Slide 1

REDUCTIONS AND COMPLETENESS

- A complexity class is an infinite collection of languages.
  NP contains TSP(D), SAT, HORNSAT, REACHABILITY, ...
- Can problems be ordered using a relation "*at least as hard as*"?
- Such a relation can be defined using a notion of *reduction*:
  A is at least as hard as B if B reduces to A.
- B reduces to A if there is a transformation R which for every input x of B produces an equivalent input $R(x)$ of A
  where equivalent means that the answer ("yes"/"no") for $R(x)$ considered as A's input is correct answer to x as an input of B.

Slide 2

- The notion of reduction seems reasonable to capture that A is at least as hard as B except when R is very hard to compute.
  Without any limit, e.g., TSP(D) can be reduced to HORNSAT!
- Possible limit on resources in reductions:
  Karp reductions (polynomial-time many-one reductions)
  Cook reductions (Turing reductions)

Definition. A language L_1 is reducible to L_2 iff there is a function R from strings to strings computable by a deterministic Turing machine in space $\mathcal{O}(\log n)$ such that for all inputs x the following holds:
$x \in L_1$ iff $R(x) \in L_2$.
R is called a reduction from L_1 to L_2.

Slide 3

Proposition. If R is a reduction computed by a TM M, then for all inputs x, M halts after a polynomial number of steps.

Proof sketch.

- There are $\mathcal{O}(nc^{\log n})$ possible configurations for M on input x where $|x| = n$.
- Since M is deterministic and halts on every input, it cannot repeat any configuration. Hence M halts in at most $c'nc^{\log n} = c'nn^{\log c} = \mathcal{O}(n^k)$ steps for some k.

Note that as output string $R(x)$ is computed in a polynomial number of steps, its length is also polynomial w.r.t. $|x|$.

Slide 4

Example. HAMILTON PATH:
INSTANCE: A graph G.
QUESTION: Is there a path in G that visits every node exactly once?

HAMILTON PATH is reducible to SAT using a reduction R as follows:

For a graph G, $R(G)$ is a conjunction of clauses.
Suppose G has n nodes, $1, 2, \ldots, n$.
Then $R(G)$ has n^2 Boolean variables, $x_{ij}, 1 \leq i, j \leq n$
(x_{ij} represents the fact that j is the ith node in the Hamilton path).
$R(G)$ is the conjunction of the following clauses:

1. For each node j: $x_{1j} \vee \cdots \vee x_{nj}$
   (node j appears on the path)
2. For all j, i, k where $i \neq k$: $\neg x_{ij} \vee \neg x_{kj}$
   (j cannot be the ith and kth node)

Slide 5

3. For all i: $x_{i1} \vee \cdots \vee x_{in}$
   (some node is ith)
4. For all i, j, k where $j \neq k$: $\neg x_{ij} \vee \neg x_{ik}$
   (no two nodes can be ith)
5. For each pair (i, j) which is **not** an edge in G for all $k = 1, \ldots, n-1$: $\neg x_{ki} \vee \neg x_{(k+1)j}$
   (it must not be the case that j comes right after i in the path)

To show that R is a reduction from HAMILTON PATH to SAT two claims need to be established:
(i) G has a Hamilton path iff $R(G)$ is satisfiable and
(ii) R can be computed in space $\mathcal{O}(\log n)$

Slide 6

(i) ($\Leftarrow$) Let $R(G)$ have a satisfying truth assignment T.

By clauses (1,2) for every j there is unique i such that $T(x_{ij}) = \mathbf{true}$.

By clauses (3,4) for every i there is unique j such that $T(x_{ij}) = \mathbf{true}$.

Hence, T represents a permutation $\pi(1), \ldots, \pi(n)$ of the nodes where $\pi(i) = j$ iff $T(x_{ij}) = \mathbf{true}$

By clauses (5) for all k, $(\pi(k), \pi(k+1))$ is an edge in G. Hence $(\pi(1), \ldots, \pi(n))$ a Hamilton path.

($\Rightarrow$) Let G have a Hamilton path $(\pi(1), \ldots, \pi(n))$ where π is a permutation. Then the truth assignment T,

with $T(x_{ij}) = \mathbf{true}$ if $\pi(i) = j$ else $T(x_{ij}) = \mathbf{false}$,

satisfies all the clauses.

Slide 7

(ii) We show that that $R(G)$ can be computed in space $\mathcal{O}(\log n)$.
Given G as an input, a TM M outputs $R(G)$ as follows:
M first outputs clauses (1-4) not depending on G one by one using three counters i, j, k. Each counter needs (in binary) at most $\log n$ space.
M outputs clauses (5) one by one as follows:
it considers each pair (i, j), checks whether (i, j) is an edge in G and if not it outputs one by one $\neg x_{ki} \vee \neg x_{(k+1)j}$ for all $k = 1, \ldots, n-1$.
Again space is needed only for the counters i, j, k at most $3 \log n$ in total.

Hence, $R(G)$ can be computed in space $\mathcal{O}(\log n)$.

Slide 8

Example. REACHABILITY is reducible to CIRCUIT VALUE.

- Given a graph G, the reduction R constructs a variable-free circuit $R(G)$ such that the output of $R(G)$ is $\mathbf{true}$ iff there is a path from 1 to n in G.
- The gates of $R(G)$ are of the form
  g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$
  (g_{ijk} is $\mathbf{true}$ iff there is a path in G from i to j not using any intermediate node bigger that k.)
  h_{ijk} with $1 \leq i, j, k \leq n$
  (h_{ijk} is $\mathbf{true}$ iff there is a path in G from i to j not using any intermediate node bigger that k but using k as an intermediate node.)

Slide 9

- $R(G)$ is the following circuit:
  1. For $k = 0$ all g_{ijk} gates are input gates such that g_{ijk} is a **true** gate if (i, j) is an edge in G else it is a **false** gate.
  2. For $k = 1, 2, \ldots, n$,
     h_{ijk} is an AND gate with predecessors $g_{ik(k-1)}$ and $g_{kj(k-1)}$
     g_{ijk} is an OR gate with predecessors $g_{ij(k-1)}$ and h_{ijk}

  g_{1nn} is the output gate of $R(G)$.
- $R(G)$ forms an acyclic circuit and it is variable-free.

Slide 10

- Values of the gates correspond to their given intuitive readings. (Proof by induction on $k = 0, 1, \ldots, n$)
  For $k = 0$ this is clearly the case.
  For $k > 0$ the definition of h_{ijk} is $g_{ik(k-1)} \wedge g_{kj(k-1)}$. Hence, by the inductive hypothesis h_{ijk} is **true** if there is a path from i to k and from k to j not using any intermediate node bigger that $k - 1$, i.e., there is a path from i to j not using any intermediate node bigger that k but going through k.
  The definition of g_{ijk} is $g_{ij(k-1)} \vee h_{ijk}$. By the inductive hypothesis g_{ijk} is **true** if there is a path from i to j not using any intermediate node bigger that $k - 1$ or a path not using any node bigger that k but going through k iff there is a path from i to j not using any intermediate node bigger that k.

Slide 11

- The output of $R(G)$ is **true** iff g_{1nn} is **true** iff there is a path from 1 to n in G without any intermediate nodes bigger than n iff there is a path from 1 to n in G.
- $R(G)$ can be computed in $\mathcal{O}(\log n)$ space using only three counters i, j, k.

$R(G)$ implements the Floyd-Warshall algorithm for REACHABILITY.

$R(G)$ is a monotone circuit (no NOT gates).

Slide 12

Example. CIRCUIT SAT is reducible to SAT.

Given a Boolean circuit C, the reduction construct a conjunction of clause $R(C)$ such that C is satisfiable iff $R(C)$ is satisfiable.

$R(C)$ contains all variables of C and a variable g for each gate g of C.

$R(C)$ is a conjunction of the following clauses for each gate g in C:

1. If g is variable gate x: $(g \vee \neg x), (\neg g \vee x)$
2. If g is **true** gate: g
3. If g is **false** gate: $\neg g$
4. If g is NOT gate with predecessor h: $(\neg g \vee \neg h), (g \vee h)$

Slide 13

5. If g is AND gate with predecessors h, h':
   $(\neg g \vee h), (\neg g \vee h'), (g \vee \neg h \vee \neg h')$
6. If g is OR gate with predecessors h, h':
   $(\neg g \vee h \vee h'), (g \vee \neg h'), (g \vee \neg h)$
7. If g is the output gate: g

(i) Now C is satisfiable iff $R(C)$ is satisfiable and

(ii) $R(C)$ can be computed in $\log n$ space.

Slide 14

Example. CIRCUIT VALUE is reducible to CIRCUIT SAT.

This is reduction by generalization, i.e., CIRCUIT VALUE is a special case of CIRCUIT SAT:

all inputs of CIRCUIT VALUE are inputs of CIRCUIT SAT and for those inputs CIRCUIT VALUE and CIRCUIT SAT give the same answers.

Or CIRCUIT SAT is a generalization of CIRCUIT VALUE.

- There are chains of reductions, i.e., REACHABILITY $\leq_{\mathrm{L}}$ CIRCUIT VALUE $\leq_{\mathrm{L}}$ CIRCUIT SAT $\leq_{\mathrm{L}}$ SAT
- But do reductions compose, i.e., is the ordering transitive?
  For instance, does REACHABILITY $\leq_{\mathrm{L}}$ SAT hold?

Slide 15

Proposition. If R is a reduction from language L_1 to L_2 and R' is a reduction from language L_2 to L_3, then the composition $R \cdot R'$ is a reduction from L_1 to L_3.

Proof.

- As R, R' are reductions, $x \in L_1$ iff $R'(R(x)) \in L_3$.
- To construct a machine M to compute the composition $R \cdot R'$ in space $\mathcal{O}(\log n)$ from machines M_R and $M_{R'}$ requires some care as the intermediate result computed by M_R cannot be stored (could be much longer than $\log n$).
- A solution: simulate $M_{R'}$ on input $R(x)$ by remembering the cursor position i of the input string of $M_{R'}$ (which is the output string of M_R). Only the index i is stored (in binary) and the symbol currently scanned but not the whole string.

Slide 16

- Initially $i = 1$ and it is easy to simulate the first move of $M_{R'}$ (scanning $\triangleright$).
- If $M_{R'}$ moves right, simulate M_R to generate the next output symbol and increment i by one.
- If $M_{R'}$ moves left, decrement i by one and run M_R on x *from the beginning*, counting symbols output and stopping when the ith symbol is output.
- Space needed is $\mathcal{O}(\log n)$:
  space for simulating M_R on x: $\mathcal{O}(\log n)$ $(n = |x|)$
  space for simulating $M_{R'}$ on $R(x)$: $\mathcal{O}(\log n)$ as $|R(x)| = \mathcal{O}(n^k)$
  space for bookkeeping on output of M_R on x: $\mathcal{O}(\log n)$ as $|R(x)| = \mathcal{O}(n^k)$ (we need only indices stored in binary).

Slide 17

Completeness

As reducibility is transitive, it orders problems w.r.t. to their difficulty.

Maximal elements in this order are particularly interesting.

Definition. Let $\mathcal{C}$ be a complexity class and let L be a language in $\mathcal{C}$. We say that L is $\mathcal{C}$-complete if any language $L' \in \mathcal{C}$ is reducible to L.

(Often a term $\mathcal{C}$-hardness is used for a language L if any language $L' \in \mathcal{C}$ is reducible to L but it is not know whether $L \in \mathcal{C}$ holds.

The main complexity classes ($\mathbf{P}, \mathbf{NP}, \mathbf{PSPACE}, \mathbf{NL}, \ldots$) have natural complete problems (as we shall see).

Slide 18

- Complete problems are a central concept and methodological tool in complexity theory.
- The complexity of a problem is categorized by showing that it is complete for a complexity class.
- Complete problems capture the essence of a class.
- Complete problems can be used as a negative complexity results:
  A complete problem is the least likely among all problems in $\mathcal{C}$ to belong to a weaker class $\mathcal{C}' \subseteq \mathcal{C}$.
  (it it does, then the whole class $\mathcal{C}$ coincides with the weaker class $\mathcal{C}'$ as long as $\mathcal{C}'$ is closed under reductions.)

Slide 19

A class $\mathcal{C}'$ is closed under reductions if whenever L is reducible to L' and $L' \in \mathcal{C}'$, $L \in \mathcal{C}'$.

Proposition. $\mathbf{P}, \mathbf{NP}, \mathbf{coNP}, \mathbf{L}, \mathbf{NL}, \mathbf{PSPACE}, \mathbf{EXP}$ are all closed under reductions.

Complete problems are a tool for showing that complexity classes coincide.

Proposition. If two classes $\mathcal{C}$ and $\mathcal{C}'$ are both closed under reductions and there is a language L which is complete for both $\mathcal{C}$ and $\mathcal{C}'$, then $\mathcal{C} = \mathcal{C}'$.

Proof. Since L is complete $\mathcal{C}$, all languages in $\mathcal{C}$ reduce to $L \in \mathcal{C}'$. As $\mathcal{C}'$ is closed under reductions, $\mathcal{C} \subseteq \mathcal{C}'$. The other direction by symmetry.

Slide 20

Table method

- How to establish that a problem is a complete one for a class?
- Finding the first is the most problematic
  (then things become more straightforward as we shall see).
- To establish the first complete problem we need capture in a problem the essence of the computation mode and resource bound for the class in question.
- Below we do this for $\mathbf{P}$ and $\mathbf{NP}$ using the table method. Logic plays a major role in this.

Slide 21

- Consider a polynomial time TM $M = (K, \Sigma, \delta, s)$ deciding a language L.
- Its computation on input x can be thought of as a $|x|^k \times |x|^k$ **computation table** where $|x|^k$ is the time bound for M.
- Each row in the table is a time step of the computation (ranging from 0 to $|x|^k - 1$).
  Each column is a position in the string (same range)
- Hence, the entry (i, j) $(T_{i,j})$ in the table represents the contents of position j of the string of M at time i (after i steps of M on x).

Slide 22

- Some standardizing assumptions: (i) M has one string.
  (ii) M halts on any input x after at most $|x|^k - 2$ steps .
  (iii) Strings in the table are padded with ⊔s to be of same length.
  (iv) If at time i the state is q and the cursor is scanning jth symbol σ, then entry $T_{i,j}$ is σ_q except for "yes"("no") for which the entry is "yes"("no"), respectively.
  (v) The cursor starts not from ▷ but at the first symbol of input.
  (vi) The cursor never visits the leftmost ▷ (by merging two moves if M would be visiting the leftmost ▷). Hence, the first symbol of each row is always ▷ ($\triangleright_q$ is not possible).
  (vii) If M halts before its time bound $|x|^k$ expires (one of "yes"/"no" appear on a row before the last) then all subsequent rows will be identical to that one.
- The table is said to be accepting iff $T_{|x|^k-1,j}$ = "yes" for some j.

Slide 23

Proposition. M accepts x iff the computation table of M on input x is accepting.

Any deterministic polynomial time computation can captured as a problem of determining the value of a Boolean circuit!

Theorem. CIRCUIT VALUE is $\mathbf{P}$-complete.
Proof.

- As CIRCUIT VALUE $\in \mathbf{P}$, to establish $\mathbf{P}$-completeness it is enough to show that for every language $L \in \mathbf{P}$, there is a reduction R from L to CIRCUIT VALUE.
- For an input x, $R(x)$ is a variable-free circuit such that $x \in L$ iff the value of $R(x)$ is **true**.
- Let M be a TM that decides L in time n^k and consider the computation table T of M on input x.

Slide 24

- When $i = 0$ or $j = 0$ or $j = |x|^k - 1$, the value of $T_{i,j}$ is known a priori: in the first case x or ⊔s, in the second ▷, and ⊔ in the third.
- Any other entry $T_{i,j}$ depends only on the contents of the same position or adjacent positions at time $i - 1$:
  $T_{i,j}$ depends only on $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$
- Let Γ denote the set of all symbols appearing in the table.
  Encode each symbol $\sigma \in \Gamma$ as a vector $(s_1, s_2, \ldots, s_m)$ where $s_1, s_2, \ldots, s_m \in \{0, 1\}$ and $m = \lceil \log |\Gamma| \rceil$.
- The computation table can be thought of as a table of binary entries $S_{i,j,l}$ with $0 \leq i, j \leq n^k - 1$, $1 \leq l \leq m$.
- Each $S_{i,j,l}$ depends only on $3m$ entries
  $S_{i-1,j-1,l'}, S_{i-1,j,l'}, S_{i-1,j+1,l'}$ where $1 \leq l' \leq m$

Slide 25

- There are Boolean functions $F_1, \ldots, F_m$ with $3m$ inputs each such that for all $i, j > 0$

  $S_{i,j,l} = F_l(S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,m}, S_{i-1,j,1}, \ldots S_{i-1,j+1,m})$
- Since every Boolean function can be represented by a Boolean circuit, there is a Boolean circuit C with $3m$ inputs and m outputs that computes the binary encoding of $T_{i,j}$ given the binary encodings of $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$ for all $i = 1, \ldots |x|^k - 1$ and for all $j = 1, \ldots |x|^k - 1$.
- C depends only on M and has a fixed constant size independent of the length of input x.

Slide 26

- Reduction $R(x)$ consists of $(|x|^k - 1) \times (|x|^k - 2)$ copies of circuit C one for each entry $T_{i,j}$ that is not on the top row or the two extreme columns (call this $C_{i,j}$)

  For $i \geq 1$, the input gates of $C_{i,j}$ are identified by the output gates of $C_{i-1,j-1}, C_{i-1,j}, C_{i-1,j+1}$.

  The sorts ($\mathbf{true}$/$\mathbf{false}$) of the input gates of $R(x)$ correspond to the known values of the first row and the first and last column.

  The output gate of $R(x)$ is the first output of $C_{|x|^k-1,1}$ (assuming that M halts always with cursor in the second string position and the first bit of "yes" is 1 and that of "no" is 0).

Slide 27

- Value of $R(x)$ is $\mathbf{true}$ iff $x \in L$

  Suppose value of $R(x)$ is $\mathbf{true}$. By induction on i it can be seen that the values of output of $C_{i,j}$ give the binary encoding of the computation table of M. As $R(x)$ is $\mathbf{true}$, then the entry $T_{|x|^k-1,1}$ is "yes". Hence, the table is accepting and so is M implying $x \in L$.

  If $x \in L$, then the table is accepting and thus the value of $R(x)$ is $\mathbf{true}$.
- $R(x)$ can be computed in logarithmic space:

  Input gates can be constructed by counting up to $|x|^k$ and inspecting input x ($\mathcal{O}(\log n)$ space).

  Other gates can be generated by manipulating indices in $\mathcal{O}(\log n)$ space as the size of C is fixed and independent of $|x|$.

Slide 28

- Note that NOT gates can be eliminated from variable-free circuits:

  Move NOTs downwards by applying De Morgan's laws until input gates are reached where $\neg\mathbf{true}$ is changed to $\mathbf{false}$ and vice versa.
- The result is a monotone Boolean function.

Corollary. MONOTONE CIRCUIT VALUE is $\mathbf{P}$-complete.

Corollary. HORNSAT is $\mathbf{P}$-complete.

Slide 29

Any nondeterministic polynomial time computation can captured as a circuit satisfiability problem!

Theorem. (Cook's theorem)
CIRCUIT SAT is **NP**-complete.

Proof.

- CIRCUIT SAT is in **NP**.
- Let $L \in \mathbf{NP}$. We describe a reduction R which for each string x construct a Boolean circuit $R(x)$ such that $x \in L$ iff $R(x)$ is satisfiable.
- Let M be a NTM that decides L in time n^k.
- Assume that M has a single string.

Slide 30

- We can assume that M has at each step exactly two nondeterministic choices:
  When M has $m > 2$ choices, these can be modified to binary choices with $m-2$ new states.
  For $m = 1$, we consider that choices coincide and for $m = 0$, we change nothing.
- Hence, M has at each step exactly two nondeterministic choices 0 and 1. So a sequence of nondeterministic choice is a bitstring $(c_1, c_2, \ldots, c_{|x|^k-1}) \in \{0,1\}^{|x|^k-1}$.
- If we fix the sequence of choices $\mathbf{c} = (c_1, c_2, \ldots, c_{|x|^k-1})$, then the computation is effectively deterministic and we can defined the computation table $T(M, x, \mathbf{c})$ corresponding to the machine M, input x, and choices $\mathbf{c}$.

Slide 31

- The top row and extreme columns are predetermined.
- All other entries $T_{i,j}$ depend only on $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$ and the *choice* c_{i-1} *at the previous step*.
- Now there is a Boolean circuit C with $3m+1$ inputs and m outputs that computes the binary encoding of $T_{i,j}$ given the binary encodings of $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$ and the previous choice c_{i-1}.
- $R(x)$ can be computed in logarithmic space as C has a fixed constant size independent of $|x|$.
- $R(x)$ is satisfiable iff there is a sequence of choices $\mathbf{c} = (c_1, c_2, \ldots, c_{|x|^k-1})$ such that the computation table is accepting iff $x \in L$.

Slide 32

Actually the following result is often called the Cook's theorem.

Corollary. SAT is **NP**-complete.

Proof. Let $L \in \mathbf{NP}$. Hence, L is reducible to CIRCUIT SAT as CIRCUIT SAT is **NP**-complete. But CIRCUIT SAT is reducible to SAT. Hence, L is reducible to SAT (reductions compose). As SAT $\in \mathbf{NP}$, SAT is **NP**-complete.