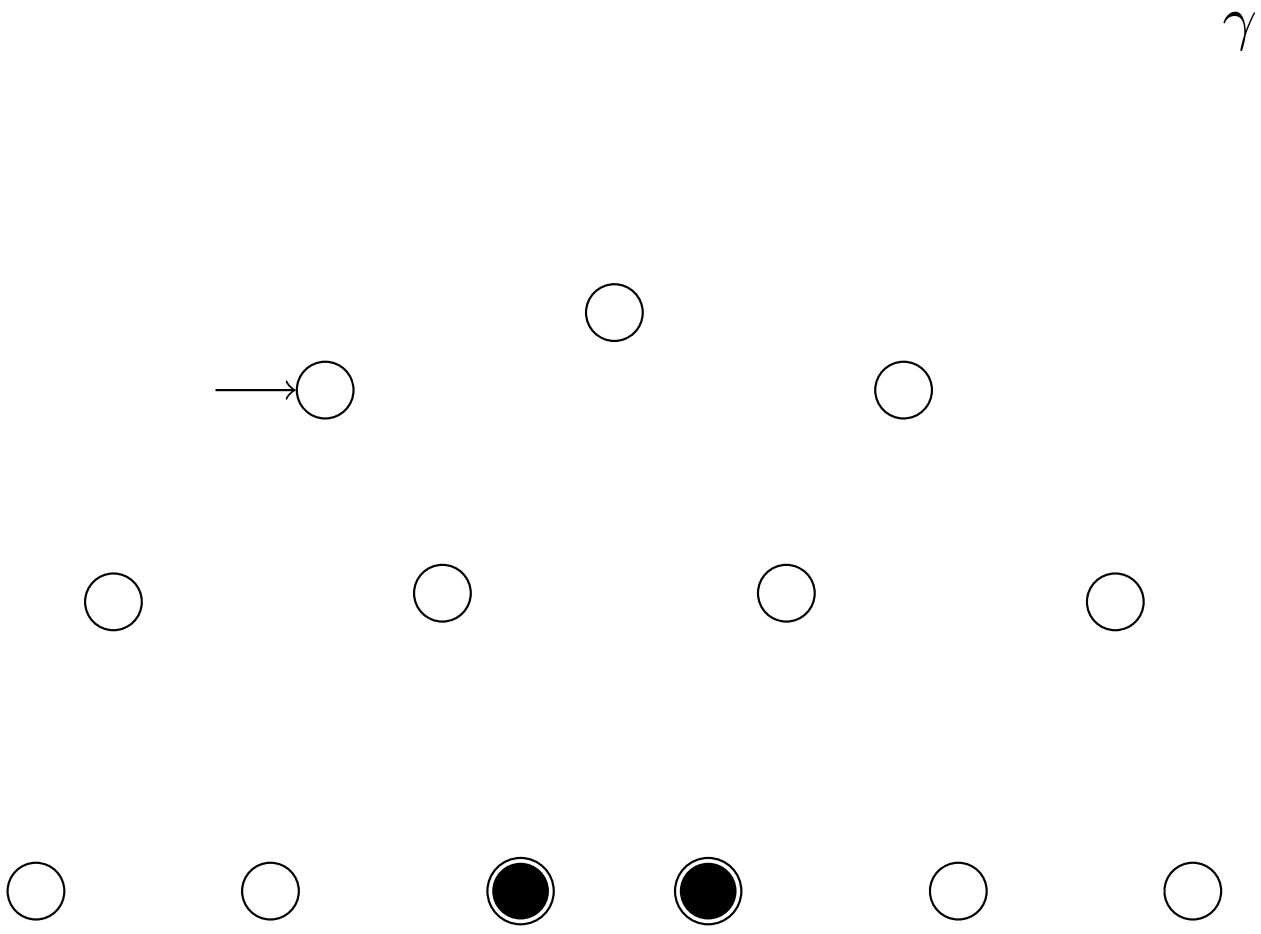
search space for regression in a planning task $\Pi = \langle V, I, O, \gamma \rangle$
(search states are formulas φ describing sets of world states;
actions of search space are operators $o \in O$)

- init() $\rightsquigarrow$ returns γ
- is_goal(φ) $\rightsquigarrow$ tests if $I \models \varphi$
- succ(φ) $\rightsquigarrow$ returns all pairs $\langle o, \mathit{regr}(\varphi, o) \rangle$ where $o \in O$ and $\mathit{regr}(\varphi, o)$ is defined
- cost(o) $\rightsquigarrow$ returns $\mathit{cost}(o)$ as defined in Π
- h(φ) $\rightsquigarrow$ estimates cost from I to φ ($\rightsquigarrow$ Parts D–F)

Regression Planning Example (Depth-first Search)

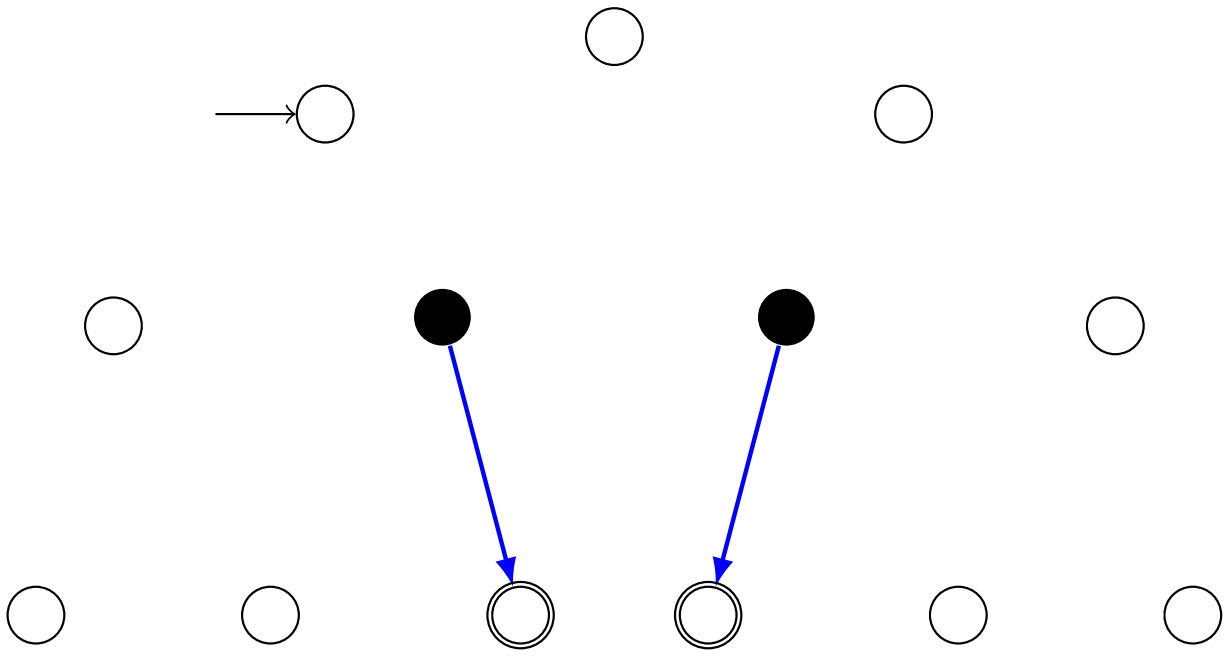


Regression Planning Example (Depth-first Search)



Regression Planning Example (Depth-first Search)

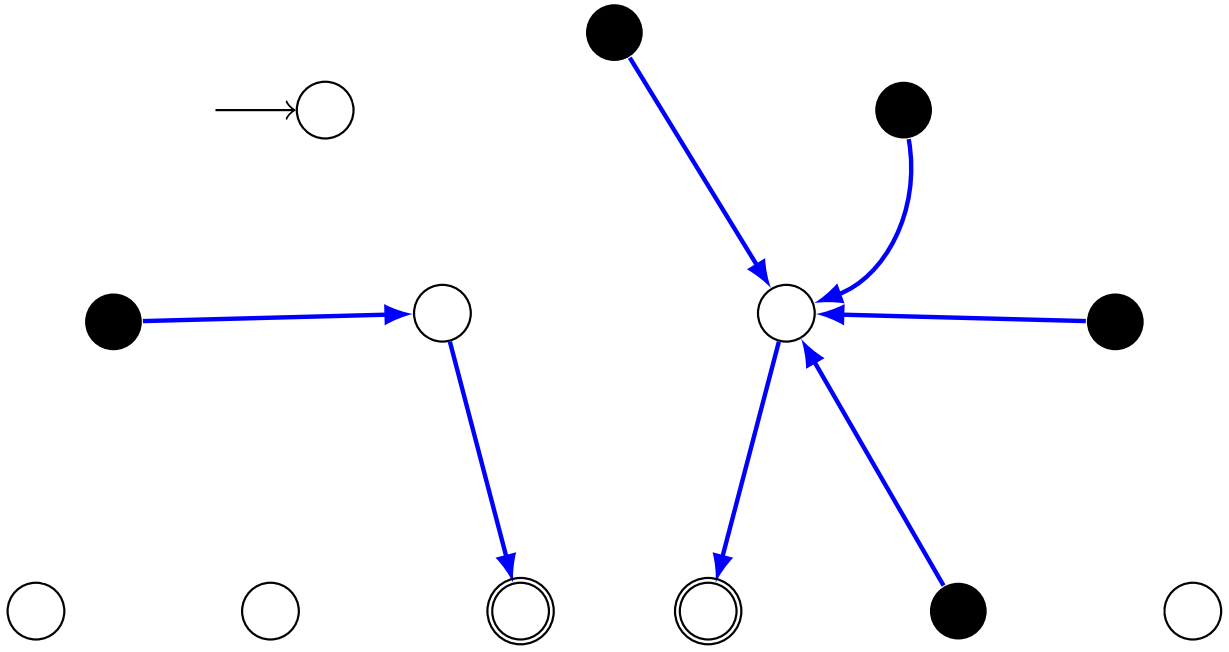
$\varphi_1 = \mathit{regr}(\gamma, \longrightarrow)$ $\qquad$ $\varphi_1 \longrightarrow \gamma$



Regression Planning Example (Depth-first Search)

$\varphi_1 = \mathit{regr}(\gamma, \longrightarrow)$
$\varphi_2 = \mathit{regr}(\varphi_1, \longrightarrow)$

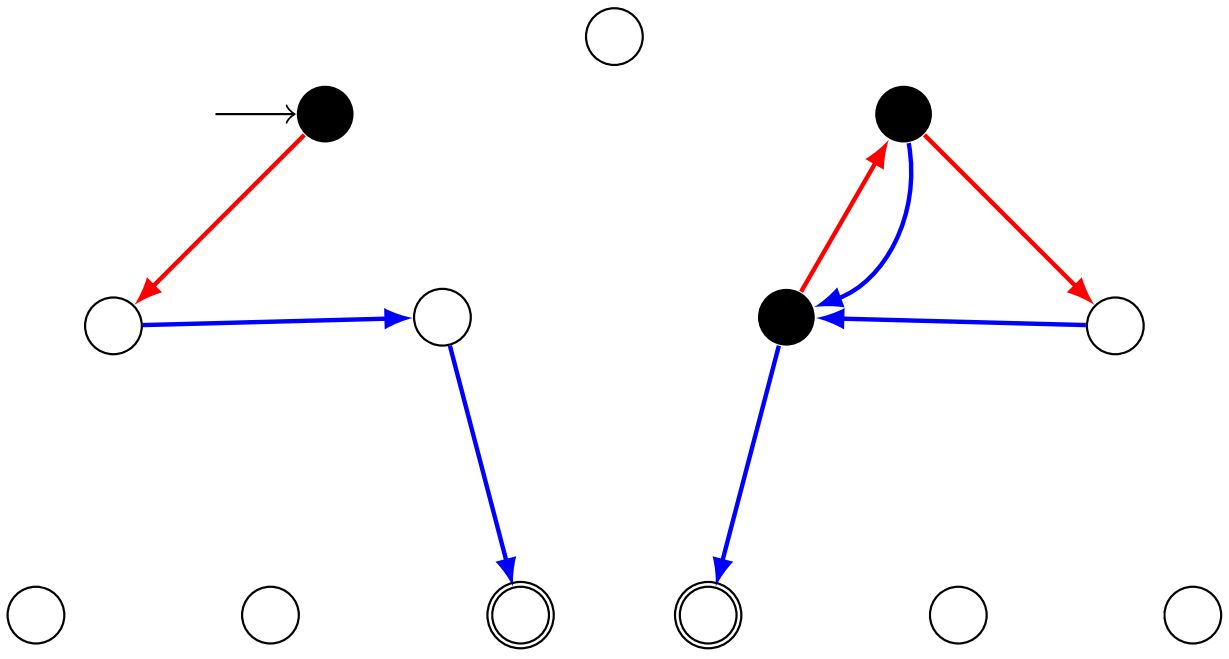
$\varphi_2 \longrightarrow \varphi_1 \longrightarrow \gamma$



Regression Planning Example (Depth-first Search)

$\varphi_1 = \mathit{regr}(\gamma, \longrightarrow)$
$\varphi_2 = \mathit{regr}(\varphi_1, \longrightarrow)$
$\varphi_3 = \mathit{regr}(\varphi_2, \longrightarrow), I \models \varphi_3$

$\varphi_3 \longrightarrow \varphi_2 \longrightarrow \varphi_1 \longrightarrow \gamma$



Regression for STRIPS Tasks

Regression for STRIPS Planning Tasks

Regression for STRIPS planning tasks is much simpler than the general case:

- Consider subgoal φ that is conjunction of atoms $a_1 \wedge \cdots \wedge a_n$ (e.g., the original goal γ of the planning task).
- First step: Choose an operator o that deletes no a_i.
- Second step: Remove any atoms added by o from φ.
- Third step: Conjoin $pre(o)$ to φ.

$\rightsquigarrow$ Outcome of this is regression of φ w.r.t. o.
It is again a conjunction of atoms.

optimization: only consider operators adding at least one a_i

STRIPS Regression

Definition (STRIPS Regression)

Let $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_n$ be a conjunction of atoms, and let o be a STRIPS operator which adds the atoms $a_1, \ldots, a_k$ and deletes the atoms $d_1, \ldots, d_l$.

The STRIPS regression of φ with respect to o is

$$sregr(\varphi, o) := \begin{cases} \bot & \text{if } \varphi_i = d_j \text{ for some } i, j \\ pre(o) \wedge \bigwedge(\{\varphi_1, \ldots, \varphi_n\} \setminus \{a_1, \ldots, a_k\}) & \text{else} \end{cases}$$

Note: $sregr(\varphi, o)$ is again a conjunction of atoms, or $\bot$.

Does this Capture the Idea of Regression?

For our definition to capture the concept of regression,
it must have the following property:

Regression Property

For all sets of states described by a conjunction of atoms φ,
all states s and all STRIPS operators o,

$$s \models \mathit{sregr}(\varphi, o) \quad \text{iff} \quad s[\![o]\!] \models \varphi.$$

This is indeed true. We do not prove it now because we prove this property for general regression (not just STRIPS) later.

Summary

Summary

- Progression search proceeds forward from the initial state.
- In progression search, the search space is identical to the state space of the planning task.
- Regression search proceeds backwards from the goal.
- Each search state corresponds to a set of world states, for example represented by a formula.
- Regression is simple for STRIPS operators.
- The theory for general regression is more complex. This is the topic of the following chapter.

Planning and Optimization

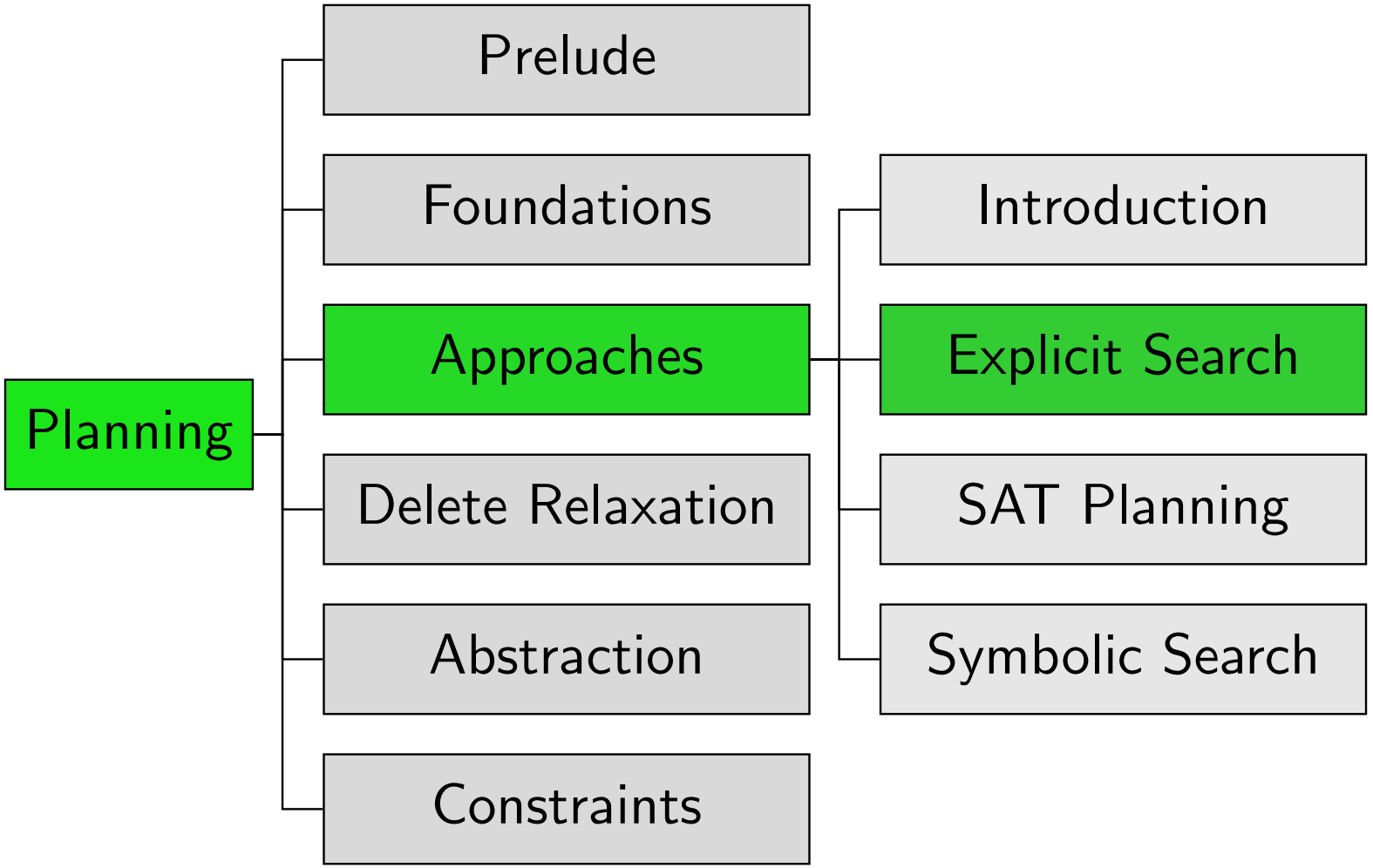
C4. General Regression

Malte Helmert and Gabriele Röger

Universität Basel

October 8, 2025

Content of the Course



Regression for General Planning Tasks

- With disjunctions and conditional effects, things become more tricky. How to regress $a \vee (b \wedge c)$ with respect to $\langle q, d \rhd b \rangle$?
- In this chapter, we show how to regress general sets of states through general operators.
- We extensively use the idea of representing sets of states as formulas.

Regressing State Variables

Regressing State Variables: Motivation

Key question for general regression:

- Assume we are applying an operator with effect e.
- What must be true in the predecessor state for propositional state variable v to be true in the successor state?

If we can answer this question, a general definition of regression is only a small additional step.

Regressing State Variables: Key Idea

Assume we are in state s and apply effect e to obtain successor state s'.

Propositional state variable v is true in s' iff

- effect e makes it true, or
- it remains true, i.e., it is true in s and not made false by e.

Regressing a State Variable Through an Effect

Definition (Regressing a State Variable Through an Effect)

Let e be an effect of a propositional planning task,
and let v be a propositional state variable.

The regression of v through e, written $regr(v, e)$,
is defined as the following logical formula:

$$regr(v, e) = effcond(v, e) \vee (v \wedge \neg effcond(\neg v, e)).$$

Does this capture add-after-delete semantics correctly?

Regressing State Variables: Example

Example

Let $e = (b \rhd a) \land (c \rhd \neg a) \land b \land \neg d$.

v	$effcond(v, e)$	$effcond(\neg v, e)$	$regr(v, e)$
a	b	c	$b \lor (a \land \neg c)$
b	$\top$	$\bot$	$\top \lor (b \land \neg \bot) \equiv \top$
c	$\bot$	$\bot$	$\bot \lor (c \land \neg \bot) \equiv c$
d	$\bot$	$\top$	$\bot \lor (d \land \neg \top) \equiv \bot$

Reminder: $regr(v, e) = effcond(v, e) \lor (v \land \neg effcond(\neg v, e))$

Regressing State Variables: Correctness (1)

Lemma (Correctness of $regr(v, e)$)

Let s be a state, e be an effect and v be a state variable of a propositional planning task.

Then $s \models regr(v, e)$ iff $s[\![e]\!] \models v$.

Regressing State Variables: Correctness (2)

Proof.

($\Rightarrow$): We know $s \models \mathit{regr}(v, e)$, and hence $s \models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.

Do a case analysis on the two disjuncts.

Regressing State Variables: Correctness (2)

Proof.

($\Rightarrow$): We know $s \models \mathit{regr}(v, e)$, and hence
$s \models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.

Do a case analysis on the two disjuncts.

Case 1: $s \models \mathit{effcond}(v, e)$.
Then $s[\![e]\!] \models v$ by the first case in the definition of $s[\![e]\!]$ (Ch. B3).

Regressing State Variables: Correctness (2)

Proof.

($\Rightarrow$): We know $s \models \mathit{regr}(v, e)$, and hence
$s \models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.

Do a case analysis on the two disjuncts.

Case 1: $s \models \mathit{effcond}(v, e)$.
Then $s[\![e]\!] \models v$ by the first case in the definition of $s[\![e]\!]$ (Ch. B3).

Case 2: $s \models (v \wedge \neg \mathit{effcond}(\neg v, e))$.
Then $s \models v$ and $s \not\models \mathit{effcond}(\neg v, e)$.
We may additionally assume $s \not\models \mathit{effcond}(v, e)$
because otherwise we can apply Case 1 of this proof.
Then $s[\![e]\!] \models v$ by the third case in the definition of $s[\![e]\!]$. ...

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

- By prerequisite, $s \not\models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

- By prerequisite, $s \not\models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.
- Hence $s \models \neg \mathit{effcond}(v, e) \wedge (\neg v \vee \mathit{effcond}(\neg v, e))$.

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

- By prerequisite, $s \not\models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.
- Hence $s \models \neg \mathit{effcond}(v, e) \wedge (\neg v \vee \mathit{effcond}(\neg v, e))$.
- From the first conjunct, we get $s \models \neg \mathit{effcond}(v, e)$ and hence $s \not\models \mathit{effcond}(v, e)$.

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

- By prerequisite, $s \not\models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.
- Hence $s \models \neg \mathit{effcond}(v, e) \wedge (\neg v \vee \mathit{effcond}(\neg v, e))$.
- From the first conjunct, we get $s \models \neg \mathit{effcond}(v, e)$ and hence $s \not\models \mathit{effcond}(v, e)$.
- From the second conjunct, we get $s \models \neg v \vee \mathit{effcond}(\neg v, e)$.

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

- By prerequisite, $s \not\models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.
- Hence $s \models \neg \mathit{effcond}(v, e) \wedge (\neg v \vee \mathit{effcond}(\neg v, e))$.
- From the first conjunct, we get $s \models \neg \mathit{effcond}(v, e)$ and hence $s \not\models \mathit{effcond}(v, e)$.
- From the second conjunct, we get $s \models \neg v \vee \mathit{effcond}(\neg v, e)$.
- Case 1: $s \models \neg v$. Then v is false before applying e and remains false, so $s[\![e]\!] \not\models v$.

Regressing State Variables: Correctness (3)

Proof (continued).

($\Leftarrow$): Proof by contraposition.

We show that if $\mathit{regr}(v, e)$ is false in s, then v is false in $s[\![e]\!]$.

- By prerequisite, $s \not\models \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e))$.
- Hence $s \models \neg \mathit{effcond}(v, e) \wedge (\neg v \vee \mathit{effcond}(\neg v, e))$.
- From the first conjunct, we get $s \models \neg \mathit{effcond}(v, e)$ and hence $s \not\models \mathit{effcond}(v, e)$.
- From the second conjunct, we get $s \models \neg v \vee \mathit{effcond}(\neg v, e)$.
- Case 1: $s \models \neg v$. Then v is false before applying e and remains false, so $s[\![e]\!] \not\models v$.
- Case 2: $s \models \mathit{effcond}(\neg v, e)$. Then v is deleted by e and not simultaneously added, so $s[\![e]\!] \not\models v$.

Regressing Formulas Through Effects

Regressing Formulas Through Effects: Idea

- We can now generalize regression from state variables to general formulas over state variables.
- The basic idea is to replace every occurrence of every state variable v by $regr(v, e)$ as defined in the previous section.
- The following definition makes this more formal.

Regressing Formulas Through Effects: Definition

Definition (Regressing a Formula Through an Effect)

In a propositional planning task, let e be an effect,
and let φ be a formula over propositional state variables.

The regression of φ through e, written $\mathit{regr}(\varphi, e)$,
is defined as the following logical formula:

$$
\begin{aligned}
\mathit{regr}(\top, e) &= \top \\
\mathit{regr}(\bot, e) &= \bot \\
\mathit{regr}(v, e) &= \mathit{effcond}(v, e) \vee (v \wedge \neg \mathit{effcond}(\neg v, e)) \\
\mathit{regr}(\neg\psi, e) &= \neg \mathit{regr}(\psi, e) \\
\mathit{regr}(\psi \vee \chi, e) &= \mathit{regr}(\psi, e) \vee \mathit{regr}(\chi, e) \\
\mathit{regr}(\psi \wedge \chi, e) &= \mathit{regr}(\psi, e) \wedge \mathit{regr}(\chi, e).
\end{aligned}
$$

Regressing Formulas Through Effects: Example

Example

Let $e = (b \rhd a) \wedge (c \rhd \neg a) \wedge b \wedge \neg d$.

Recall:

- $regr(a, e) \equiv b \vee (a \wedge \neg c)$
- $regr(b, e) \equiv \top$
- $regr(c, e) \equiv c$
- $regr(d, e) \equiv \bot$

We get:

$$\begin{aligned} regr((a \vee d) \wedge (c \vee d), e) &\equiv ((b \vee (a \wedge \neg c)) \vee \bot) \wedge (c \vee \bot) \\ &\equiv (b \vee (a \wedge \neg c)) \wedge c \\ &\equiv b \wedge c \end{aligned}$$

Regressing Formulas Through Effects: Correctness (1)

Lemma (Correctness of $regr(\varphi, e)$)

Let φ be a logical formula, e an effect and s a state of a propositional planning task.

Then $s \models regr(\varphi, e)$ iff $s[\![e]\!] \models \varphi$.

Regressing Formulas Through Effects: Correctness (2)

Proof.

The proof is by structural induction on φ.

Regressing Formulas Through Effects: Correctness (2)

Proof.

The proof is by structural induction on φ.

Induction hypothesis: $s \models \mathit{regr}(\psi, e)$ iff $s[\![e]\!] \models \psi$ for all proper subformulas ψ of φ.

Regressing Formulas Through Effects: Correctness (2)

Proof.

The proof is by structural induction on φ.

Induction hypothesis: $s \models \mathit{regr}(\psi, e)$ iff $s[\![e]\!] \models \psi$ for all proper subformulas ψ of φ.

Base case $\varphi = \top$:

We have $\mathit{regr}(\top, e) = \top$, and $s \models \top$ iff $s[\![e]\!] \models \top$ is correct.

Regressing Formulas Through Effects: Correctness (2)

Proof.

The proof is by structural induction on φ.

Induction hypothesis: $s \models \mathit{regr}(\psi, e)$ iff $s[\![e]\!] \models \psi$
for all proper subformulas ψ of φ.

Base case $\varphi = \top$:

We have $\mathit{regr}(\top, e) = \top$, and $s \models \top$ iff $s[\![e]\!] \models \top$ is correct.

Base case $\varphi = \bot$:

We have $\mathit{regr}(\bot, e) = \bot$, and $s \models \bot$ iff $s[\![e]\!] \models \bot$ is correct.

Regressing Formulas Through Effects: Correctness (2)

Proof.

The proof is by structural induction on φ.

Induction hypothesis: $s \models \mathit{regr}(\psi, e)$ iff $s[\![e]\!] \models \psi$
for all proper subformulas ψ of φ.

Base case $\varphi = \top$:

We have $\mathit{regr}(\top, e) = \top$, and $s \models \top$ iff $s[\![e]\!] \models \top$ is correct.

Base case $\varphi = \bot$:

We have $\mathit{regr}(\bot, e) = \bot$, and $s \models \bot$ iff $s[\![e]\!] \models \bot$ is correct.

Base case $\varphi = v$:

We have $s \models \mathit{regr}(v, e)$ iff $s[\![e]\!] \models v$ from the previous lemma. ...

Regressing Formulas Through Effects: Correctness (3)

Proof (continued).

Inductive case $\varphi = \neg\psi$:

$$
\begin{aligned}
s \models \mathit{regr}(\neg\psi, e) \text{ iff } & s \models \neg \mathit{regr}(\psi, e) \\
\text{iff } & s \not\models \mathit{regr}(\psi, e) \\
\text{iff } & s[\![e]\!] \not\models \psi \\
\text{iff } & s[\![e]\!] \models \neg\psi
\end{aligned}
$$

Regressing Formulas Through Effects: Correctness (3)

Proof (continued).

Inductive case $\varphi = \neg\psi$:

$$\begin{aligned} s \models \mathit{regr}(\neg\psi, e) &\text{ iff } s \models \neg\mathit{regr}(\psi, e) \\ &\text{iff } s \not\models \mathit{regr}(\psi, e) \\ &\text{iff } s[\![e]\!] \not\models \psi \\ &\text{iff } s[\![e]\!] \models \neg\psi \end{aligned}$$

Inductive case $\varphi = \psi \vee \chi$:

$$\begin{aligned} s \models \mathit{regr}(\psi \vee \chi, e) &\text{ iff } s \models \mathit{regr}(\psi, e) \vee \mathit{regr}(\chi, e) \\ &\text{iff } s \models \mathit{regr}(\psi, e) \text{ or } s \models \mathit{regr}(\chi, e) \\ &\text{iff } s[\![e]\!] \models \psi \text{ or } s[\![e]\!] \models \chi \\ &\text{iff } s[\![e]\!] \models \psi \vee \chi \end{aligned}$$

Regressing Formulas Through Effects: Correctness (3)

Proof (continued).

Inductive case $\varphi = \neg\psi$:

$$\begin{aligned} s \models \mathit{regr}(\neg\psi, e) &\text{ iff } s \models \neg\mathit{regr}(\psi, e) \\ &\text{iff } s \not\models \mathit{regr}(\psi, e) \\ &\text{iff } s[\![e]\!] \not\models \psi \\ &\text{iff } s[\![e]\!] \models \neg\psi \end{aligned}$$

Inductive case $\varphi = \psi \vee \chi$:

$$\begin{aligned} s \models \mathit{regr}(\psi \vee \chi, e) &\text{ iff } s \models \mathit{regr}(\psi, e) \vee \mathit{regr}(\chi, e) \\ &\text{iff } s \models \mathit{regr}(\psi, e) \text{ or } s \models \mathit{regr}(\chi, e) \\ &\text{iff } s[\![e]\!] \models \psi \text{ or } s[\![e]\!] \models \chi \\ &\text{iff } s[\![e]\!] \models \psi \vee \chi \end{aligned}$$

Inductive case $\varphi = \psi \wedge \chi$:

Like previous case, replacing "$\vee$" by "$\wedge$" and replacing "or" by "and".

□

Regressing Formulas Through Operators

Regressing Formulas Through Operators: Idea

- We can now regress arbitrary formulas through arbitrary effects.
- The last missing piece is a definition of regression through operators, describing exactly in which states s applying a given operator o leads to a state satisfying a given formula φ.
- There are two requirements:
  - The operator o must be applicable in the state s.
  - The resulting state $s[\![o]\!]$ must satisfy φ.

Regressing Formulas Through Operators: Definition

Definition (Regressing a Formula Through an Operator)

In a propositional planning task, let o be an operator, and let φ be a formula over state variables.

The regression of φ through o, written $\mathit{regr}(\varphi, o)$, is defined as the following logical formula:

$$\mathit{regr}(\varphi, o) = \mathit{pre}(o) \wedge \mathit{regr}(\varphi, \mathit{eff}(o)).$$

Regressing Formulas Through Operators: Correctness (1)

Theorem (Correctness of $regr(\varphi, o)$)

Let φ be a logical formula, o an operator and s a state of a propositional planning task.

Then $s \models regr(\varphi, o)$ iff o is applicable in s and $s[\![o]\!] \models \varphi$.

Regressing Formulas Through Operators: Correctness (2)

Reminder: $regr(\varphi, o) = pre(o) \wedge regr(\varphi, eff(o))$

Proof.

Case 1: $s \models pre(o)$.

Then o is applicable in s and the statement we must prove simplifies to: $s \models regr(\varphi, e)$ iff $s[\![e]\!] \models \varphi$, where $e = eff(o)$. This was proved in the previous lemma.

Regressing Formulas Through Operators: Correctness (2)

Reminder: $regr(\varphi, o) = pre(o) \wedge regr(\varphi, eff(o))$

Proof.

Case 1: $s \models pre(o)$.

Then o is applicable in s and the statement we must prove simplifies to: $s \models regr(\varphi, e)$ iff $s[\![e]\!] \models \varphi$, where $e = eff(o)$.
This was proved in the previous lemma.

Case 2: $s \not\models pre(o)$.

Then $s \not\models regr(\varphi, o)$ and o is not applicable in s.
Hence both statements are false and therefore equivalent. □

Regression Examples (1)

Examples: compute regression and simplify to DNF

- $regr(b, \langle a, b \rangle)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot))$
  $\equiv a$
- $regr(b \wedge c \wedge d, \langle a, b \rangle)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot)) \wedge (\bot \vee (c \wedge \neg\bot)) \wedge (\bot \vee (d \wedge \neg\bot))$
  $\equiv a \wedge c \wedge d$
- $regr(b \wedge \neg c, \langle a, b \wedge c \rangle)$
  $\equiv a \wedge (\top \vee (b \wedge \neg\bot)) \wedge \neg(\top \vee (c \wedge \neg\bot))$
  $\equiv a \wedge \top \wedge \bot$
  $\equiv \bot$

Regression Examples (2)

Examples: compute regression and simplify to DNF

- $regr(b, \langle a, c \rhd b \rangle)$
  $\equiv a \wedge (c \vee (b \wedge \neg\bot))$
  $\equiv a \wedge (c \vee b)$
  $\equiv (a \wedge c) \vee (a \wedge b)$
- $regr(b, \langle a, (c \rhd b) \wedge ((d \wedge \neg c) \rhd \neg b) \rangle)$
  $\equiv a \wedge (c \vee (b \wedge \neg(d \wedge \neg c)))$
  $\equiv a \wedge (c \vee (b \wedge (\neg d \vee c)))$
  $\equiv a \wedge (c \vee (b \wedge \neg d) \vee (b \wedge c))$
  $\equiv a \wedge (c \vee (b \wedge \neg d))$
  $\equiv (a \wedge c) \vee (a \wedge b \wedge \neg d)$

Summary

Summary

- Regressing a propositional state variable through an (arbitrary) operator must consider two cases:
  - state variables made true (by add effects)
  - state variables remaining true (by absence of delete effects)
- Regression of propositional state variables can be generalized to arbitrary formulas φ by replacing each occurrence of a state variable in φ by its regression.
- Regressing a formula φ through an operator involves regressing φ through the effect and enforcing the precondition.