Description Logics: a Nice Family of Logics — Complexity, Part 1 —

Uli Sattler[1] *Thomas Schneider*[2]

[1]School of Computer Science, University of Manchester, UK

[2]Department of Computer Science, University of Bremen, Germany

ESSLLI, 17 August 2016

Goal for today & tomorrow

Automated reasoning plays an **important role** for DLs.

- It allows the development of intelligent applications.
- The expressivity of DLs is strongly tailored towards this goal.

Requirements for automated reasoning:

- Decidability of the relevant decision problems
- Low complexity if possible
- Algorithms that perform well in practice

Yesterday & today: 1 & 3

Now: 2

And now ...

Cognitive versus Computational Complexity

Consider decision problems for reasoning, e.g. $\mathcal{O} \models^? C \sqsubseteq D$ (⊛)

Cognitive complexity (more on Friday)

- How hard is it, for a human, to decide or understand (⊛)?
- interesting, little understood topic
- relevant to provide tool support for ontology engineers

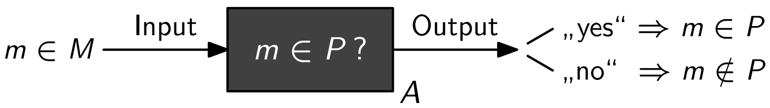
Computational complexity **(today)**

- How much time/space is needed to decide (⊛)?
- interesting, well understood topic
- loads of results thanks to relationships DL – FOL – ML
- relevant to understand
  - trade-off: expressivity of a DL ↔ complexity of reasoning
  - whether a given algorithm is optimal/can be improved

Decidability

A **(decision) problem**

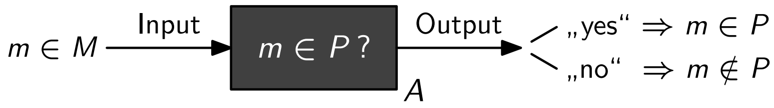
- ... is a subset $P \subseteq M$
- Examples:
  - P = set of all prime numbers, $M = \mathbb{N}$
  - P = set of triples $(\mathcal{O}, C, D)$ with $\mathcal{O} \models C \sqsubseteq D$, M = set of *all* triples $(\mathcal{O}, C, D)$ from $\mathcal{ALC}$
- think of it as a black box:



Decidability: P is decidable if there is an algorithm A that implements the black box.

(Programming language and machine model are largely irrelevant)

Computational complexity



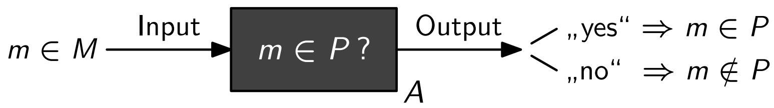
Complexity:
measures time/space needed by A in the worst case, depending on the length of the input $|m|$

- **Polynomial time:** Number of computation steps is $\leqslant pol(|m|)$, for some polynomial function pol
- **Polynomial space:** Number of memory cells used is $\leqslant pol(|m|)$
- **Exponential time:** Number of computation steps is $\leqslant 2^{pol(|m|)}$
- ...

Some standard complexity classes

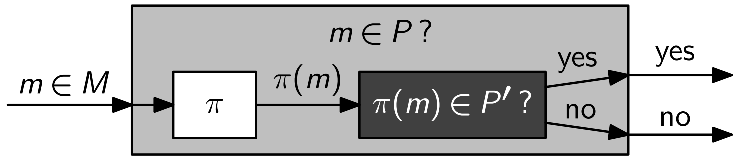
Name	Restriction	Example problem
L	logarithmic space	graph connectivity
NL	nondeterministic log. space	graph accessibility
P	polynomial time	prime numbers
NP	nondeterm. polynomial time	(propositional) SAT
PSpace	polynomial space	QBF-SAT
ExpTime	exponential time	CTL-SAT
NExpTime	nondeterm. exponential time	
ExpSpace	exponential space	
⋮	⋮	
	undecidable	first-order SAT

Reductions



A **(polynomial) reduction** of $P \subseteq M$ to $P' \subseteq M'$ is a (poly-time computable) function $\pi : M \to M'$ with

$$m \in P \quad \text{iff} \quad \pi(m) \in P'$$



If P reducible to P' then P is **"at most as hard"** as P'.

If **all** problems from a complexity class $\mathcal{C}$ are reducible to P, then P is **hard for** $\mathcal{C}$.

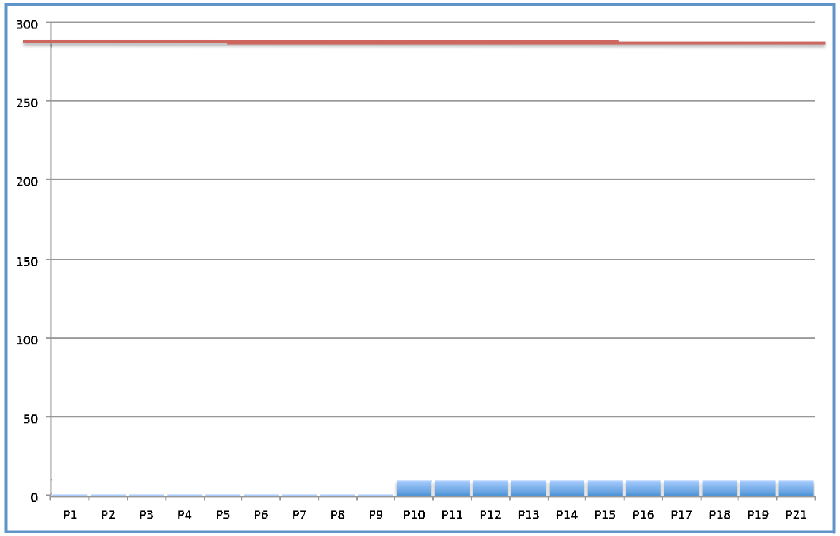
Determining the complexity

Usually one shows that a problem $P \subseteq M$ is . . .

- **in** a complexity class $\mathcal{C}$, by
  - designing/finding an algorithm A that solves P,
  - showing that A is sound, complete, and terminating
  - showing that A runs, for every $m \in M$, in **at most** $\mathcal{C}$ ressources
  - . . . A can be, e.g., a reduction to a problem known to be in $\mathcal{C}$
- **hard for** $\mathcal{C}$, by finding
  - a suitable problem $P' \subseteq M'$ that is known to be **hard for** $\mathcal{C}$
  - and a reduction of P' to P
- **complete for** $\mathcal{C}$, by showing that P is
  - **in** $\mathcal{C}$ and
  - **hard for** $\mathcal{C}$

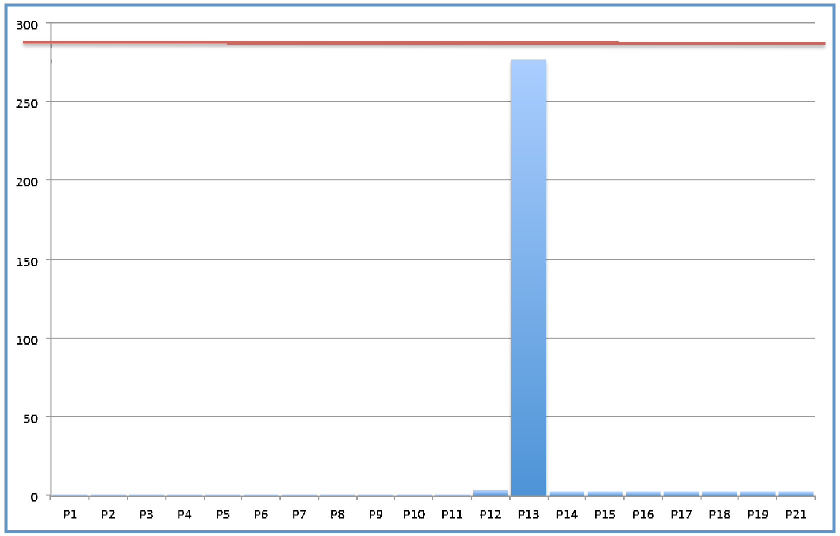
Worst-case complexity

Worst case: algorithm runs, for all $m \in M$, in **at most** $\mathcal{C}$ resources, e.g., like this on all problems of size 7:



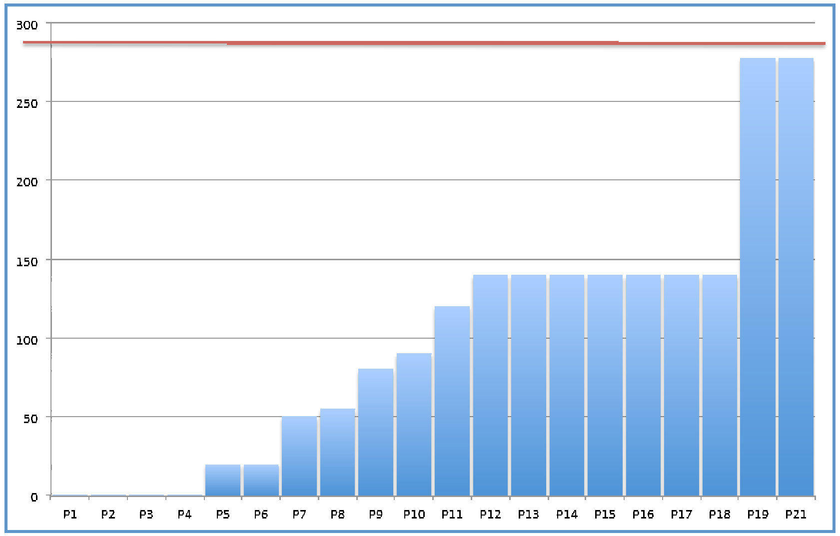
Worst-case complexity

Worst case: algorithm runs, for all $m \in M$, in **at most** $\mathcal{C}$ resources, e.g.,or like this on all problems of size 7:



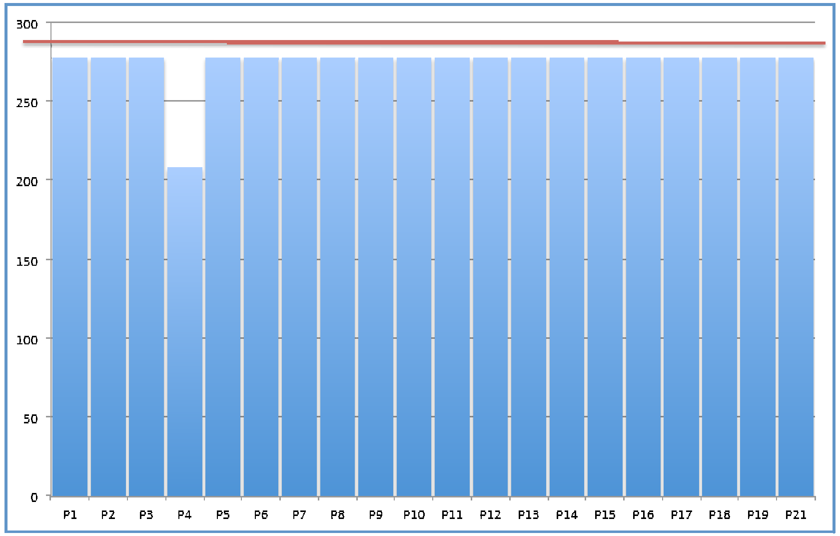
Worst-case complexity

Worst case: algorithm runs, for all $m \in M$, in **at most** $\mathcal{C}$ resources, e.g.,or like this on all problems of size 7:



Worst-case complexity

Worst case: algorithm runs, for all $m \in M$, in **at most** $\mathcal{C}$ resources, e.g.,or like this on all problems of size 7:



Known complexity results from Days 2–3

From the tableau technique, we know that

- all considered reasoning problems are **decidable** for $\mathcal{ALCQI}$
  because the tableau algorithm is sound, complete, terminating
- consistency of $\mathcal{ALC}$ ontologies is **in ExpSpace**
  and so are satisfiability and subsumption w.r.t. ontologies
  ➻ We can do better: we'll show they are **ExpTime-complete**
- satisfiability and subsumption of $\mathcal{ALC}$ concepts are **in PSpace**
  ➻ We **cannot** do better: we'll show that they are **PSpace-hard**

And now . . .

EXPTIME-membership

We start with an ExpTime upper bound
for concept satisfiability in $\mathcal{ALC}$ relative to TBoxes.

Theorem

The following problem is in ExpTime.

Input: an $\mathcal{ALC}$ concept C_0 and an $\mathcal{ALC}$ TBox $\mathcal{T}$

Question: is there a model $\mathcal{I} \models \mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$?

We'll use a technique known from modal logic:
type elimination [Pratt 1978]

The basis is a *syntactic* notion of a *type*.

Syntactic types

We assume that

- the input concept C_0 is in NNF
- the input TBox is $\mathcal{T} = \{\top \sqsubseteq C_\mathcal{T}\}$ with $C_\mathcal{T}$ in NNF

Let $\mathsf{sub}(C_0, \mathcal{T})$ be the set of subconcepts of C_0 and $C_\mathcal{T}$.
A **type for C_0 and $\mathcal{T}$** is a subset $t \subseteq \mathsf{sub}(C_0, \mathcal{T})$ such that

1. $A \in t$ iff $\neg A \notin t$ for all $\neg A \in \mathsf{sub}(C_0, \mathcal{T})$
2. $C \sqcap D \in t$ iff $C \in t$ and $D \in t$ for all $C \sqcap D \in \mathsf{sub}(C_0, \mathcal{T})$
3. $C \sqcup D \in t$ iff $C \in t$ or $D \in t$ for all $C \sqcup D \in \mathsf{sub}(C_0, \mathcal{T})$
4. $C_\mathcal{T} \in t$

Intuition:
Types describe domain elements completely, up to $\mathsf{sub}(C_0, \mathcal{T})$.

General idea

General idea of type elimination for input C_0, $\mathcal{T}$:

- Generate all types for C_0 and $\mathcal{T}$ (exponentially many).
- Repeatedly eliminate types that cannot occur in any model of C_0 and $\mathcal{T}$.
- Check whether some type containing C_0 has survived.
- If yes, return "satisfiable"; otherwise "unsatisfiable".