Motivation: Why Analizing the Expressive Power?

- Why do we care about the expressive power of planning formalisms?
- How can we compare planning formalisms?
- What are the implications of different expressive powers for planning algorithms?

Motivation: Why Complexity Analysis?

- Complexity analysis helps us understand the limitations of planning algorithms.
- It guides us in designing efficient algorithms.
- It reveals the inherent difficulty of planning problems.

Overview

1. Introduction
   - Motivation
   - Expressive Power

2. Computational Complexity of Classical Planning
   - STRIPS with First-Order Terms and DATALOG-STRIPS
   - Propositional STRIPS and Restrictions
   - Domain-Dependent Planning

3. Expressive Power of Planning Formalisms
   - How to Compare Expressive Power?
   - The Compilability Approach
   - Conditional Effects and General Preconditions

4. Conclusions and Further Research
A small Example

State, action, goal

State change induced by operator

Operator or action: Operator schema with empty parameter list

Operator, effect of action, operator schema

Operator schema: Operator schemata, operators & state change

Some basic knowledge in classical planning: Operator

Some basic knowledge in classical STRIPS planning: operator

some basic knowledge in first-order logic, propositional logic, complexity classes, and Turing machine

Prequisites
A Plan for Changing the Light Bulb

State resulting from executing a plan:

More Expressive Power
The Complexity Class P

A problem is a member of the complexity class P if the problem can be decided by a deterministic Turing machine using only polynomial time, i.e., the number of computation steps on input $w$ is bounded by $p(|w|)$, where $p$ is a polynomial.

The problems in P are considered as "efficiently solvable" since there exists an algorithm that can decide the problem in polynomial time.

Reductions

The Complexity Class NP

There are many problems that are not known to be decidable in polynomial time, but there exists no proof that these problems need necessarily super-polynomial time.

Use non-deterministic machines to characterize these problems.

NP is the class of languages that are decided by non-deterministic Turing machines using only polynomial time, i.e., the number of computation steps on input $w$ is bounded by $p(|w|)$, where $p$ is a polynomial.

The problems in P are considered as "efficiently solvable" since there exists an algorithm that can decide the problem in polynomial time.

Non-deterministic machines can be decided by a deterministic Turing machine using only polynomial time, i.e., the number of computation steps on input $w$ is bounded by $p(|w|)$, where $p$ is a polynomial.

A problem is a member of the complexity class P if the problem can be decided by a deterministic Turing machine using only polynomial time, i.e., the number of computation steps on input $w$ is bounded by $p(|w|)$, where $p$ is a polynomial.

Reducions
NP-complete Problems

• A problem is **NP-complete** iff it is NP-hard and in NP.

• Example: **SAT** — the satisﬁability problem for propositional logic — is NP-complete (Cook/Karp)

• **Membership proofs:** Guess & check algorithm (veriﬁcation in poly. time = short “proofs”)

• **Hardness proofs:** Either **generic reduction** to show \( L \leq_m X \) for all \( L \in \text{NP} \) or use a known NP-complete problem \( K \) and show \( K \leq_m X \).

The Class PSPACE

**Deﬁnition.** PSPACE (NPSPACE) is the class of decision problems that can be decided on deterministic (non-deterministic) Turing machines using only polynomial size.

Some facts about PSPACE:

• PSPACE is closed under complements (as all other deterministic classes)

• PSPACE is identical to NPSPACE (because NDTMs can be simulated on DTMs using only quadratic space)

• \( \text{NP} \subseteq \text{PSPACE} \) (because in polynomial time one can “visit” only polynomial space, i.e., \( \text{NP} \subseteq \text{NPSPACE} \))

• It is unknown whether \( \text{NP} \neq \text{PSPACE} \), but it is believed that this is true.

PSPACE-completeness

A decision problem (or language) is **PSPACE-complete**, if it is in PSPACE and all other problems in PSPACE can be polynomially many-one reduced to it.

Intuitively, PSPACE-complete problems are the “hardest” problems in PSPACE (similar to NP-completeness). They appear to be “harder” than NP-complete problems from a practical point of view.

Example for a PSPACE-complete problem:

**NDFA equivalence problem:**

**Instance:** Two non-deterministic ﬁnite state automata \( A_1 \) and \( A_2 \).

**Question:** Are the languages accepted by \( A_1 \) and \( A_2 \) identical?
There are complexity classes above PSPACE (EXPTIME, SX-PSPACE), there are (infinitely many) classes between NP and PSPACE, there are (infinitely many) classes inside P, and a positive integer n, in other words, does there exist a plan \( \pi \) of length \( n \) or less that solves \( s \)?

**Plan existence problem (PLANEX):**

- Instance: \( s, \pi, n \)
- Question: Does there exist a plan \( \pi \) of length \( n \) or less that solves \( s \)?

From a practical point of view, also PLANGEN (generating a plan) and PLANLENGEN (generating a plan that solves \( s \)) are interesting (but at least as hard as the decision problems).

**Analyzing Planning:**

- there are (infinitely many) classes inside P.
- there are (infinitely many) classes between NP and PSPACE.
- there are (infinitely many) classes above PSPACE (EXPTIME, SX-PSPACE).