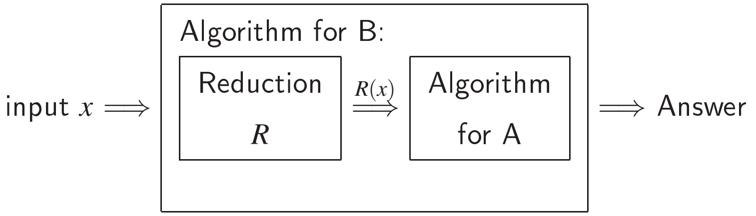
REDUCTIONS AND COMPLETENESS

- A complexity class is an infinite collection of languages.
  NP contains TSP(D), SAT, HORNSAT, REACHABILITY, ...
- Not all problems seem to be equally hard to solve; can problems be ordered by difficulty?
- Such an ordering relation can be defined using a notion of *reduction*:
  A is at least as hard as B if B reduces to A.

Reductions

- 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 the correct answer to x as an input of B, i.e., $x \in B$ *iff* $R(x) \in A$.
  A is at least as hard as B:



Reductions—cont'd

- 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 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 $O(\log n)$ 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.

Reductions—cont'd

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.

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

Example reductions

Example. HAMILTON PATH is reducible to SAT (SAT is at least as hard as HAMILTON PATH).

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:

For a graph G, $R(G)$ 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, $x_{ij}, 1 \leq i, j \leq n$

- x_{ij} : 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)
  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 $k = 1, \ldots, n-1$: $\neg x_{ki} \vee \neg x_{(k+1)j}$
     (j cannot come 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 $O(\log n)$
- (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.

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

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

satisfies all the clauses.

- (ii) 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) 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 $O(\log n)$.

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 **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 $0 \leq k \leq n$

  h_{ijk} with $1 \leq i, j, k \leq n$

- g_{ijk} is **true**: there is a path in G from i to j not using any intermediate node bigger than k.

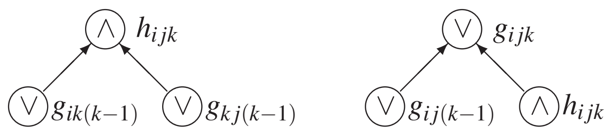
  h_{ijk} is **true**: there is a path in G from i to j not using any intermediate node bigger than k but using k.

- $R(G)$ is the following circuit:

  For $k = 0$, g_{ijk} is an input gate:

  g_{ij0} is a **true** gate if $i = j$ or (i, j) is an edge in G else it is a **false** gate.

  For $k = 1, 2, \ldots, n$:



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

- $R(G)$ forms an acyclic circuit and it is variable-free.
- 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 $k > 0$, $h_{ijk} = g_{ik(k-1)} \wedge g_{kj(k-1)}$. 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 $k-1$, i.e., there is a path from i to j not using any interm. node bigger than k but going through k.
  - $g_{ijk} = g_{ij(k-1)} \vee h_{ijk}$. 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 interm. node bigger than k.

- 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 $O(\log n)$ 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).

Example. CIRCUIT SAT is reducible to SAT.

Given a Boolean circuit C, the reduction constructs 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: $(g \vee \neg x), (\neg g \vee x)$ $[g \leftrightarrow x]$
2. If g is **true** gate: g
3. If g is **false** gate: $\neg g$
4. If g is NOT gate with predecessor h:
   $(\neg g \vee \neg h), (g \vee h)$ $[g \leftrightarrow \neg h]$

5. If g is AND gate with predecessors h, h':
   $(\neg g \vee h), (\neg g \vee h'), (g \vee \neg h \vee \neg h')$ $[g \leftrightarrow (h \wedge h')]$
6. If g is OR gate with predecessors h, h':
   $(\neg g \vee h \vee h'), (g \vee \neg h'), (g \vee \neg h)$ $[g \leftrightarrow (h \vee h')]$
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.

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.

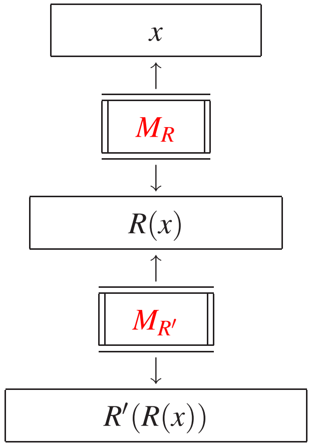
Are reductions transitive?

- There is 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 the ordering 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.

Proof.

- As R, R' are reductions, $x \in L_1$ iff $R(x) \in L_2$ iff $R'(R(x)) \in L_3$.
- 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 (possibly longer than $\log n$).



(Proof—cont'd)

- 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.
- 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.

(Proof—cont'd)

- Space needed is $O(\log n)$:

  space for simulating M_R on x: $O(\log n)$ $(n = |x|)$

  space for simulating $M_{R'}$ on $R(x)$: $O(\log n)$

  space for bookkeeping of output of M_R on x: $O(\log n)$ as $|R(x)| = O(n^k)$ (we need only indices stored in binary).

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 known 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).

Completeness—cont'd

- 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 $\mathcal{C}$ 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.)

Completeness—cont'd

- 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.

- 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 $\mathbf{P} = \mathbf{NP}$.

Completeness—cont'd

- 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.

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.

Table method—cont'd

- 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).
- 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).

Computation table

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

Computation table—cont'd

Some standardizing assumptions:

- M has one string.
- M halts on any input x after at most $|x|^k - 2$ steps .
- 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 entry $T_{i,j}$ is σ_q except for "yes"("no") for which the entry is "yes"("no"), respectively.
- The cursor starts not from $\triangleright$ but at the first symbol of input.

Computation table—cont'd

- The cursor never visits the leftmost $\triangleright$ (by merging two moves if M would be visiting the leftmost $\triangleright$). Hence, the first symbol of each row is always $\triangleright$ ($\triangleright_q$ is not possible).
- 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.

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

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.

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.

(Proof—cont'd)

- 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 position or adjacent positions at time $i-1$:

  $T_{i,j}$ depends only on $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$

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

(Proof—cont'd)

- Let Γ denote the set of all symbols appearing in the table.

  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 $0 \leq i,j \leq n^k-1$, $1 \leq l \leq m$.
- Each $S_{i,j,l}$ depends only on $3m$ entries

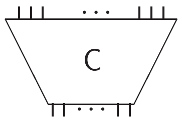
  $S_{i-1,j-1,l'}, S_{i-1,j,l'}, S_{i-1,j+1,l'}$ where $1 \leq l' \leq m$
- So there are Boolean functions $F_1, \ldots, F_m$ with $3m$ inputs each such that for all $i,j > 0$

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

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

$S_{i-1,j-1,1} \ldots S_{i-1,j+1,m}$



$S_{i,j,1} \ldots S_{i,j,m}$

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

(Proof—cont'd)

- 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 (**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).

(Proof—cont'd)

- Value of $R(x)$ is **true** iff $x \in L$

  Suppose value of $R(x)$ is **true**. By induction on i it can be shown that the values of output of $C_{i,j}$ give the binary encoding of the computation table of M after each step i.
  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**.
- $R(x)$ 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.
- The result is a monotone Boolean function.

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

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

(See tutorials.)

Nondeterministic computation as a Boolean circuit

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 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.

(Proof—cont'd)

- We can assume that M has at each step exactly two nondeterministic choices:

  When some configuration has $m > 2$ choices, these can be modified to binary choices with $m-2$ new states.

  If there is only one choice, we consider that two choices coincide and if there is no choice, we add a choice which changes nothing.

- Hence, M has at each step exactly two nondeterministic choices 0 and 1. So a sequence of nondeterministic choice is a bitstring $(c_0, c_2, \ldots, c_{|x|^k-2}) \in \{0,1\}^{|x|^k-1}$.

(Proof—cont'd)

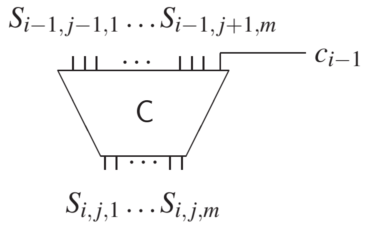
- If we fix the sequence of choices $\mathbf{c} = (c_0, c_2, \ldots, c_{|x|^k-2})$, then the computation is effectively deterministic and we can defined the computation table $T(M, x, \mathbf{c})$ corresponding to the machine M, input x, and choices $\mathbf{c}$.

- 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*.

(Proof—cont'd)

- 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}.



(Proof—cont'd)

- $R(x)$ is constructed as in the reduction from deterministic computation to CIRCUIT VALUE with the choices added.

- $R(x)$ 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 $x \in L$.

SAT is NP-complete

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

Corollary. (Cook's theorem)
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.