Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 7: Reductions and Completeness

Aalto University
School of Science
Department of Computer Science

Spring 2020

Topics

- Reductions between problems
- Examples of reductions
- Composing reductions
- Completeness and hard problems
- The computation table method
- Computation as a Boolean circuit
- Capturing nondeterministic computation

(C. Papadimitriou: *Computational Complexity*, Chapters 8.1–8.2)

8.1 Reductions between Problems

- A complexity class is an infinite family of languages ($\sim$ decision problems) determined by some complexity resource bound.

Example

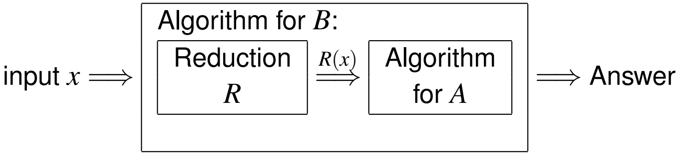
The class $\mathbf{NP}$ contains languages such as TSP(D), SAT, HORNSAT, REACHABILITY, ...

- Not all decision problems seem to be equally hard to solve.
- An ordering of problems by their computational difficulty is provided by the notion of a *reduction*:

If B reduces to A, then B is at most as hard as A.

Recap on reductions

- A *reduction* from a problem B to a problem A is an algorithm R that transforms any instance x of B to an equivalent instance $y = R(x)$ of A.
- Here "equivalent" means that the "yes"/"no" answer to $R(x)$ considered as an instance of A is the correct answer to x as an instance of B, i.e., *$x \in B$ iff $R(x) \in A$*.
- To solve B on input x, we can compute $R(x)$ and solve A on it:



☞ In a sense, R transforms problem B into a *special case* of problem A, which suggests that *B is not harder A*, or *A is at least as hard as B*.

Resource-limited reductions

- A reducibility relation yields a reasonable notion of "B is not harder than A" only when it is easier to compute the reduction R than to solve B or A directly.
- Some resource-limited reduction notions:
  – Cook reductions (polynomial-time "oracle-TM" reductions)
  – Karp reductions (polynomial-time many-one reductions)
  – Log-space many-one reductions (used here)

Definition (8.1)

A language L_1 is *log-space reducible* to language L_2 (denoted $L_1 \leq_m^{\log} L_2$ or $L_1 \leq_{\mathrm{L}} L_2$) if there is a function R from strings to strings computable by a deterministic Turing machine in space $O(\log n)$ such that for all strings x,

$$x \in L_1 \text{ iff } R(x) \in L_2.$$

☞ By a "reduction" we will henceforth mean a log-space reduction, unless otherwise indicated.

Time efficiency of reductions

Proposition (8.1)

If R is a reduction computed by a deterministic TM M, then for all inputs x, M halts after a polynomial number of steps. (I.e. a log-space reduction is also a polynomial-time reduction.)

Proof sketch

- As M works in space $O(\log n)$, there are $O(nc^{\log n})$ 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

$$c_1 nc^{\log n} = c_1 nn^{\log c} = O(n^k)$$

  steps for some k.

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

8.2 Examples of Reductions

We will consider a number of reductions, viz.

1. from HAMILTON PATH to SAT,
2. from REACHABILITY to CIRCUIT VALUE,
3. from CIRCUIT SAT to SAT, and
4. from CIRCUIT VALUE to CIRCUIT SAT.

In each case, we present a reduction R from the former language (say L_1) to the latter language (say L_2) such that for every string x based on the alphabet of L_1,

(i) $x \in L_1$ iff $R(x) \in L_2$ and

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

(However we will in most cases not check the space bound condition in detail.)

8.2.1 Reducing HAMILTON PATH to SAT

The problem HAMILTON PATH is defined as follows:

Definition (8.2)

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

- To show that SAT is at least as hard as HAMILTON PATH we must establish a reduction R from HAMILTON PATH to SAT.
- For a graph G, the outcome $R(G)$ is a conjunction of clauses such that G has a Hamilton path iff $R(G)$ is satisfiable.
- Suppose G has n vertices, $1, 2, \ldots, n$.
- Then $R(G)$ has n^2 Boolean variables x_{ij} where $1 \leq i, j \leq n$ and x_{ij} denotes that the ith vertex on the path is j.

Reducing a graph G to a CNF Boolean formula $R(G)$

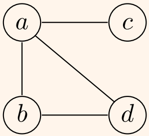
For a graph G with n vertices, the reduction mapping R produces a formula $R(G)$ that is the conjunction of the following clauses:

1. For each vertex j: $x_{1j} \vee \cdots \vee x_{nj}$ (vertex j appears on the path).
2. For all j, i, k where $i \neq k$: $\neg x_{ij} \vee \neg x_{kj}$
   (vertex j cannot be the ith and kth vertex simultaneously).
3. For all i: $x_{i1} \vee \cdots \vee x_{in}$ (some vertex is the ith vertex).
4. For all i, j, k where $j \neq k$: $\neg x_{ij} \vee \neg x_{ik}$
   (no two vertices can be ith simultaneously).
5. For each pair (i, j) where $\{i, j\}$ is **not** an edge in G and for all $k = 1, \ldots, n-1$: $\neg x_{ki} \vee \neg x_{(k+1)j}$
   (vertex j cannot come right after vertex i in the path).

Example

For readability, vertices are here named with $a, \ldots, d$.

The graph G



The CNF formula $R(G)$:

$$(x_{1,a} \vee x_{2,a} \vee x_{3,a} \vee x_{4,a}) \wedge \ldots \wedge (x_{1,d} \vee x_{2,d} \vee x_{3,d} \vee x_{4,d}) \wedge$$
$$(\neg x_{1,a} \vee \neg x_{2,a}) \wedge \ldots \wedge (\neg x_{3,a} \vee \neg x_{4,a}) \wedge$$
$$(\neg x_{1,b} \vee \neg x_{2,b}) \wedge \ldots \wedge \ldots \wedge (\neg x_{3,d} \vee \neg x_{4,d}) \wedge$$
$$(x_{1,a} \vee x_{1,b} \vee x_{1,c} \vee x_{1,d}) \wedge \ldots \wedge (x_{4,a} \vee x_{4,b} \vee x_{4,c} \vee x_{4,d}) \wedge$$
$$(\neg x_{1,a} \vee \neg x_{1,b}) \wedge \ldots \wedge (\neg x_{1,c} \vee \neg x_{1,d}) \wedge$$
$$(\neg x_{2,a} \vee \neg x_{2,b}) \wedge \ldots \wedge \ldots \wedge (\neg x_{4,c} \vee \neg x_{4,d}) \wedge$$
$$(\neg x_{1,b} \vee \neg x_{2,c}) \wedge (\neg x_{1,c} \vee \neg x_{2,d}) \wedge \ldots \wedge$$
$$(\neg x_{3,b} \vee \neg x_{4,c}) \wedge (\neg x_{3,c} \vee \neg x_{4,d})$$

Path:

$c - a - d - b$

Satisfying truth assignment:

$x_{1,c}, x_{2,a}, x_{3,d}, x_{4,b}$ are true, all other variables false

Delete the edge $\{b, d\}$ in G and add the corresponding clauses in $R(G)$ to get a “no” instance.

Proof of reduction condition

($\Rightarrow$) Let G have a Hamilton path $(\pi(1),\ldots,\pi(n))$ where π is a permutation of the vertices. Then $R(G)$ is satisfied by a truth assignment T defined by $T(x_{ij}) = \mathbf{true}$ if $\pi(i) = j$ else $T(x_{ij}) = \mathbf{false}$.

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

- By clauses (1,2) for every vertex j there is unique i such that $T(x_{ij}) = \mathbf{true}$.
- By clauses (3,4) for every i there is unique vertex j such that $T(x_{ij}) = \mathbf{true}$.
- Thus, T represents a permutation $\pi(1),\ldots,\pi(n)$ of the vertices where $\pi(i) = j$ iff $T(x_{ij}) = \mathbf{true}$.
- By clauses (5) for all k, there is an edge $\{\pi(k),\pi(k+1)\}$ in G. Hence $(\pi(1),\ldots,\pi(n))$ is a Hamilton path.

Proof of logarithmic space bound

We show that $R(G)$ can be computed in space $O(\log n)$.
Given G as an input, a TM M outputs $R(G)$ as follows:

- M first outputs clauses (1-4), which do not depend on G, one by one using three counters i, j, k.
- Each counter is represented in binary within $\log n$ space.
- M outputs clauses (5) by considering each pair (i, j) in turn: If $\{i, j\}$ is not an edge in G (M checks this first), then M outputs clauses $\neg x_{ki} \vee \neg x_{(k+1)j}$ one by one for all $k = 1, \ldots, n-1$.
- Again space is needed only for the counters i, j, k, i.e. at most $3 \log n$ in total.

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

8.2.2 Reducing REACHABILITY to CIRCUIT VALUE

We design a reduction mapping R that for a graph G gives a variable-free circuit $R(G)$ such that

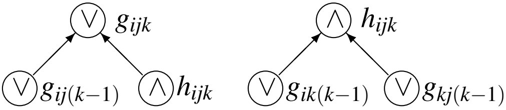
the value of the circuit $R(G)$ is $\mathbf{true}$ iff
there is a path from 1 to n in G.

- The gates of $R(G)$ are of the following two forms:
  – g_{ijk} with $1 \leq i,j \leq n$ and $0 \leq k \leq n$ and
  – h_{ijk} with $1 \leq i,j,k \leq n$.
- Here g_{ijk} is intended to be $\mathbf{true}$ iff there is a path in G from i to j *not using any intermediate vertex bigger than* k;
- h_{ijk} is intended to be $\mathbf{true}$ iff there is a path in G from i to j not using any intermediate vertex bigger than k *but using* k.

The structure of the circuit $R(G)$

Given a graph G, $R(G)$ consists of the following gates:

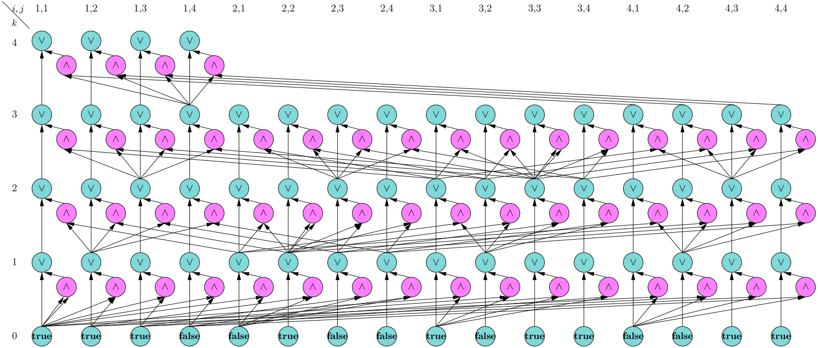
- For $k = 0$, every gate g_{ijk} is an input gate in $R(G)$ with a fixed truth value.
  g_{ij0} is a **true** gate if $i = j$ or (i,j) is an edge in G and a **false** gate otherwise.
- For $k = 1, 2, \ldots, n$, the following gates are in $R(G)$:



- Gate g_{1nn} is the output of $R(G)$.

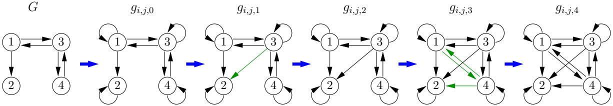
☞ The circuit $R(G)$ is acyclic and variable-free.

The structure of the circuit $R(G)$: example



☞ Warshall's algorithm for reflexive and transitive closure of digraphs:

$g_{i,j,0} := (i = j) \vee G(i,j)$ and $g_{i,j,k} := g_{i,j,k-1} \vee \left(g_{i,k,k-1} \wedge g_{k,j,k-1}\right)$



Correct value assignment for h_{ijk} and g_{ijk}

We show that the gates h_{ijk} and g_{ijk} satisfy their intended meaning by induction on $k = 0, 1, \ldots, n$.

- The base case $k = 0$ is covered by the definition of input gates g_{ij0}.
- For $k > 0$, the circuit assigns $h_{ijk} = g_{ik(k-1)} \wedge g_{kj(k-1)}$.
  By the inductive hypothesis (IH) h_{ijk} is **true** *iff* there is a path from i to k and from k to j not using any intermediate vertex bigger than $k - 1$ *iff* there is a path from i to j not using any intermediate vertex bigger than k but going through k.
- For $k > 0$, the circuit assigns $g_{ijk} = g_{ij(k-1)} \vee h_{ijk}$.
  By IH g_{ijk} is **true** *iff* there is a path from i to j not using any vertex bigger than $k - 1$; or a path not using any vertex bigger than k but going through k *iff* there is a path from i to j not using any intermediate vertex bigger than k.

Correctness of the reduction

- In fact, the circuit $R(G)$ implements the Floyd-Warshall algorithm for REACHABILITY.
- Given a graph G with n vertices, the value of $R(G)$ is **true** iff
  g_{1nn} is **true** iff
  there is a path from 1 to n in G without any intermediate vertices bigger than n iff
  there is a path from 1 to n in G.
- Given a graph G with n vertices, the circuit $R(G)$ can be computed in $O(\log n)$ space using only three counters i, j, k.
- Note that $R(G)$ is a *monotone* circuit, i.e. it has no NOT gates.

8.2.3 Reducing CIRCUIT SAT to SAT

We design a reduction mapping R that given a Boolean circuit C produces a CNF Boolean formula $R(C)$ such that C is satisfiable, i.e., has a truth assignment T such that $T(C) = \mathbf{true}$ iff $R(C)$ is satisfiable.

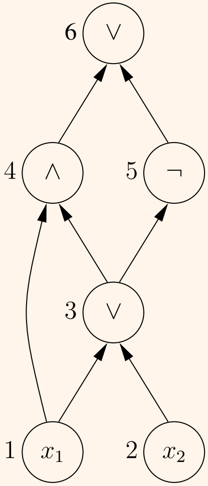
The formula $R(C)$ uses all variables of C and it includes for each gate g of C a new variable g and the following clauses:

1. If g is a variable gate x: $(g \vee \neg x), (\neg g \vee x)$. $[g \leftrightarrow x]$
2. If g is a **true** (resp. **false**) gate: g (resp. $\neg g$).
3. If g is a NOT gate with a predecessor h: $(\neg g \vee \neg h), (g \vee h)$. $[g \leftrightarrow \neg h]$
4. If g is an AND gate with predecessors h, h':
   $(\neg g \vee h), (\neg g \vee h'), (g \vee \neg h \vee \neg h')$. $[g \leftrightarrow (h \wedge h')]$
5. If g is an OR gate with predecessors h, h':
   $(\neg g \vee h \vee h'), (g \vee \neg h'), (g \vee \neg h)$. $[g \leftrightarrow (h \vee h')]$
6. If g is also the output gate: g.

We skip the correctness proof which is straightforward.

Example

Boolean circuit C:



The corresponding CNF formula $R(C)$:

$(g_6) \wedge$
$(\neg g_6 \vee g_4 \vee g_5) \wedge (g_6 \vee \neg g_4) \wedge (g_6 \vee \neg g_5) \wedge$
$(\neg g_5 \vee \neg g_3) \wedge (g_5 \vee g_3) \wedge$
$(\neg g_4 \vee g_1) \wedge (\neg g_4 \vee g_3) \wedge (g_4 \vee \neg g_1 \vee \neg g_3) \wedge$
$(\neg g_3 \vee g_1 \vee g_2) \wedge (g_3 \vee \neg g_1) \wedge (g_3 \vee \neg g_2) \wedge$
$(g_2 \vee \neg x_2) \wedge (\neg g_2 \vee x_2) \wedge$
$(g_1 \vee \neg x_1) \wedge (\neg g_1 \vee x_1)$

8.2.4 Reducing CIRCUIT VALUE to CIRCUIT SAT

- CIRCUIT VALUE is a special case of CIRCUIT SAT: all instances of CIRCUIT VALUE are also instances of CIRCUIT SAT, and for those instances the solutions to CIRCUIT VALUE and CIRCUIT SAT coincide.
- The identity function $I(x) = x$ thus gives a trivial reduction mapping from CIRCUIT VALUE to CIRCUIT SAT.

8.3 Composing Reductions

- So far, we have established a chain of reductions, i.e. REACHABILITY $\leq_{\mathrm{L}}$ CIRCUIT VALUE $\leq_{\mathrm{L}}$ CIRCUIT SAT $\leq_{\mathrm{L}}$ SAT.
- It is natural to expect that reductions compose, i.e. that the $\leq_{\mathrm{L}}$ relation is transitive, and we could deduce e.g. that REACHABILITY $\leq_{\mathrm{L}}$ SAT.
- Establishing this requires, however, a small proof to check that the resource bounds are maintained.

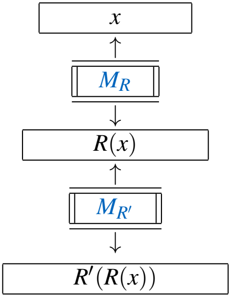
Proposition (8.2)

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.

- As R, R' are reductions, $x \in L_1$ iff $R(x) \in L_2$ iff $R'(R(x)) \in L_3$.
- It remains to show that $R'(R(x))$ can be computed in $O(\log n)$ space where $n = |x|$.

Logarithmic space bound

- To construct a machine M for the composition $R \cdot R'$ working in space $O(\log n)$ requires care as the intermediate result computed by M_R cannot be stored. (It is possibly longer than $\log n$.)
- A solution: simulate $M_{R'}$ on input $R(x)$ by remembering the cursor position i on the input tape of $M_{R'}$, which is the output tape of M_R. Only the index i is stored (in binary) and the symbol currently scanned but not the whole string.



Logarithmic space bound – cont'd

- 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 the symbols output and stopping when the ith symbol is output.
- The space required for simulating M_R on x as well as $M_{R'}$ on $R(x)$ is $O(\log n)$ where $n = |x|$.
- The space required for bookkeeping the output $R(x)$ of M_R on x is $O(\log n)$ since $|R(x)| = O(n^k)$ and we need only an index stored in binary.

8.4 Completeness and Hard Problems

- The reducibility relation $\leq_{\mathrm{L}}$ orders problems with respect to their difficulty, as it is reflexive and transitive (a *preorder*).
- Maximal elements in this order are particularly interesting.

Definition (8.3)

Let $\mathcal{C}$ be a complexity class and let L be a language in $\mathcal{C}$. Then L is *$\mathcal{C}$-complete* if for every $L' \in \mathcal{C}$, $L' \leq_{\mathrm{L}} L$.

- A language L is called *$\mathcal{C}$-hard* if any language $L' \in \mathcal{C}$ is reducible to L (but it is not known whether $L \in \mathcal{C}$ holds).
- The main complexity classes ($\mathbf{P}, \mathbf{NP}, \mathbf{PSPACE}, \mathbf{NL}, \ldots$) all have natural complete problems (as we shall see).

The role of completeness in complexity theory

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

Closure under reductions

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

$$\text{if } L' \leq_{\mathrm{L}} L \text{ and } L \in \mathcal{C}, \text{ then } L' \in \mathcal{C}.$$

Proposition (8.4)

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

- For example, if a $\mathbf{P}$-complete problem L is in $\mathbf{NL}$, then $\mathbf{P} = \mathbf{NL}$.
  Proof. We know that $\mathbf{NL} \subseteq \mathbf{P}$, so let's establish $\mathbf{P} \subseteq \mathbf{NL}$ under the given assumption.
  Let $L' \in \mathbf{P}$. As L is $\mathbf{P}$-complete, then L' is reducible to L. Since $L \in \mathbf{NL}$ and $\mathbf{NL}$ is closed under reductions, then also $L' \in \mathbf{NL}$.
  Hence, $\mathbf{P} \subseteq \mathbf{NL}$ and $\mathbf{P} = \mathbf{NL}$.
- Similarly, if an $\mathbf{NP}$-complete problem is in $\mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.

Proving the equality of complexity classes

Proposition (8.5)

If two complexity classes $\mathcal{C}$ and $\mathcal{C}'$ are

1. *both closed under reductions and*
2. *there is a language L which is complete for $\mathcal{C}$ and $\mathcal{C}'$,*

then $\mathcal{C} = \mathcal{C}'$.

Proof

($\subseteq$) Since L is complete for $\mathcal{C}$, all languages in $\mathcal{C}$ reduce to L. As $\mathcal{C}'$ is closed under reductions and $L \in \mathcal{C}'$, $\mathcal{C} \subseteq \mathcal{C}'$.
($\supseteq$) Follows by symmetry.

8.5 The Computation Table Method

- How to establish that a problem is complete for a class?
- Finding the *first* complete problem is the biggest challenge. Then things become much more straightforward, as we shall see.
- To establish the first complete problem for a class, we need to capture in its description the essence of the computation mode and resource bound for the class in question.
- Below we do this for the classes $\mathbf{P}$ and $\mathbf{NP}$ using the so-called *computation table method*.

Computation tables

- Consider a polynomially time-bounded single-string TM $M = (K, \Sigma, \delta, s)$ deciding a language L over alphabet Σ.
- Its computation on input x can be thought of as an $|x|^k \times |x|^k$ **computation table** T, where $|x|^k$ is the time bound for M.
- Each row in the table represents a time step of the computation ranging from 0 to $|x|^k - 1$.
- Each column represents a position on M's tape (same range).
- Entry (i, j) in T, $T_{i,j}$, represents the contents of position j on M's tape at time i, i.e. after i steps of computation on input x.

Example

i/j	0	1	2	3	$\ldots$	$\lvert x\rvert^k - 1$
0	$\triangleright$	0_s	1	1	$\ldots$	$\sqcup$
1	$\triangleright$	0_q	1	1	$\ldots$	$\sqcup$
2	$\triangleright$	1	1_q	1	$\ldots$	$\sqcup$
$\vdots$	$\vdots$					
$\lvert x\rvert^k - 1$	$\triangleright$	"yes"	$\sqcup$	$\sqcup$	$\ldots$	$\sqcup$

Computation tables – cont'd

Some standardising assumptions:

- M has only one tape.
- M halts on any input x after at most $|x|^k - 2$ steps (k is chosen so that this is guaranteed for $|x| \geq 2$).
- Rows of the table are padded with $\sqcup$'s to be of same length $|x|^k$.
- If at time i the state of M is q and the cursor is scanning jth symbol σ, then the entry $T_{i,j}$ is σ_q (rather than σ); except for "yes"/"no" for which the entry is "yes"/"no".
- The cursor starts at the first symbol of the input (not at $\triangleright$).

Computation tables – cont'd

- The cursor never visits the leftmost $\triangleright$. (This can be achieved by merging two moves of M if M is about to visit the leftmost $\triangleright$.)
  ☞ The first symbol of each row is always $\triangleright$ (never $\triangleright_q$).
- If M halts before its time bound $|x|^k$ expires ($T_{i,j} =$ "yes"/"no" for some $i < |x|^k - 1$ and j), then all subsequent rows will be identical.
- The table is *accepting* iff $T_{|x|^k-1,j} =$ "yes" for some j.

Proposition (8.6)

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

8.6 Computation as a Boolean Circuit

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

Theorem (8.7)

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

- 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, the result $R(x)$ is to be a variable-free circuit such that $x \in L$ iff the value of $R(x)$ is $\mathbf{true}$.
- In the sequel, we consider a TM M deciding L in time n^k.

Reducing any $L \in \mathbf{P}$ to CIRCUIT VALUE

Consider the computation table T of M on input x:

- 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 $\sqcup$, in the second $\triangleright$, and $\sqcup$ in the third.
- Any other entry $T_{i,j}$ depends only on the contents of the same or adjacent positions $T_{i-1,j-i}$, $T_{i-1,j}$ and $T_{i-1,j+1}$ at time $i-1$:

$T_{i-1,j-1}$	$T_{i-1,j}$	$T_{i-1,j+1}$
	$T_{i,j}$	

- The idea is to encode this relationship using a Boolean circuit.

A binary encoding for T

- Let Γ denote the set of all symbols appearing in the table T. Encode each symbol $\sigma \in \Gamma$ as a bit 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 $0 \le i,j \le n^k - 1$ and $1 \le l \le m$.
- Thus, each $S_{i,j,l}$ depends only on $3m$ entries

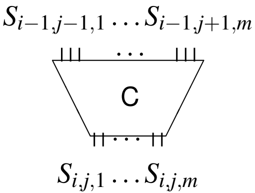
  $$S_{i-1,j-1,l'},\ S_{i-1,j,l'},\ \text{and}\ S_{i-1,j+1,l'}$$

  where $1 \le l' \le m$.
- So there are Boolean functions $F_1, \ldots, F_m$ with $3m$ inputs each such that for all $i,j > 0$,

  $$S_{i,j,l} = F_l(S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,m}, S_{i-1,j,1}, \ldots S_{i-1,j+1,m}).$$

A binary encoding for T – cont'd

- 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 $T_{i-1,j-1}$, $T_{i-1,j}$, and $T_{i-1,j+1}$ for all $i = 1, \ldots |x|^k$ and for all $j = 1, \ldots |x|^k - 2$.

- Note that C *depends only on M and has a fixed constant size* independent of the length of input x.

Definition of the reduction

- The reduction image $R(x)$ of x consists of $(|x|^k - 1) \times (|x|^k - 2)$ copies of circuit C, one for each entry $T_{i,j}$ that is not in 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 tape position and the first bit of “yes” is 1 and that of “no” is 0).

Correctness of the reduction

☞ The value of $R(x)$ is **true** iff $x \in L$:

- Suppose that the value of $R(x)$ is $\mathbf{true}$.
  – It can be shown by induction on i that the output values of $C_{i,j}$ give the binary encoding of the ith row of T.
  – 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 $x \in L$, the table is accepting and the value of $R(x)$ is $\mathbf{true}$.

☞ The circuit $R(x)$ can be computed in logarithmic space:

- Input gates can be constructed by counting up to $|x|^k$ and inspecting input x ($O(\log n)$ space).
- Other gates can be generated by manipulating indices in $O(\log n)$ space, as the size of C is fixed and independent of $|x|$.

Other $\mathbf{P}$-complete problems

- 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. Change $\neg$**true** to **false** and vice versa.
- Circuits containing only AND and OR gates (but no NOT gates) are called *monotone circuits*.
- Monotone circuits can only compute *monotone Boolean functions*. (A Boolean function is monotone if it satisfies the property: if one of the inputs changes from **false** to **true**, the value of the function cannot change from **true** to **false**.)

Corollary (8.8)

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

Corollary (8.9)

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

8.7 Capturing Nondeterministic Computation

Any *nondeterministic polynomial time computation* can be captured as a *circuit satisfiability problem*.

Theorem (8.10)

CIRCUIT SAT is $\mathbf{NP}$*-complete.*

Proof.

- CIRCUIT SAT is in $\mathbf{NP}$.
- Let $L \in \mathbf{NP}$. We'll describe a reduction R that for each string x constructs a Boolean circuit $R(x)$ such that

$$x \in L \text{ iff } R(x) \text{ is satisfiable.}$$

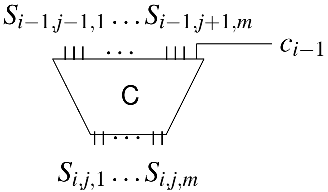
- Let M be a single-tape NTM that decides L in time n^k.

Standardising choices made by M

- Wlog assume that M has exactly two nondeterministic choices ($\delta_1, \delta_2 \in \Delta$) at each step of computation.
  (The cases that $|\Delta| > 2$ or $|\Delta| < 2$ can be avoided by adding new states to M or by assuming that choices coincide ($\delta_1 = \delta_2$).)
- Under this assumption, a sequence of nondeterministic choices $\mathbf{c}$ can be represented as a bitstring $(c_0, c_1, \ldots, c_{|x|^k-2}) \in \{0,1\}^{|x|^k-1}$.
- If we fix the sequence of choices $\mathbf{c}$, then the computation of M becomes effectively deterministic.
- Let us define the computation table $T(M, x, \mathbf{c})$ corresponding to the machine M, an input x, and a sequence of choices $\mathbf{c}$.

A binary encoding for $T(M, x, \mathbf{c})$

- The top row and extreme columns are predetermined as before.
- 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*.
- Thus, 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}.

Correctness of the reduction

- The circuit $R(x)$ is constructed as in the deterministic case but circuitry for $\mathbf{c}$ must be incorporated.
- The circuit $R(x)$ can be computed in logarithmic space as C has a fixed constant size independent of $|x|$.
- Moreover, the circuit $R(x)$ is satisfiable iff there is a sequence of choices $\mathbf{c}$ such that the computation table is accepting iff $x \in L$.

Corollary (8.11; S. Cook/L. Levin $\sim$ 1971)

SAT is $\mathbf{NP}$*-complete.*

Proof. Let $L \in \mathbf{NP}$. Then L is reducible to CIRCUIT SAT as CIRCUIT SAT is $\mathbf{NP}$-complete. But CIRCUIT SAT is reducible to SAT. Hence, L is reducible to SAT as reductions compose.
On the other hand, SAT $\in \mathbf{NP}$ so that SAT is $\mathbf{NP}$-complete.

Learning Objectives

- The idea of reducing one problem, or language, into another.
- The basic properties of $\mathbf{L}$-reductions (e.g. compositionality) and ability to construct reductions on one's own.
- The definitions of $\mathcal{C}$-hard and $\mathcal{C}$-complete problems/languages for a complexity class $\mathcal{C}$.
- Understanding the role of complete problems in complexity theory.
- Fundamental completeness results regarding CIRCUIT VALUE, HORNSAT, CIRCUIT SAT, and SAT.