REDUCTIONS AND COMPLETENESS

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

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

1. Reductions between Problems

- A complexity class is an infinite collection of languages.

  Example. The class **NP** contains languages such as TSP(D), SAT, HORNSAT, REACHABILITY, . . .
- Not all decision problems seem to be equally hard to solve; can problems be somehow ordered by difficulty?
- Such an ordering relation is definable using a notion of a *reduction*:

  A is at least as hard as B if B reduces to A.

Basic requirements for reductions

- A problem 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.
- Here equivalent means that the "yes"/"no" answer for $R(x)$ considered as A's input is the correct answer to x as an input of B, i.e., $x \in B$ *iff* $R(x) \in A$.
- To solve B on input x we need to compute $R(x)$ and solve A on it:

Algorithm for B:

input $x \Longrightarrow$ Reduction R $\overset{R(x)}{\Longrightarrow}$ Algorithm for A $\Longrightarrow$ Answer

Limiting resources in reductions

- The notion of a reduction seems reasonable to capture that A is at least as hard as B except when R is very hard to compute (e.g., when reducing TSP(D) to REACHABILITY).
- Possible limits on resources in reductions:
  - Cook reductions (polynomial-time Turing reductions)
  - Karp reductions (polynomial-time many-one reductions)
  - Log-space reductions (used here)

Definition. A language L_1 *is reducible to* L_2 ($L_1 \leq_{\mathrm{L}} L_2$) iff there is a function R from strings to strings computable by a deterministic Turing machine in space $\mathrm{O}(\log n)$ such that for all inputs x,

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

The function R is called a *reduction* from L_1 to L_2.

Time efficiency of reductions

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

Proof sketch.

- As M works in space $\mathrm{O}(\log n)$, there are $\mathrm{O}(nc^{\log n})$ 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
$$c_1 nc^{\log n} = c_1 nn^{\log c} = \mathrm{O}(n^k)$$
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. $|x|$.

2. Examples of Reductions

We will consider a number of reductions, i.e.

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 $\mathrm{O}(\log n)$ space.

Reducing HAMILTON PATH to SAT

Definition. The problem HAMILTON PATH is defined as follows:
INSTANCE: A graph G.
QUESTION: Is there a path in G that visits every node 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 nodes, $1, 2, \ldots, n$.
- Then $R(G)$ has n^2 Boolean variables x_{ij} where $1 \leq i, j \leq n$ and x_{ij} denotes that the ith node on the path is j.

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

For a graph G with n nodes, the reduction function R produces a formula $R(G)$ which 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 $i \neq k$: $\neg x_{ij} \vee \neg x_{kj}$
(node j cannot be the ith and kth node simultaneously).
3. For all i: $x_{i1} \vee \cdots \vee x_{in}$ (some node is the ith node).
4. For all i, j, k where $j \neq k$: $\neg x_{ij} \vee \neg x_{ik}$
(no two nodes can be ith simultaneously).
5. For each pair (i, j) **not** an edge in G and for all $k = 1, \ldots, n-1$:
$\neg x_{ki} \vee \neg x_{(k+1)j}$
(node j cannot come right after node i in the path).

Proof of correspondence

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

- By clauses (1,2) for every node j there is unique i such that $T(x_{ij}) = \mathbf{true}$.
- By clauses (3,4) for every i there is unique node j such that $T(x_{ij}) = \mathbf{true}$.
- Thus 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, there is an edge $(\pi(k), \pi(k+1))$ 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 $R(G)$ is satisfied by a truth assignment T defined by $T(x_{ij}) = \mathbf{true}$ if $\pi(i) = j$ else $T(x_{ij}) = \mathbf{false}$.

Proof of logarithmic space consumption

We show that $R(G)$ can be computed in space $\mathrm{O}(\log n)$.

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 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 $\mathrm{O}(\log n)$.

Reducing REACHABILITY to CIRCUIT VALUE

We devise a reduction function R which for a graph G gives a variable-free circuit $R(G)$ such that

the output of $R(G)$ is **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 $1 \leq i, j, k \leq n$.
- Now g_{ijk} is intended to be **true** iff there is a path in G from i to j not using any intermediate node bigger than k;

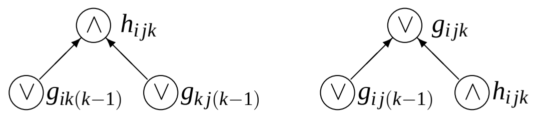
  and h_{ijk} is intended to be **true** iff there is a path in G from i to j not using any intermediate node bigger than k but using k.

The structure of the circuit $R(G)$

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

- For $k = 0$, the gate g_{ijk} is an input gate in $R(G)$:

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



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

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

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.
- 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 node bigger than $k-1$ *iff* there is a path from i to j not using any intermediate node 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 node bigger than $k-1$; or a path not using any node bigger than k but going through k *iff* there is a path from i to j not using any intermediate node bigger than k.

Correctness of the reduction

- In fact, the circuit $R(G)$ implements the Floyd-Warshall algorithm for REACHABILITY.
- 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.
- 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 (no NOT gates).

Reducing CIRCUIT SAT to SAT

We devise a reduction function R which given a Boolean circuit C produces a Boolean formula $R(C)$ in CNF such that C is satisfiable 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.

Reducing CIRCUIT VALUE to CIRCUIT SAT

- CIRCUIT VALUE is a special case of CIRCUIT SAT:

  all inputs of CIRCUIT VALUE are also inputs of CIRCUIT SAT and for those CIRCUIT VALUE and CIRCUIT SAT coincide.
- Thus CIRCUIT SAT is a generalization of CIRCUIT VALUE.
- There is a trivial reduction, i.e. identity function $I(x) = x$, from CIRCUIT VALUE to CIRCUIT SAT.

3. Composing Reductions

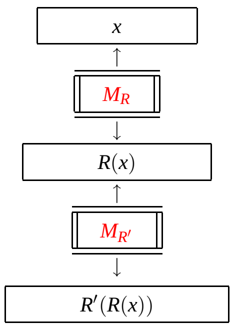
- So far, we have established a chain of reductions, i.e. REACHABILITY $\leq_{\mathrm{L}}$ CIRCUIT VALUE $\leq_{\mathrm{L}}$ CIRCUIT SAT $\leq_{\mathrm{L}}$ SAT.
- But do reductions compose, i.e., is $\leq_{\mathrm{L}}$ transitive?
  For instance, does REACHABILITY $\leq_{\mathrm{L}}$ SAT hold?

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.

- 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 $\mathrm{O}(\log n)$ space where $n = |x|$.

Logarithmic space consumption

- To construct a machine M for the composition $R \cdot R'$ working in space $\mathrm{O}(\log n)$ requires care as the intermediate result computed by M_R cannot be stored (possibly 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.



Space consumption—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 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 $\mathrm{O}(\log n)$ where $n = |x|$.
- The space needed for bookkeeping the output of M_R on x is $\mathrm{O}(\log n)$ as $|R(x)| = \mathrm{O}(n^k)$ as we need only indices stored in binary.

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. Let C be a complexity class and let L be a language in C. Then L is *C-complete* if for every $L' \in \mathrm{C}$, $L' \leq_{\mathrm{L}} L$.
- A language L is called *C-hard* if any language $L' \in \mathrm{C}$ is reducible to L but it is not known whether $L \in \mathrm{C}$ holds.
- The main complexity classes $(\mathbf{P}, \mathbf{NP}, \mathbf{PSPACE}, \mathbf{NL}, \ldots)$ 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 *categorized* 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 C to *belong to a weaker class* $\mathrm{C}' \subseteq \mathrm{C}$.

  (If it does, then the whole class C coincides with the weaker class C' as long as C' is closed under reductions; see below.)

Closure under reductions

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

  Proposition. $\mathbf{P}, \mathbf{NP}, \mathbf{coNP}, \mathbf{L}, \mathbf{NL}, \mathbf{PSPACE}, \mathbf{EXP}$ 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 $\mathbf{NL} \subseteq \mathbf{P}$.

  Let $L' \in \mathbf{P}$. As L is $\mathbf{P}$-complete, then L' is reducible to L. Since $\mathbf{NL}$ is closed under reductions, $L' \in \mathbf{NL}$. Hence, $\mathbf{P} \subseteq \mathbf{NL}$.

- Similarly, if an $\mathbf{NP}$-complete problem is in $\mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.

Proving the equality of complexity classes

Proposition. If two complexity classes C and C' are

1. both closed under reductions and
2. there is a language L which is complete for C and C',

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

Proof.

($\subseteq$) Since L is complete for C, all languages in C reduce to $L \in \mathrm{C}'$. As C' is closed under reductions, $\mathrm{C} \subseteq \mathrm{C}'$.

($\supseteq$) Follows by symmetry.

5. Table Method

- How to establish that a problem is a complete one for a class?
- Finding the first complete problem is the most problematic case (then things become more straightforward as we shall see).
- To establish the first one 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 the classes $\mathbf{P}$ and $\mathbf{NP}$ using the so-called *table method* in which logic plays a major role.

Computation table

- Consider a polynomial time TM $M = (K, \Sigma, \delta, s)$ deciding a language L based on Σ.
- Its computation on input x can be thought of as a $|x|^k \times |x|^k$ **computation table** T 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).
- The entry (i, j) in T, (i.e. $T_{i,j}$) represents the contents of position j of the string of M at time i (after i steps of M on 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 table—cont'd

Some standardizing assumptions are made:

- M has only one string.
- M halts on any input x after at most $|x|^k - 2$ steps (k is chosen so that this is guaranteed for $|x| \geq 2$).
- Strings in the table are padded with $\sqcup$s to be of same length ($|x|^k$).
- If at time i the state 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 table—cont'd

- The cursor never visits the leftmost $\triangleright$ which is 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.

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

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. 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 **true**.
- In the sequel, we consider a TM M deciding L in time n^k.

Reduction from $L \in \mathbf{P}$ 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 $\sqcup$s, 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 $0 \leq i, j \leq n^k - 1$ and $1 \leq l \leq m$.
- Thus each $S_{i,j,l}$ depends only on $3m$ entries

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

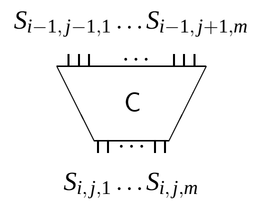
where $1 \leq l' \leq m$

- 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.

The definition of the reduction

- The reduction $R(x)$ 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 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 (**true**/**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).

Correctness of the reduction

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

  Suppose that the value of $R(x)$ is **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 **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 **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 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 where $\neg$**true** is changed to **false** and vice versa.
- We call circuits containing only AND and OR gates (but no NOT gates) *monotone circuits*.
- Monotone circuits can only compute *monotone Boolean functions*.

  (A Boolean function is monotone if it satisfies the following property: if one of the inputs changes from **false** to **true**, the value of the function cannot changes from **true** to **false**.)

Corollary. MONOTONE CIRCUIT VALUE is **P**-complete.

Corollary. HORNSAT is **P**-complete.

(See tutorials.)

7. Capturing nondeterministic computation

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

Theorem. CIRCUIT SAT is **NP**-complete.

Proof.

- CIRCUIT SAT is in **NP**.
- Let $L \in$ **NP**. We'll describe a reduction R which 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-string NTM that decides L in time n^k.

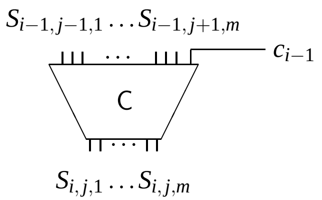
Standardizing choices made by M

- It is assumed 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 bit string $(c_0, c_1, \ldots, c_{|x|^k-2}) \in \{0,1\}^{|x|^k-1}$.
- 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*. 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. (Cook's theorem) SAT is $\mathbf{NP}$-complete.

Proof. Let $L \in \mathbf{NP}$. Hence, 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.
- You should know the basic properties of $\mathbf{L}$-reductions (e.g. compositionality) and be able to construct reductions on your own.
- The definitions of C-hard and C-complete problems/languages for a complexity class C.
- Understanding the role of complete problems in complexity theory.
- Fundamental completeness results regarding CIRCUIT VALUE, HORNSAT, CIRCUIT SAT, and SAT.