Algorithms and Computability

Short Summary of the Past Lectures

Christian Schilling (christianms@cs.aau.dk)

Goal: Study the limits of computation

- In previous courses, we were always concerned with how to solve problems, and how to do that efficiently
- In this course, we rather looked at the other end of the spectrum
- The question was: are there problems we cannot solve algorithmically?
- To study this question, we first had to formalize some things:
  - What is a problem?
  - What is an algorithmic solution?

What is a problem?

- We focused on yes/no problems
- We defined a problem as the set of problem instances for which the answer is "yes"
- A problem instance is just the input we expect, which we can view as a string/word
- Thus, a problem is a set of words, aka a formal language
- The complement of a problem is then the set of problem instances for which the answer is "no"
- This is a very robust definition (i.e., we can indeed phrase many yes/no questions like that)

What is an algorithmic solution?

- We introduced the (innocent-looking) paradigm of a deterministic Turing machine (DTM)
- Language of a DTM: words the DTM accepts
- All other words: DTM either rejects or does not halt
- We defined the classes of computable and computably enumerable languages
- Extensions (multiple tapes, two-sided infinite tape, nondeterminism)
  - Increase efficiency
  - Do not increase effectiveness (same class of languages)
- Church-Turing thesis: "computable" captures precisely our notion of algorithmic solution
  - Not a formal thesis, so it cannot be proven (only refuted)
  - Stood the test of time so far

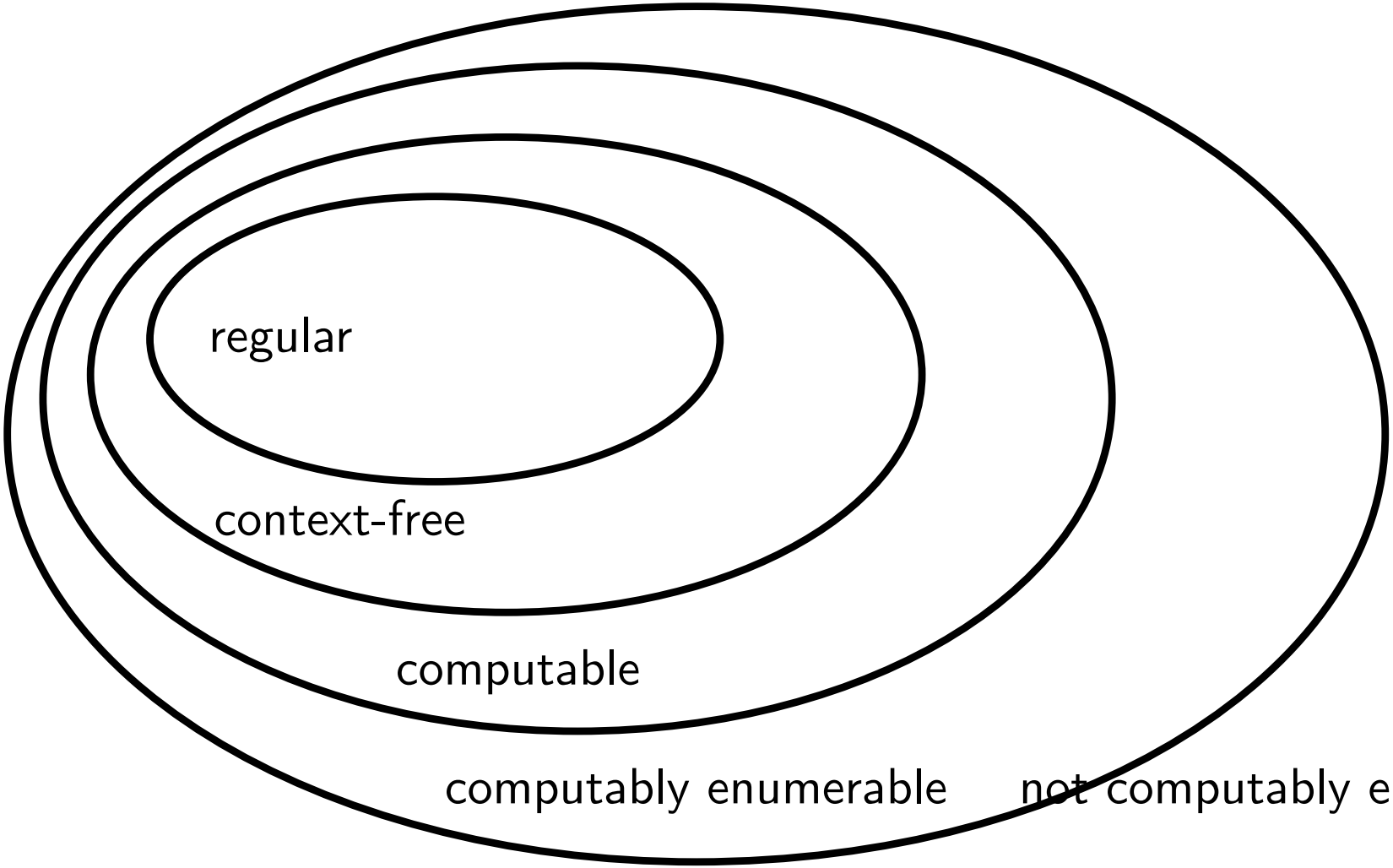
What are the limits of computation?

- We saw that DTMs can be encoded as words
- That allows us to let DTMs take other DTMs as inputs

  In other words, we can reason about problems reasoning about other problems

- We then constructed the halting problem and proved that it is not computable

Classes of formal languages



How can we prove more problems "hard"?

- (Mapping) Reduction $A \leq_m B$
- Reduce (= translate) an instance of problem A to an instance of problem B (while preserving membership)
- Intuitively, A is "at most as hard" as B (but can be "easier")
  - (Thus, the term "reduce" may be misleading)
  - We have not seen this, but every (nontrivial) computably enumerable problem is reducible to the halting problem

    Clearly, some of these problems are much "easier"