COMP3630 / COMP6363

week 11: **Bonus Lecture**

slides created by: Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- Hardness with “powerful” reductions.
- Recap on stati of Turing Machines

On **P** membership vs. **P**-completeness

- For (almost*) all classes we looked into completeness.
  (*Due to time constraints we sometimes did not look into completeness.)
- This makes sense, because providing, e.g., an **EXPTIME** membership proof for a language doesn't prevent it from also being in **NP** or even **P**!
- However, we never did that for **P**... We only showed membership! Why?

Theorem. Under Karp reductions, all non-trivial problems in **P** are **P**-complete.

Idea.

- We can use the polytime granted by the reduction to solve the other problem!
- We'll see in the proof (next slide) why trivial problems don't work.

P-hardness of non-trivial **P** problems, Proof

- Let $L \in \mathbf{P}$ and $L \neq \emptyset$ and $L \neq \Sigma^*$.
- This implies that there are w_{yes}, w_{no}, $w_{yes} \neq w_{no}$, such that $w_{yes} \in L$ and $w_{no} \notin L$.
- To show **P**-hardness, we prove that "it can be used" (under Karp-reductions!) to solve any other problem in **P**. In other words: for any $L' \in \mathbf{P}$ we get $L' \leq_{\mathbf{P}} L$
- We used the quotes here because we are actually not going to use L's decider (TM) to solve any other problem, but just the reduction itself! (And the yes and no instances, but only implicitly.)
- Let L' be some problem in **P**. Need to show: for any word w, $w \in L'$ iff $f(w) \in L$.
  - Check if $w \in L'$, which we can do in poly-time (as part of the "reduction").
  - If yes, return $f(w) = w_{yes}$
  - If no, return $f(w) = w_{no}$
- This gives a poly-time reduction.

Or does it?

- Note that we don't know how long w_{yes}/w_{no} are in comparison to the input w. But f needs to run in poly-time! Is the proof wrong after all?
- w_{yes}/w_{no} are fixed in advance, "hard-coded" into the function. So the runtime for writing the word $f(w)$ is <u>constant</u> and does not scale with the length of w!
  (f however still needs non-constant poly-time to check for $w \in L'$.)

P-completeness, final notes

So, why did we never talk about **P**-completeness?

- Because we only covered Karp-reductions, for which he have that **P** membership implies **P**-hardness.
- Note that **P**-hardness/completeness is not defined using Karp-reductions! (Now you see why!)
- They are based on log-time reductions to differentiate between problems in **P** and those (believed to be?) below. This goes beyond this course, just remember that **P**-hardness is defined differently than used in this course!

Why polytime-reductions for **PSPACE**-hardness?

Recall last week (week 10), chapter 11c. We asked why we define **PSPACE**-hardness via poly-time reductions, rather than poly-space reductions. Now you know why!

Theorem.
Under poly-space reductions, all non-trivial problems in **PSPACE** are **PSPACE**-complete. (This includes problems in **P**!)

Proof.
Identical to the one before. Just replace **P** by **PSPACE** and poly-time by poly-space.

Disclaimer

- The following slides are exactly the same as those in week 5, pages 9 to 11.
- Thus they will be excluded <u>here</u> from the upload to prevent redundancy.