COMS4236: Introduction to Computational Complexity

Summer 2014

Mihalis Yannakakis
Outline

• Hardness and completeness

• Composition of reductions

• Graph Reachability is NL-complete
Hardness and Completeness

• A Language L or decision problem $\Pi$ is **hard** for a class C, or **C–hard**, under a type of reduction (eg. polynomial or log-space reduction) if every problem in the class C reduces to it.

• It is **complete** for a class C, or **C–complete**, under a type of reduction if
  1. it belongs to C
  2. it is C–hard under the reduction.
Hardness and Completeness

• **NP-complete**: usually we use p-reductions for NP and for classes above P. It turns out that the usual NP-complete problems are complete also under the more restrictive log-space reduction.

• **NL-complete, P-complete**: usually we use log-space reductions for P and classes below it

• **Reason**: All nontrivial problems in P are complete under p-reductions (nontrivial means there is a yes instance and a no instance), so p-reductions do not give any useful information inside P
Properties of completeness

• Intuitively, Complete problems are the hardest in the class

• If L is complete for a class $C$ (e.g. $NP$) under polynomial reductions then $C \subseteq P \iff L \in P$

• If L is complete for a class $C$ (e.g. $P$ or $NL$) under logspace reductions then $C \subseteq \text{LOGSPACE} \iff L \in \text{LOGSPACE}$
Polynomial-time reductions compose

- \( L_1 \leq_p L_2 \) and \( L_2 \leq_p L_3 \) \( \implies \) \( L_1 \leq_p L_3 \)

Suppose \( f \) reduces \( L_1 \) to \( L_2 \) and \( g \) reduces \( L_2 \) to \( L_3 \)
- Then \( x \in L_1 \iff f(x) \in L_2 \iff g(f(x)) \in L_3 \)

- Suppose \( f \) runs in time \( O(n^c) \) and \( g \) runs in time \( O(n^d) \)
- Then the algorithm that first runs \( f \) on input \( x \) and then \( g \) on \( f(x) \) runs in time \( O( (n^c)^d) = O( n^{cd} ) \)
Log Space reductions compose

- \( L_1 \leq_{\log} L_2 \) and \( L_2 \leq_{\log} L_3 \) \( \Rightarrow \) \( L_1 \leq_{\log} L_3 \)

- Same construction as in the proof that \( L_1 \leq_{\log} L_2 \) and \( L_2 \in \text{LOGSPACE} \) imply \( L_1 \in \text{LOGSPACE} \)

- Combine the log-space TMs \( M_{1 \rightarrow 2} \) and \( M_{2 \rightarrow 3} \) for the two reductions, but do not write explicitly the output of the first reduction but only generate it one symbol at a time, as needed.
Properties of completeness

• Compositions of reductions implies:
  • If \( L_1 \) is hard for a class under \( p \)-reductions and \( L_1 \leq_P L_2 \) then \( L_2 \) is also hard for the class
  • If \( L_1 \) is hard for a class under logspace-reductions and \( L_1 \leq_{\text{log}} L_2 \) then \( L_2 \) is also hard for the class

• Complementation implies:
  • co\( C \)-complete problems = complements of \( C \)-complete problems.
  • eg. co\( \text{NP} \)-complete problems = complements of NP-complete problems.
Reachability is NL-complete (under log reductions)

- Corollary: \( NL=L \iff \text{Reachability} \in L \)

- Must show: For every language \( L \) in NL there is a logspace reduction from \( L \) to \text{Reachability}.
- Take a language \( L \) in NL, let \( M \) be NTM that decides \( L \) in space \( \log n \). Must construct a \( O(\log n) \)-space (deterministic) TM \( M' \) that maps every string \( x \) over the input alphabet \( \Sigma \) of \( L \) to an instance (graph \( G \), nodes \( s,t \)) such that \( x \in L \iff s \) can reach \( t \) in \( G \)
- \( G = \) configuration graph of \( M \) on input \( x \)
- \( s = \) initial configuration
- \( t = \) accepting configuration (assume wlog unique)
- Remains to show how to construct \( G \) in log space.
Construction of Configuration Graph

• Can generate graph for example as a list of nodes, followed by a list of edges
• Nodes: Enumerate systematically all configurations of $M$ (remember, not including the input) that use space $\leq \log n$
• Edges: Generate each $\log n$-space configuration $C$ of $M$, and for each possible move of $M$ from $C$, generate the next configuration $C'$ and output edge $(C,C')$
• Output $s =$ initial configuration, $t =$ accepting configuration

• Only need to keep track of the current configuration $C$ being considered, and generate the next config. $C'$
• $O(\log n)$ space