# Introduction to Complexity Theory*
# Lecture 2: NP-completeness and Self Reducibility

Notes taken by Nir Piterman and Dana Fisman

December 24, 1998

**Summary:** It will be proven that the relation $R$ of any $\mathcal{NP}-$complete language is Self-reducible. This will be done using the $\mathcal{SAT}$ self reducibility proved previously and the fact that $\mathcal{SAT}$ is $\mathcal{NP}-$hard (under Levin reduction). Prior to that, a simpler proof of the existence of $\mathcal{NP}-$complete languages will be given.

## 1 Reductions

The notions of self reducibility and $\mathcal{NP}-$completeness require a definition of the term reduction. The idea behind reducing problem $\Pi_1$ to problem $\Pi_2$, is that if $\Pi_2$ is known to be easy, so is $\Pi_1$ or vice versa, if $\Pi_1$ is known to be hard so is $\Pi_2$

**Definition 1** (*Cook Reduction*):
*A* Cook reduction *from problem $\Pi_1$ to problem $\Pi_2$ is a polynomial oracle machine that solves problem $\Pi_1$ on input $x$ while getting oracle answers for problem $\Pi_2$.*

For example:
Let $\Pi_1$ and $\Pi_2$ be decision problems of languages $L_1$ and $L_2$ respectively and $\chi_L$ the characteristic function of $L$ defined to be $\chi_L(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$
Then $\Pi_1$ will be Cook reducible to $\Pi_2$ if exists an oracle machine that on input $x$ asks query $q$, gets answer $\chi_{L_2}(q)$ and gives as output $\chi_{L_1}(x)$ (May ask multiple queries).

**Definition 2** (*Karp Reduction*):
*A* Karp reduction (many to one reduction) *of language $L_1$ to language $L_2$ is a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that $x \in L_1$ if and only if $f(x) \in L_2$.*

**Claim 1.1** *A Karp reduction is a special case of a Cook reduction.*

**Proof:** Given a Karp reduction $f(\cdot)$ from $L_1$ to $L_2$ and an input $x$ to be decided whether $x$ belongs to $L_1$, define the following oracle machine:

1. On input $x$ compute the value $f(x)$.
2. Present $f(x)$ to the oracle of $L_2$.

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

3. The oracle's answer is the desired decision.

The machine runs polynomial time since Step 1 is polynomial as promised by Karp reduction and both Steps 2 and 3 require constant time.
Obviously $M$ accepts $x$ if and only if $x$ is in $L_1$. ■

Hence a Karp reduction can be viewed as a Cook reduction.

**Definition 3** (*Levin Reduction*):
*A* Levin reduction *from relation $R_1$ to relation $R_2$ is a triplet of polynomial time computable functions $f, g$ and $h$ such that:*

1. $x \in L(R_1) \Longleftrightarrow f(x) \in L(R_2)$
2. $\forall (x, y) \in R_1 \;,\; (f(x), g(x, y)) \in R_2$
3. $\forall x, z \;\; (f(x), z) \in R_2 \Longrightarrow (x, h(x, z)) \in R_1$

Note: A Levin reduction from $R_1$ to $R_2$ implies a Karp reduction of the decision problem (using condition 1) and a Cook reduction of the search problem (using conditions 1 and 3).

**Claim 1.2** *Karp reduction is transitive.*

**Proof:** Let $f_1 : \Sigma^* \longrightarrow \Sigma^*$ be a Karp reduction from $L_a$ to $L_b$ and $f_2 : \Sigma^* \longrightarrow \Sigma^*$ be a Karp reduction from $L_b$ to $L_c$
The function $f_1 \circ f_2(\cdot)$ is a Karp reduction from $L_a$ to $L_c$:

- $x \in L_a \iff f_1(x) \in L_b \iff f_2(f_1(x)) \in L_c$.
- $f_1$ and $f_2$ are polynomial time computable, so the composition of the functions is again polynomial time computable.

■

**Claim 1.3** *Levin reduction is transitive.*

**Proof:** Let $(f_1, g_1, h_1)$ be a Levin reduction from $R_a$ to $R_b$ and $(f_2, g_2, h_2)$ be a Levin reduction from $R_b$ to $R_c$. Define:

- $f_3(x) \stackrel{\triangle}{=} f_2(f_1(x))$
- $g_3(x, y) \stackrel{\triangle}{=} g_2(f_1(x), g_1(x, y))$
- $h_3(x, y) \stackrel{\triangle}{=} h_1(x, h_2(f_1(x), y))$

We show that the triplet $(f_3, g_3, h_3)$ is a Levin reduction from $R_a$ to $R_c$:

- $x \in L(R_a) \iff f_3(x) \in L(R_c)$
  Since:
  $x \in L(R_a) \iff f_1(x) \in L(R_b) \iff f_2(f_1(x)) \in L(R_c) \iff f_3(x) \in L(R_c)$

- $\forall (x,y) \in R_a\ ,\ (f_3(x), g_3(x,y)) \in R_c$
  Since:
  $(x,y) \in R_a \Longrightarrow (f_1(x), g_1(x,y)) \in R_b \Longrightarrow (f_2(f_1(x)), g_2(f_1(x), g_1(x,y))) \in R_c \Longrightarrow$
  $(f_3(x), g_3(x,y)) \in R_c$

- $\forall x, z\ (f_3(x), z) \in R_c \Longrightarrow (x, h_3(x,z)) \in R_a$
  Since:
  $(f_3(x), z) \in R_c \Longrightarrow (f_2(f_1(x)), z) \in R_c \Longrightarrow (f_1(x), h_2(f_1(x), z)) \in R_b \Longrightarrow$
  $(x, h_1(x, h_2(f_1(x), z))) \in R_a \Longrightarrow (x, h_3(x,z)) \in R_a$

■

**Theorem 4** *If $\Pi_1$ Cook reduces to $\Pi_2$ and $\Pi_2 \in \mathcal{P}$ then $\Pi_1 \in \mathcal{P}$.*

Here class $\mathcal{P}$ denotes not only languages but also any problem that can be solved in polynomial time.
**Proof:** We shall build a deterministic polynomial time Turing machine that recognizes $\Pi_1$:
As $\Pi_1$ Cook reduces to $\Pi_2$, there exists a polynomial oracle machine $M_1$ that recognizes $\Pi_1$ while asking queries to an oracle of $\Pi_2$.
As $\Pi_2 \in \mathcal{P}$, there exists a deterministic polynomial time Turing machine $M_2$ that recognizes $\Pi_2$.
Now build a machine $M$, recognizer for $\Pi_1$ that works as following:

- On input $x$, emulate $M_1$ until it poses a query to the oracle.
- Present the query to the machine $M_2$ and return the answer to $M_1$.
- Proceed until no more queries are presented to the oracle.
- The output of $M_1$ is the required answer.

Since the oracle and $M_2$ give the same answers to the queries, correctness is obvious.
Considering the fact that $M_1$ is polynomial, the number of queries and the length of each query are polynomial in $|x|$. Hence the delay caused by introducing the machine $M_2$ is polynomial in $|x|$.
Therefore the total run time of $M$ is polynomial. ■

## 2 All $\mathcal{NP}$-complete relations are Self-reducible

**Definition 5** ($\mathcal{NP}-$complete *language*):
*A language $L$ is $\mathcal{NP}$-complete if:*

1. $L \in \mathcal{NP}$
2. *For every language $L'$ in $\mathcal{NP}$, $L'$ Karp reduces to $L$.*

These languages are the hardest problems in $\mathcal{NP}$, in the sense that if we knew how to solve an $\mathcal{NP}-$complete problem efficiently we could have efficiently solved any problem in $\mathcal{NP}$. $\mathcal{NP}-$completeness can be defined in a broader sense by Cook reductions. There are not many known $\mathcal{NP}-$complete problems by Cook reductions that are not $\mathcal{NP}-$complete by Karp reductions.

**Definition 6** *1. $R$ is a $\mathcal{NP}$* relation *if $L(R) \in \mathcal{NP}$*

*2. A relation $R$ is $\mathcal{NP}$*-hard **under Levin reduction** *if any $\mathcal{NP}$ relation $R'$ is Levin reducible to $R$.*

**Theorem 7** *For every $\mathcal{NP}$ relation $R$, if $L(R)$ is $\mathcal{NP}-$*complete *then $R$ is Self-reducible.*

**Proof:** To prove the theorem we shall use two facts:

1. $\mathcal{SAT}$ is Self-reducible (was proved last lecture).
2. $R_{\mathcal{SAT}}$ is $\mathcal{NP}-$hard under Levin reduction (will be proven later).

Given an $\mathcal{NP}$ relation $R$ of an $\mathcal{NP}-$complete language, a Levin reduction $(f, g, h)$ from $R$ to $R_{\mathcal{SAT}}$, a Karp reduction $k$ from $\mathcal{SAT}$ to $L(R)$ and $x$, the following algorithm will find $y$ such that $(x, y) \in R$ (provided that $x \in L(R)$).

The idea behind the proof is very similar to the self reducibility of $R_{\mathcal{SAT}}$:

1. Ask $L(R)$'s oracle whether $x \in L(R)$.
2. On answer $'no'$ declare: $x \notin L(R)$ and abort.
3. On answer $'yes'$ use the function $f$, that preserves the property of belonging to the language, to translate the input $x$ for $L(R)$ into a satisfiable $CNF$ formula $\varphi = f(x)$.
4. Compute $(\sigma_1, ..., \sigma_n)$ a satisfying assignment for $\varphi$ as follows:
   (a) Given a partial assignment $\sigma_1, ..., \sigma_i$ such that $\varphi_i(x_{i+1}, ..., x_n) = \varphi(\sigma_1, ..., \sigma_i, x_{i+1}, x_{i+2}, ..., x_n) \in \mathcal{SAT}$, where $x_{i+1}, ..., x_n$ are variables and $\sigma_1, ..., \sigma_i$ are constants.
   (b) Assign $x_{i+1} = 1$ and compute $\varphi_i(1, x_{i+2}, ..., x_n) = \varphi(\sigma_1, ..., \sigma_i, 1, x_{i+2}, ..., x_n)$
   (c) Use the function $k$ to translate the $CNF$ formula $\varphi_i(1, x_{i+2}, ..., x_n)$ into an input to the language $L(R)$. Ask $L(R)$'s oracle wheather $k(\varphi_i(1, x_{i+2}, ..., x_n)) \in L(R)$.
   (d) On answer $'yes'$ assign $\sigma_{i+1} = 1$, otherwise assign $\sigma_{i+1} = 0$.
   (e) Iterate until $i = n - 1$.
5. Use the function $h$ that translates a pair $x$ and a satisfying assignment $\sigma_1, ..., \sigma_n$ to $\varphi = f(x)$ into a witness $y = h(x, (\sigma_1, ..., \sigma_n))$ such that $(x, y) \in R$.

Clearly $(x, y) \in R$. ∎

Note: The above argument uses a Karp reduction of $\mathcal{SAT}$ to $L(R)$ (guaranteed by the NP-completeness of the latter). One may extend the argument to hold also for the case one is only given a Cook reduction of $\mathcal{SAT}$ to $L(R)$. Specifically in stage 4(c) instead of getting the answer to whether $\varphi_i(1, x_{i+2}, ..., x_n)$ is in $\mathcal{SAT}$ by quering on whether $k(\varphi_i)$ is in $L(R)$, we can get the answer by running the oracle machine given in the Cook reduction (which makes queries to $L(R)$).

## 3 $B_{ounded}H_{alting}$ is $\mathcal{NP}-$complete

In order to show that indeed exist problems in $\mathcal{NP}-$complete (i.e. the class $\mathcal{NP}-$complete is not empty) the language $BH$ will be introduced and proved to be $\mathcal{NP}-$complete.

**Definition 8** (Bounded Halting):

1. $BH \triangleq \left\{ (\langle M \rangle, x, 1^t) \,\middle|\, \begin{array}{l} \langle M \rangle \textit{ is the description of a non-deterministic machine} \\ \textit{that accepts input } x \textit{ within } t \textit{ steps.} \end{array} \right\}$
2. $BH \triangleq \left\{ (\langle M \rangle, x, 1^t) \,\middle|\, \begin{array}{l} \langle M \rangle \textit{ is the description of a deterministic machine and exists } y \textit{ whose} \\ \textit{length is polynomial in } |x| \textit{ such that } M \textit{ accepts } (x,y) \textit{ within } t \textit{ steps.} \end{array} \right\}$

The two definitions are equivalent if we consider the $y$ wanted in the second as the sequence of non deterministic choices of the first. The computation is bounded by $t$ hence so is $y$'s length.

**Definition 9** $R_{BH} \triangleq \left\{ ((\langle M \rangle, x, 1^t), y) \,\middle|\, \begin{array}{l} \langle M \rangle \textit{ is the description of a deterministic machine} \\ \textit{that accepts input } (x,y) \textit{ within } t \textit{ steps.} \end{array} \right\}$

Once again the length of the witness $y$ is bounded by $t$, hence it is polynomial in the length of the input $(\langle M \rangle, x, 1^t)$.

Directly from $\mathcal{NP}$'s definition: $BH \in \mathcal{NP}$.

**Claim 3.1** *Any language $L$ in $\mathcal{NP}$, Karp reduces to $BH$*

**Proof:**
Given a language $L$ in $\mathcal{NP}$, implies the following:

- A witness relation $R_L$ exists and has a polynomial bound $b_L(\cdot)$ such that:
  $\forall \ (x,y) \in R_L, \quad |y| \leq b_L(|x|)$
- A recognizer machine $M_L$ for $R_L$ exists and its time is bounded by another polynomial $p_L(\cdot)$.

The reduction maps $x$ to $f(x) \triangleq (\langle M_L \rangle, x, 1^{p_L(|x|+b_L(|x|))})$, which is an instance to $BH$ by Version 2 of Definition 8 above.

Notice that the reduction is indeed polynomial since $\langle M_L \rangle$ is a constant string for the reduction from $L$ to $BH$. All the reduction does is print this constant string, concatenate the input $x$ to it and then concatenate a polynomial number of ones.
We will show now that $x \in L$ if and only if $f(x) \in BH$:

$x \in L \iff$
Exists a witness $y$ whose length is bounded by $b_L(|x|)$ such that $(x,y) \in R_L \iff$
Exists a computation of $M_L$ with $t \triangleq P_L(|x| + b_L(|x|))$ steps accepting $(x,y) \iff$
$(\langle M \rangle, x, 1^t) \in BH$
■

<u>Note:</u> The reduction can be easily transformed into Levin reduction of $R_L$ to $R_{BH}$ with the identity function supplying the two missing functions.
Thus $BH \in \mathcal{NP}-$complete.

**Corollary 10** *There exist* $\mathcal{NP}-$complete *sets.*

## 4 $C_{ircuit}S_{atisfiability}$ is $\mathcal{NP}-$complete

**Definition 11** (Circuit Satisfiability):

1. *A* **Circuit** *is a directed a-cyclic graph $G = (V, E)$ with vertices labeled:*
   *output*, $\bigwedge, \bigvee, \neg, x_1, ..., x_m, 0, 1$
   *With the following restrictions:*

   - *a vertex labeled by $\neg$ has in-degree 1.*
   - *a vertex labeled by $x_i$ has in-degree 0 (i.e. is a source).*
   - *a vertex labeled by 0 (or 1) has in-degree 0.*
   - *there is a single sink (vertex of out-degree 0), it has in-degree 1 and is labeled 'output'.*
   - *in-degree of vertices labeled $\bigwedge, \bigvee$ can be restricted to 2.*

   *Given an assignment $\sigma \in \{0,1\}^m$ to the variables $x_1, ..., x_m$, $C(\sigma)$ will denote the value of the circuit's output. The value is defined in the natural manner, by setting the value of each vertex according to the boolean operation it is labeled with. For example, if a vertex is labelled $\bigwedge$ and the vertices with a directed edge to it have values $a$ and $b$, then the vertex has valu $a \bigwedge b$.*

2. **Circuit Satisfiability**
   $CS \stackrel{\triangle}{=} \{C : C$ *is a circuit and exists* $\sigma$*, an input to circuit* $C$ *such that* $C(\sigma) = 1\}$

3. $R_{CS} \stackrel{\triangle}{=} \{(C, \sigma) : C(\sigma) = 1\}$

The relation defined above is indeed an $\mathcal{NP}$ relation since:

1. $\sigma$ contains assignment for all variables $x_1, x_2, ..., x_m$ appearing in $C$ and hence its length is polynomial in $|C|$.

2. Given a couple $(C, \sigma)$ evaluating one gate takes $O(1)$ (since in-degree is restricted to 2) and in view that the number of gates is at most $|C|$, total evaluation time is polynomial in $|C|$.

Hence $CS \in \mathcal{NP}$.

**Claim 4.1** $C_{ircuit}S_{atisfiability}$ *is* $\mathcal{NP}-$complete

**Proof:** As mentioned previously $CS \in \mathcal{NP}$.
We will show a Karp reduction from $BH$ to $CS$, and since Karp reductions are transitive and $BH$ is $\mathcal{NP}-$complete, the proof will be completed. In this reduction we shall use the second definition of $BH$ as given in Definition 8.
Thus we are given a triplet $(\langle M \rangle, x, 1^t)$. This triplet is in $BH$ if exists a $y$ such that the deterministic machine $M$ accepts $(x, y)$ within $t$ steps. The reduction maps such a triplet into an instance of $CS$.
The idea is building a circuit that will simulate the run of $M$ on $(x, y)$, for the given $x$ and a generic $y$ (which will be given as an input to the circuit). If $M$ does not accept $(x, y)$ within the first $t$ steps of the run, we are ensured that $(\langle M \rangle, x, 1^t)$ is not in $BH$. Hence it suffices to simulate only the first $t$ steps of the run.
Each one of these first $t$ configurations is completely described by the letters written in the first $t$ tape cells, the head's location and the machine's state.

Hence the whole computation can be encoded in a matrix of size $t \times t$. The entry $(i, j)$ of the matrix will consist of the contents of cell $j$ at time $i$, an indicator whether the head is on this cell at time $i$ and in case the head is indeed there the state of the machine is also encoded. So every matrix entry will hold the following information:

- $a_{i,j}$ the letter written in the cell
- $h_{i,j}$ an indicator to head's presence in the cell
- $q_{i,j}$ the machine's state in case the head indicator is 1 (0 otherwise)

The contents of matrix entry $(i, j)$ is determined only by the three matrix entries $(i-1, j-1)$, $(i-1, j)$ and $(i-1, j+1)$. If the head indicator of these three entries is off, entry $(i, j)$ will be equal to entry $(i-1, j)$.
The following constructs a circuit that implements the idea of the matrix and this way emulates the run of machine $M$ on input $x$. The circuit consists of $t$ levels of $t$ triplets $(a_{i,j}, h_{i,j}, q_{i,j})$ where $0 \leq i \leq t,\ 1 \leq j \leq t$. Level $i$ of gates will encode the configuration of the machine at time $i$. The wiring will make sure that if level $i$ represents the correct configuration, so will level $i + 1$.
The $(i, j)$-th triplet, $(a_{i,j}, h_{i,j}, q_{i,j})$, in the circuit is a function of the three triplets $(i-1, j-1)$, $(i-1, j)$ and $(i-1, j+1)$.
Every triplet consists of $O(log\ n)$ bits, where $n \triangleq |(\langle M\rangle, x, 1^t)|$:

- Let $G$ denote the size of $M$'s alphabet. Representing one letter requires $log\ G$ many bits: $log\ G = O(log\ n)$ bits.
- The head indicator requires one bit.
- Let $K$ denote the number of states of $M$. Representing a state requires $log\ K$ many bits: $log\ K = O(log\ n)$ bits.

Note that the machine's description is given as input. Hence the number of states and the size of the alphabet are smaller than input's size and can be represented in binary by $O(log\ n)$ many bits (Albeit doing the reduction directly from any $\mathcal{NP}$ language to $CS$, the machine $M_L$ that accepts the language $L$ wouldn't have been given as a parameter but rather as a constant, hence a state or an alphabet letter would have required a constant number of bits).
Every bit in the description of a triplet is a boolean function of the bits in the description of three other triplets, hence it is a boolean function of $O(log\ n)$ bits.

**Claim 4.2** *Any boolean function on $m$ variables can be computed by a circuit of size $m2^m$*

**Proof:** Every boolean function on $m$ variables can be represented by a $(m + 1) \times 2^m$ matrix. The first $m$ columns will denote a certain input and the last column will denote the value of the function. The $2^m$ rows are required to describe all different inputs.
Now the circuit that will calculate the function is:
For line $l$ in the matrix in which the function value is 1 ($f(l) = 1$), build the following circuit:
$$C_l = \Big(\bigwedge_{\text{input } y_i=1} y_i\Big) \wedge \Big(\bigwedge_{\text{input } y_i=0} \neg y_i\Big)$$
Now take the OR of all lines (value 1):
$$C = \bigvee_{f(l)=1} C_l$$
The circuit of each line is of size $m$ and since there are at most $2^m$ lines of value 1, the size of the whole circuit is at most $m2^m$. ■

So far the circuit emulates a generic computation of $M$. Yet the computation we care about refers to one specific input. Similarly the initial state should be $q_0$ and the head should be located at time 0 in the first location. This will be done by setting all triplets $(0, j)$ as following:
Let $x = x_1 x_2 x_3 ... x_m$ and $n \triangleq |(\langle M \rangle, x, 1^t)|$ the length of the input.

- $a_{0,j} = \begin{cases} x_j & 1 \le j \le m \quad \text{constants set by input } x \\ y_{j-m} & m < j \le t \quad \text{these are the inputs to the circuit} \end{cases}$
- $h_{0,j} = \begin{cases} 1 & j = 1 \\ 0 & j \ne 1 \end{cases}$
- $q_{0,j} = \begin{cases} q_0 & j = 1 \quad \text{where } q_0 \text{ is the initial state of } M \\ 0 & j \ne 1 \end{cases}$

The $y$ elements are the variables of the circuit. The circuit belongs to $CS$ if and only if there exists an assignment $\sigma$ for $y$ such that $C(\sigma) = 1$. Note that $y$, the input to the circuit plays the same role as the short witness $y$ to the fact that $(\langle M \rangle, x, 1^t)$ is a member of $BH$. Note that (by padding $y$ with zeros), we may assume without loss of generality that $|y| = t - |x|$.

So far (on input $y$) the circuit emulates a running of $M$ on input $(x, y)$, it is left to ensure that $M$ accepts $(x, y)$. The output of the circuit will be determined by checking whether at any time the machine entered the $'accept'$ state. This can be done by checking whether in any of the $t \times t$ triplets in the circuit the state is $'accept'$.

Since every triplet $(i, j)$ consists of $O(log\ n)$ bits we have $O(log\ n)$ functions associated with each triplet. Every function can be computed by a circuit of size $O(n\ log\ n)$, so the circuit attached to triplet $(i, j)$ is of size $O(n\ log^2\ n)$.
There are $t \times t$ such triplets so the size of the circuit is $O(n^3\ log^2\ n)$.
Checking for a triplet $(i, j)$ whether $q_{i,j}$ is $'accept'$ requires a circuit of size $O(log\ n)$. This check is implemented for $t \times t$ triplets, hence the overall size of the output check is $O(n^2\ log\ n)$ gates.
The overall size of the circuit will be $O(n^3\ log^2\ n)$.

Since the input level of the circuit was set to represent the right configuration of machine $M$ when operated on input $(x, y)$ at time 0, and the circuit correctly emulates with its $i^{th}$ level the configuration of the machine at time $i$, the value of the circuit on input $y$ indicates whether or not $M$ accepts $(x, y)$ within $t$ steps. Thus, the circuit is satisfiable if and only if there exists a $y$ so that $M$ accepts $(x, y)$ within $t$ steps, i.e. $(\langle M \rangle, x, 1^t)$ is in $BH$.

For a detailed description of the circuit and full proof of correction see Appendix.

The above description can be viewed as instructions for constructing the circuit. Assuming that building one gate takes constant time, constructing the circuit following these instructions will be linear to the size of the circuit. Hence, construction time is polynomial to the size of the input $(\langle M \rangle, x, 1^t)$.
■

Once again the missing functions for Levin reduction of $R_{BH}$ to $R_{CS}$ are the identity functions.

## 5 $R_{\mathcal{SAT}}$ *is* $\mathcal{NP}-$complete

**Claim 5.1** *$R_{\mathcal{SAT}}$ is $\mathcal{NP}-$*hard *under Levin reduction.*

**Proof:** Since Levin reduction is transitive it suffices to show a reduction from $R_{CS}$ to $R_{\mathcal{SAT}}$:
The reduction will map a circuit $C$ to an $CNF$ expression $\varphi_{\mathrm{C}}$ and an input $y$ for the circuit to an assignment $y'$ to the expression and vice versa.
We begin by describing how to construct the expression $\varphi_{\mathrm{C}}$ from $C$.
Given a circuit $C$ we allocate a variable to every vertex of the graph. Now for every one of the vertices $v$ build the $CNF$ expression $\varphi_v$ that will force the variables to comply to the gate's function:

1. For a $\neg$ vertex $v$ with edge entering from vertex $u$:
   - Write $\varphi_v(v,u) = ((v \bigvee u) \bigwedge (\neg u \bigvee \neg v))$
   - It follows that $\varphi_v(v,u) = 1$ if and only if $v = \neg u$
2. For a $\bigvee$ vertex $v$ with edges entering from vertices $u, w$:
   - Write $\varphi_v(v,u,w) = ((u \bigvee w \bigvee \neg v) \bigwedge (u \bigvee \neg w \bigvee v) \bigwedge (\neg u \bigvee w \bigvee v) \bigwedge (\neg u \bigvee \neg w \bigvee v))$
   - It follows that $\varphi_v(v,u,w) = 1$ if and only if $v = u \bigvee w$
3. For a $\bigwedge$ vertex $v$ with edges entering from vertices $u, w$:
   - Similarly write $\varphi_v(v,u,w) = ((u \bigvee w \bigvee \neg v) \bigwedge (u \bigvee \neg w \bigvee \neg v) \bigwedge (\neg u \bigvee w \bigvee \neg v) \bigwedge (\neg u \bigvee \neg w \bigvee v))$
   - It follows that $\varphi_v(v,u,w) = 1$ if and only if $v = u \bigwedge w$
4. For the vertex marked *output* with edge entering from vertex $u$:
   Write $\varphi_{output}(u) = u$

We are ready now to define $\varphi_{\mathrm{C}} = \bigwedge\limits_{v \in V} \varphi_v$, where V is the set of all vertices of in-degree at least one (i.e. the constant inputs and variable inputs to the circuit are not included).
The length of $\varphi_{\mathrm{C}}$ is linear to the size of the circuit. Once again the instructions give a way to build the expression in linear time to the circuit's size.

We next show that $C \in CS$ if and only if $\varphi_{\mathrm{C}} \in \mathcal{SAT}$. Actually, to show that the reduction is a Levin-reduction, we will show how to efficiently transform witnesses for one problem into witnesses for the other. That is, we describe how to construct the assignment $y'$ to $\varphi_{\mathrm{C}}$ from an input $y$ to the circuit $C$ (and vice versa):
Let $C$ be a circuit with $m$ input vertices labeled $x_1, ..., x_m$ and $d$ vertices labeled $\bigvee, \bigwedge$ and $\neg$ namely, $v_1, ..., v_d$. An assignment $y = y_1, ..., y_m$ to the circuit's input vertices will propagate into the circuit and set the value of all the vertices. Considering that the expression $\varphi_{\mathrm{C}}$ has a variable for every vertex of the circuit $C$, the assignment $y'$ to the expression should consist of a value for every one of the circuit vertices. We will build $y' = y'_{x_1}, ..., y'_{x_m}, y'_{v_1}, y'_{v_2}, ..., y'_{v_d}$ as following:

- The variables of the expression that correspond to input vertices of the circuit will have the same assignment: $y'_{x_h} = y_h,\ 1 \leq h \leq m$.
- The assignment $y'_{v_l}$ to every other expression variable $v_l$ will have the value set to the corresponding vertex in the circuit, $1 \leq l \leq d$.

Similarly given an assignment to the expression, an assignment to the circuit can be built. This will be done by using only the values assigned to the variables corresponding to the input vertices of the circuit. It is easy to see that:
$C \in CS \iff$ exists $y$ such that $C(y) = 1 \iff \varphi_{\mathrm{C}}(y') = 1 \iff \varphi_{\mathrm{C}} \in \mathcal{SAT}$ ■

**Corollary 12** $\mathcal{SAT}$ *is* $\mathcal{NP}-$complete

## 6 Appendix: Details for the reduction of $BH$ to $CS$

We present now the details of the reduction from $BH$ to $CS$. The circuit that will emulate the run of machine $M$ on input $x$ can be constructed in the following way:
Let $(\langle M\rangle, x, t)$ be the input to be determined whether is in $BH$, where $x = x_1 x_2 ... x_m$ and $n \triangleq |(\langle M\rangle, x, t)|$ the length of the input.
We will use the fact that every gate of in-degree $r$ can be replaced by $r$ gates of in-degree 2. This can be done by building a balanced binary tree of depth $log\ r$. In the construction $'and', 'or'$ gates of varying in-degree will be used. When analyzing complexity, every such gate will be weighed as its in-degree.
The number of states of machine $M$ is at most $n$, hence $log\ n$ bits can represent a state. Similarly the size of alphabet of machine $M$ is at most $n$, and therfore $log\ n$ bits can represent a letter.

1. Input Level
   $y$ is the witness to be entered at a later time (assume $y$ is padded by zeros to complete length $t$ as explained earlier).

   - $a_{0,j} = \begin{cases} x_j & 1 \le j \le m \quad \text{constants set by input } x \\ y_{j-m} & m < j \le t \quad \text{these are the inputs to the circuit} \end{cases}$
   - $h_{0,j} = \begin{cases} 1 & j = 1 \\ 0 & j \neq 1 \end{cases}$
   - $q_{0,j} = \begin{cases} q_0 & j = 1 \quad \text{where } q_0 \text{ is the initial state of } M \\ 0 & j \neq 1 \end{cases}$

   As said before this represents the configuration at time 0 of the run of $M$ on $(x, y)$.
   This stage sets $O(n\ log\ n)$ wires.

2. For $0 < i < t$ , $h_{i+1,j}$ will be wired as shown in figure 1:

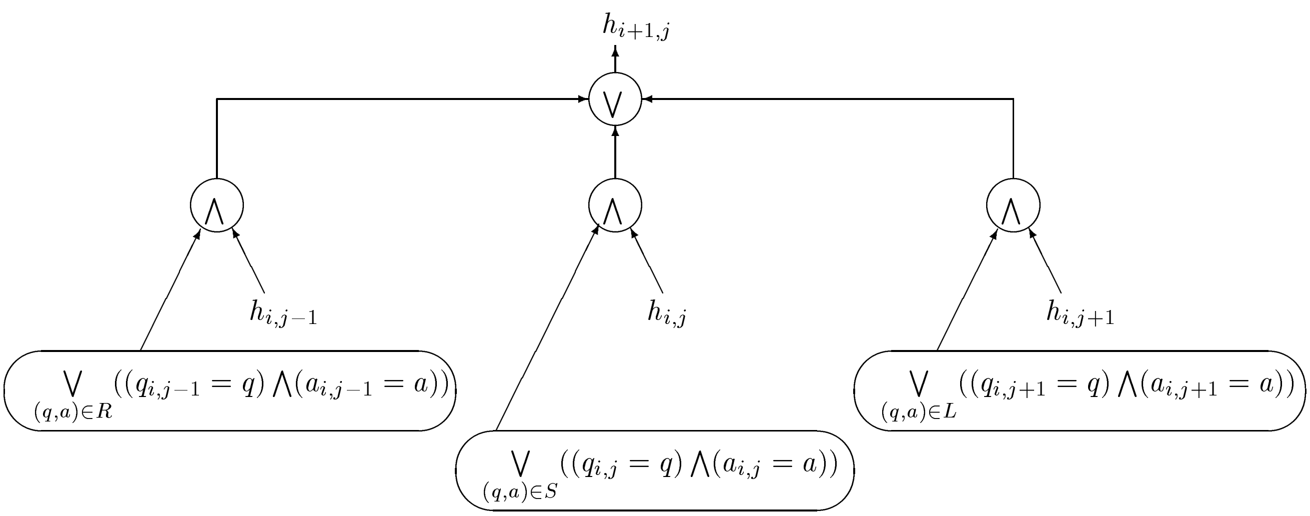


figure 1

The definition of groups $R, S, L$ is:
$R \triangleq \{(q,a) : q \in K \bigwedge a \in \{0,1\} \bigwedge \delta(q,a) = (*,*,R)\}$
$S \triangleq \{(q,a) : q \in K \bigwedge a \in \{0,1\} \bigwedge \delta(q,a) = (*,*,S)\}$
$L \triangleq \{(q,a) : q \in K \bigwedge a \in \{0,1\} \bigwedge \delta(q,a) = (*,*,L)\}$
The equations are easily wired using an $'and'$ gate for every equation.
The size of this component:
The last item on every entry in the relation $\delta$ is either $R, L$ or $S$. For every one of these entries there is one comparison above. Since $\delta$ is bounded by $n$ there are at most $n$ such comparisons. A comparison of the state requires $O(log\ n)$ gates. Similarly a comparison of the letter requires $O(log\ n)$ gates. Hence the total number of gates in figure 1 is $O(n\ log\ n)$

3. For $0 < i < t$ , $q_{i+1,j}$ will be wired as shown in figure 2:

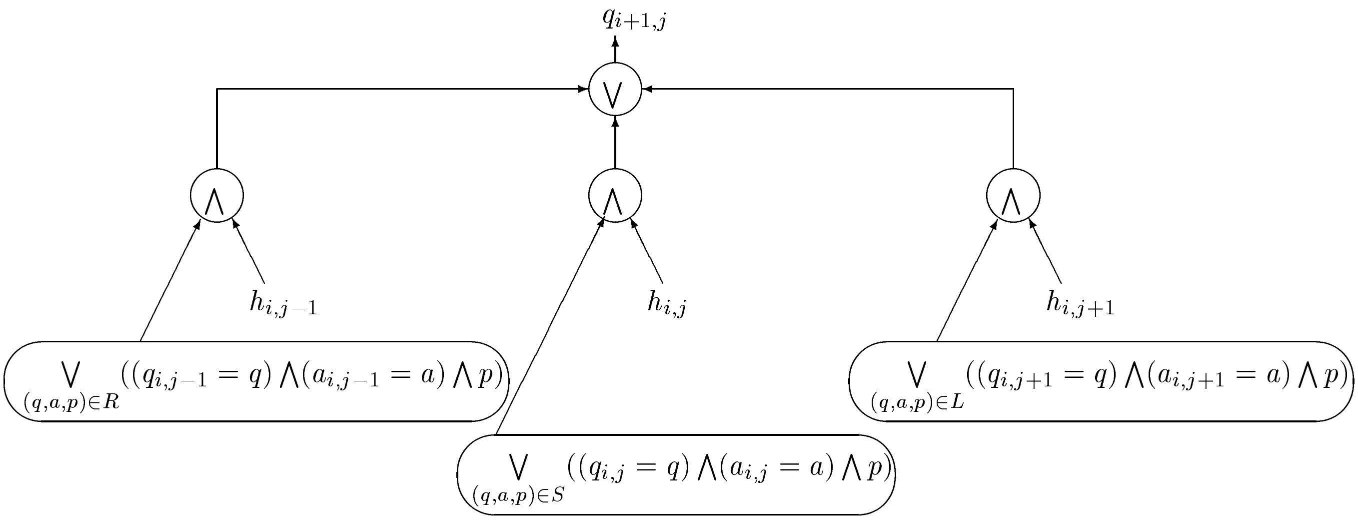


figure 2

The definition of groups $R, S, L$ in:
$R \triangleq \{(q,a,p) : q,p \in K \bigwedge a \in \{0,1\} \bigwedge \delta(q,a) = (p,*,R)\}$
$S \triangleq \{(q,a,p) : q,p \in K \bigwedge a \in \{0,1\} \bigwedge \delta(q,a) = (p,*,S)\}$
$L \triangleq \{(q,a,p) : q,p \in K \bigwedge a \in \{0,1\} \bigwedge \delta(q,a) = (p,*,L)\}$
Once again every comparison requires $O(log\ n)$ gates. Every state is represented by $log(n)$ bits so the figure has to be multiplied for every bit.
Overall complexity of the component in figure 2 is $O(n\ log^2\ n)$.

4. For $0 < i < t$ , $a_{i+1,j}$ will be wired as shown in figure 3:

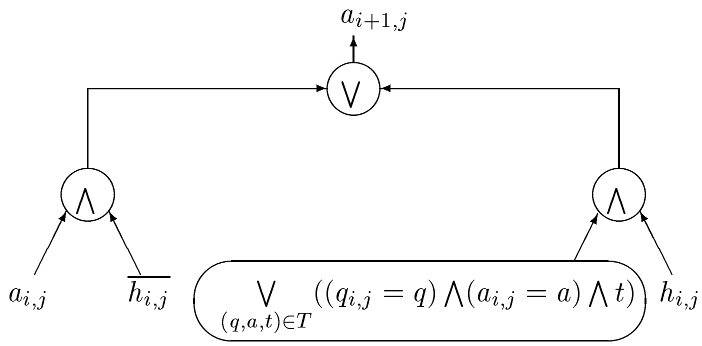


figure 3

The definition of $T$ is:
$T \triangleq \{(q,a,t) : q \in K \bigwedge a, t \in \{0,1\} \bigwedge \delta(q,a) = (*, t, *)\}$
Once again all entries of the relation $\delta$ have to be checked, hence there are $O(n)$ comparisons of size $O(log\ n)$.
Since the letter is represented by $O(log\ n)$ bits, the overall complexity of the component in figure 3 is $O(n\ log^2\ n)$.

5. Finally the output gate of the circuit will be a check whether at any level of the circuit the state was *accept*. This will be done by comparing $q_{i,j}$, $1 \leq j \leq t$, $0 \leq i \leq t$ to $'accept'$. There are $t \times t$ such comparisons, each of them takes $O(log\ n)$ gates. Taking an OR on all these comparisons costs $O(n^2\ log\ n)$ gates.

For every cell in the $t \times t$ matrix we used $O(n^3)$ gates. The whole circuit can be built with $O(n^5)$ gates. With this description, building the circuit is linear to circuit's size. Hence, this can be done in polynomial time.

Correctness: We will show now that $(\langle M\rangle, x, 1^t) \in BH$ if and only if $C_{(\langle M\rangle,x,t)} \in CS$

**Claim 6.1** *Gates at level* $i$ *of the circuit represent the exact configuration of* $M$ *at time* $i$ *on input* $(x, y)$.

**Proof:** By induction on time i.

- $i = 0$, stage 1 of the construction ensures correctness.

- Assume $C$'s gates on level $i$ correctly represent $M$'s configuration at time $i$ and prove for $i + 1$:
  Set $j$ as the position of the head at time $i$ ($h_{i,j} = 1$).

  - The letter contents of all cells $(i + 1, k), k \neq j$ does not change. Same happens in the circuit since $(a_{i,k} \bigwedge(\neg h_{i,k})) = a_{i,k}$.
  - Likewise the head can not reach cells $(i + 1, k)$ where $k < (j - 1)$ or $k > (j + 1)$. Respectively $h_{i,k} = 0$ since $h_{i,k-1} = h_{i,k} = h_{i,k+1} = 0$.

- The same argument shows that state bits for all gates of similar $k's$ will be reset to zero.

Let $\delta(q_{i,j}, a_{i,j}) = (q, a, m)$
We shall look into what happens when machine's head stays in place, i.e. $m = S$. The other two posibilities for movement of the head are similar.

- Cell $(i, j)$ on the tape will change into $a$. Since $h_{i,j} = 1$ and correspondingly $(q_{i,j} = q_{i,j} \bigwedge a_{i,j} = a_{i,j} \bigwedge a)$ will return $a$
- The head stays in place and respectively:
  1. $h_{i+1,j-1} = 0$ since $h_{i,j} = 1$ but $\delta(q_{i,j}, a_{i,j})$ is not $(*, *, L)$.
  2. $h_{i+1,j} = 1$ since $h_{i,j} = 1$ and one $\delta(q_{i,j}, a_{i,j}) = (*, *, S)$ returns 1.
  3. $h_{i+1,j+1} = 0$ since $h_{i,j} = 1$ but $\delta(q_{i,j}, a_{i,j})$ is not $(*, *, R)$.
- The machine's next state is $q$, and respectively:
  1. Similarly $q_{i+1,j-1}$ and $q_{i+1,j+1}$ will be reset to zero.
  2. $q_{i+1,j}$ will change into $q$ since $h_{i,j} = 1$ and $(q_{i,j} = q_{i,j} \bigwedge a_{i,j} = a_{i,j} \bigwedge q)$ will return $q$.

So at any time $0 \leq i \leq t$, gate level $i$ correctly represents $M$'s configuration at time $i$.
■