Theorem 1. *CircuitSAT is $\mathcal{NP}$-complete.*

Proof. To see that CircuitSAT is in $\mathcal{NP}$, consider a verifier that gets as input a circuit C of description length n and expects a witness w that is an assignment to the variables of C; the verifier accepts if and only if $C(w) = 1$. (Convince yourself that this meets the definition of $\mathcal{NP}$.) We thus focus on showing $\mathcal{NP}$-hardness.

Let $L \in \mathcal{NP}$, and let V be a verifier for L running in time $O(T(n)) = \text{poly}(n)$. Without loss of generality, we assume that witnesses causing V to accept its input must always be of length exactly $T(n)$.[1]

Given input $x \in \{0,1\}^n$, we construct in time $\text{poly}(T(n))$ a circuit C_x such that

$$x \in L \iff \langle C_x \rangle \in CircuitSAT \ .$$

The circuit C_x has an input of length $T = T(n)$, which we denote by $w \in \{0,1\}^T$. The circuit has $O(T)$ layers, and each layer has $O(T)$ nodes.

High-level idea:

- The circuit C_x will get input w, and output $C_x(w) = V(x,w)$.
- There is an initial layer, representing the initial configuration in the computation of $V(x,w)$.
- There are $O(T)$ configuration layers. The i^{th} configuration layer represents the configuration of $V(x,w)$ after i computation steps.
- Between each pair of configuration layers, there are $O(1)$ auxiliary layers, which compute the transition between two configurations.
- The last configuration layer will represent the final configuration in the computation of $V(x,w)$, hence the output.

In more detail, denote the running time of V by $\bar{T} = O(T(n))$, the initial layer as L_0, and the configuration layers as $L_1, ..., L_{\bar{T}}$. For each $i \in \{0, ..., \bar{T}\}$, layer L_i will be structured as follows.

- When C_x gets input w, the node values in layer L_i are supposed to represent the configuration of $V(x,w)$ after i steps, denoted C_i.
- Layer L_i is divided into $\bar{T}$ blocks of nodes, where each block B_j has $O(1)$ nodes and corresponds to a cell $j \in [\bar{T}]$ in V's worktape. In each B_j, the node values represent the symbol that appears in cell j (i.e., in configuration C_i), and in addition, if the head is in cell j, these nodes represent the state of $V(x,w)$ in C_i.

  Specifically, the nodes in B_j are split into three sub-blocks:

[1] Showing that this doesn't lose generality is left as an exercise. (Hint: Starting from V that doesn't satisfy this, you can define a new V' that decides the same language but satisfies this condition, with respect to larger polynomial runtime $T'(n) \geq T(n)$.)

  - **Alphabet symbol:** The values of nodes in this sub-block as represent a symbol from the alphabet of V. (The all-zero sub-block represents β.)
  - **Head:** A single node, whose bit-value represents the notion of "the head is in this cell during configuration C_i".
  - **State:** The node values in this sub-block represent the state of V, in case the head is in this cell during configuration C_i (otherwise they are all zero).

- Overall, we interpret the values of $O(\bar{T}) = O(T)$ nodes in a configuration layer L_i as representing the corresponding configuration C_i: the contents of the worktape, an indication of where the head is, and the machine's state.

We construct C_x so that it meets the following invariant:

> When C_x gets input w, for each $i \in \{0, ..., \bar{T}\}$, the values of nodes in layer L_i of $C_x(w)$ correctly represent the configuration of $V(x, w)$ after i steps.

Constructing the initial layer. Given x, we construct layer L_0 of C_x as follows. The layer represents the configuration in which the worktape has "x, w" written on it, the head is in the first cell, and the state is s_{start}. Note that the values of x are hard-wired as nodes with constant values $x_1, ..., x_n$, and the values of w are actually input nodes. Given x, computing a description of layer L_0 of C_x can be done in time $\text{poly}(T)$.

Constructing auxiliary layers that compute L_i from L_{i-1}. Assume that we constructed layer L_{i-1} of C_x such that the invariant is true for L_{i-1}. We want to show that we can efficiently construct $O(1)$ auxiliary layers that compute L_i from L_{i-1} such that the invariant is maintained for L_i.

The key observation is that each node in L_i only depends on constantly many nodes in L_{i-1}. This is because the computation of V (indeed, of any Turing machine) is local, in the following sense: at timestep i, the contents of cell j (including whether or not the head is there) only depends on the contents of cells $j-1, j, j+1$ in timestep $i-1$ (i.e., the content only changes if the head was nearby and moved). Specifically:

Observation 1.1. *Each block in L_i depends on at most three blocks in L_{i-1}, and this dependency is determined by the transition function of V.*

Proof. Let $j \in [T]$ be an index of a cell in the worktape of $V(x, w)$. Let C_{i-1}, C_i be the configurations in the computation of $V(x, w)$ after $i-1$ and i steps, respectively.

- **Alphabet symbol:** If the head doesn't reside in cell j in C_{i-1}, then the symbol appearing in cell j in C_{i-1} and C_i is identical. Otherwise, the transition function of V determines the symbol appearing in cell j, according to the content of cell j and to the state of $V(x, w)$ in C_{i-1}.

- **Head:** The head resides in cell j in C_i if it resided in cell $j-1$ in C_{i-1} and moved right, or in cell $j+1$ and moved left. This can be determined by the transition function of V according to blocks $j-1$ and $j+1$ in L_{i-1}.[2]
- **State:** Similarly, to determine values of state nodes in block j, we need to know if the head resides in cell j in C_i (otherwise state nodes are all-zero), and in case it does, the values of state nodes can be determined by the transition function of V according to blocks $j-1$ and $j+1$ in L_{i-1}. □

The auxiliary layers we construct consist of sub-circuits that, for each node in each block of L_i, compute the value of the node according to the value of $v = O(1)$ nodes in three blocks in L_{i-1}. We construct the auxiliary layers relying on the following fact:

Fact 1.2. *Any function on v variables can be computed by a Boolean circuit of size $O(v \cdot 2^v)$ that can be constructed in time* $\mathrm{poly}(2^v)$.

Proof. Fixing any f, the Boolean circuit $C_f(z_1, ..., z_v)$ implements a lookup table for all possible values for $z_1, ..., z_v$ variables that cause f to output 1. Specifically, let $A = \{a_1, ..., a_v : f(a_1, ..., a_v) = 1\}$, the circuit $C_f(z_1, ..., z_v)$ takes an OR over all $a_1, ..., a_v \in A$ of an AND_s that checks whether $z_1, ..., z_v = a_1, ..., a_v$. We can construct in time $\mathrm{poly}(2^v)$ a circuit C_f of size $O(v \cdot 2^v)$ implementing this.[3] □

We use Fact 1.2 with $v = O(1)$ to compute the value of each node in L_i by a Boolean circuit of size $O(v \cdot 2^v) = O(1)$ taking as input $v = O(1)$ nodes in L_{i-1}. The number of auxiliary layers we need to implement this circuitry (between L_{i-1} and L_i) is $O(v \cdot 2^v) = O(1)$. The total number of nodes in each auxiliary layer is $O(T)$, and we can construct the auxiliary layers in time $\mathrm{poly}(T)$. Note that the output gates of the sub-circuits in the auxiliary layers constitute layer L_i.

Wrapping it up. Overall, we constructed $O(T)$ layers each with $O(T)$ nodes in time $\mathrm{poly}(T)$, and by the invariant, for every w, the node values of L_T in $C_x(w)$ represent the worktape of $V(x, w)$ at the end of its execution. In particular, the left-most block in L_T contains the output of $V(x, w)$, so we can mark the relevant gate in this block as the output gate of C_x.

Then, for every $w \in \{0,1\}^T$ we have that $C_x(w) = V(x, w)$. And since V is a verifier for L, for every x we have:

- $x \in L \Rightarrow \exists w \in \{0,1\}^T : C_x(w) = V(x, w) = 1$.
- $x \notin L \Rightarrow \forall w \in \{0,1\}^T,\ C_x(w) = V(x, w) = 0$.

In particular, $x \in L$ if and only if $C_x \in \mathit{CircuitSAT}$. ■

[2] That is, it suffices to know if the head resided in location $j-1$ or $j+1$ in C_{i-1}, and in either case, what was the corresponding symbol (underneath the head) and the state of $V(x, w)$; all of these are written in blocks $j-1$ and $j+1$ in L_{i-1}. (When $j = 1$, there is no block $j-1$.)

[3] That is, first consider C_f that is an OR of at most 2^v ANDs. The only issue is that the nodes in this C_f have fan-in larger than two. However, we can replace each node g with large fan-in by a small tree of nodes (of the same type as g) with fan-in two such that the tree computes the same function as g. (In fact, there is even a smaller circuit with $O(2^v/v)$ nodes, but this is not part of the course material.)