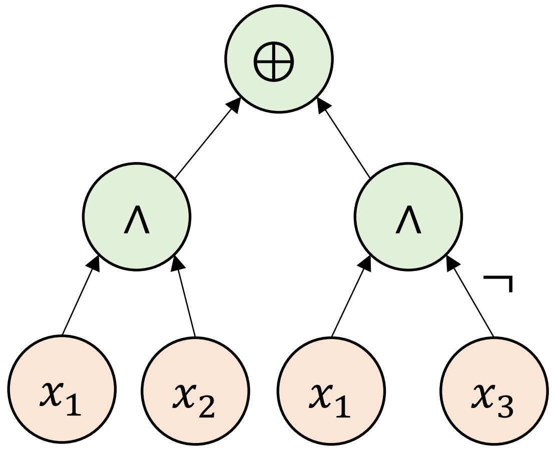
CMSC 28100

Introduction to Complexity Theory

Autumn 2025
Instructor: William Hoza

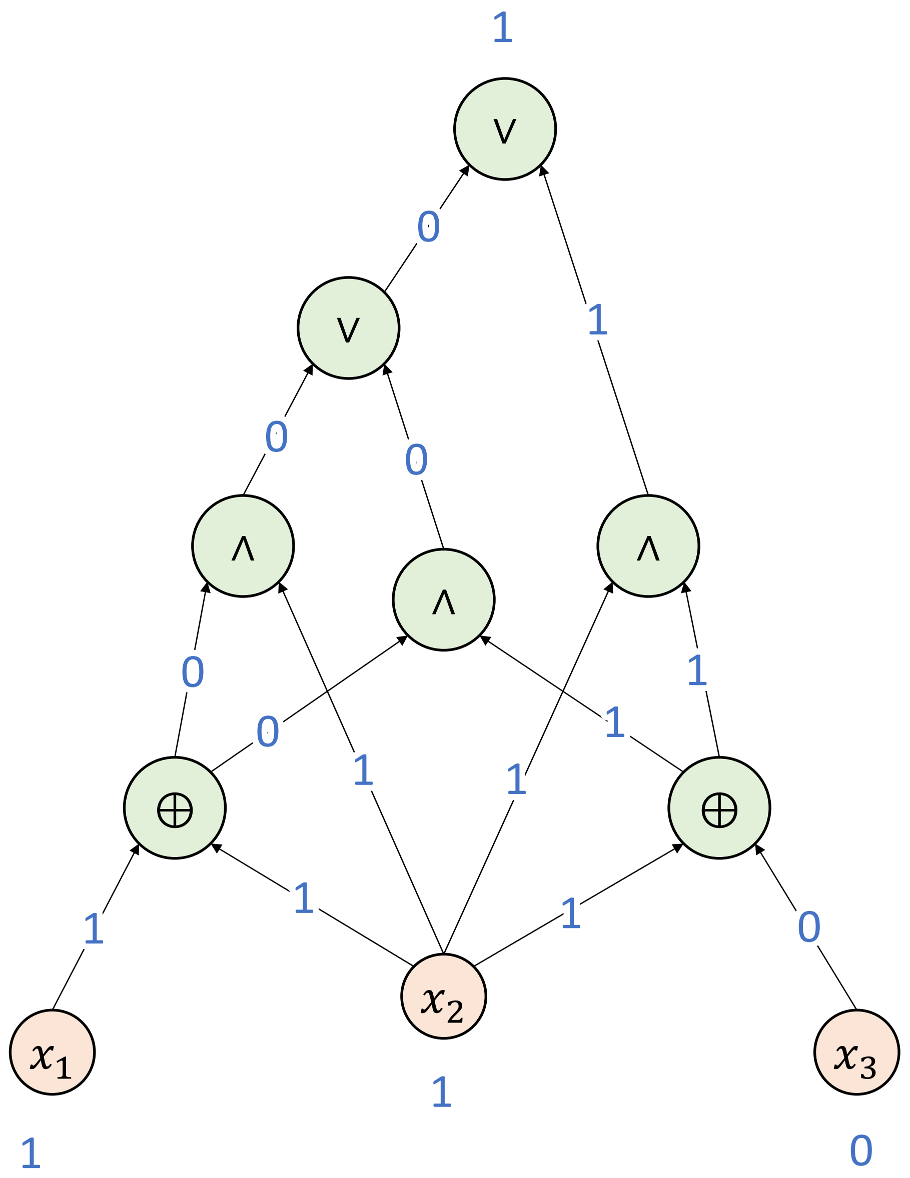
Boolean formulas

- **Definition:** An n-variate Boolean formula is a rooted binary tree
  - Each internal node is labeled with a binary logical operation
  - Each leaf is labeled with 0, 1, or a variable among $x_1, \dots, x_n$
- It computes $f: \{0,1\}^n \to \{0,1\}$
- E.g., $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \oplus (x_1 \wedge \bar{x}_3)$



Boolean circuits

- A Boolean circuit is like a Boolean formula, except that we permit vertices to have multiple outgoing wires



Boolean circuits: Rigorous definition

- **Definition:** An n-input m-output circuit is a directed acyclic graph
  - We refer to the edges as “wires”
  - Two types of nodes:
    - Each “gate” has two incoming edges and is labeled with a binary logical operation
    - Otherwise, a node has zero incoming edges and is labeled with 0, 1, or a variable among $x_1, \ldots, x_n$
  - m of the nodes are additionally labeled as “output 1”, “output 2”, …, “output m”

Boolean circuits: Rigorous definition

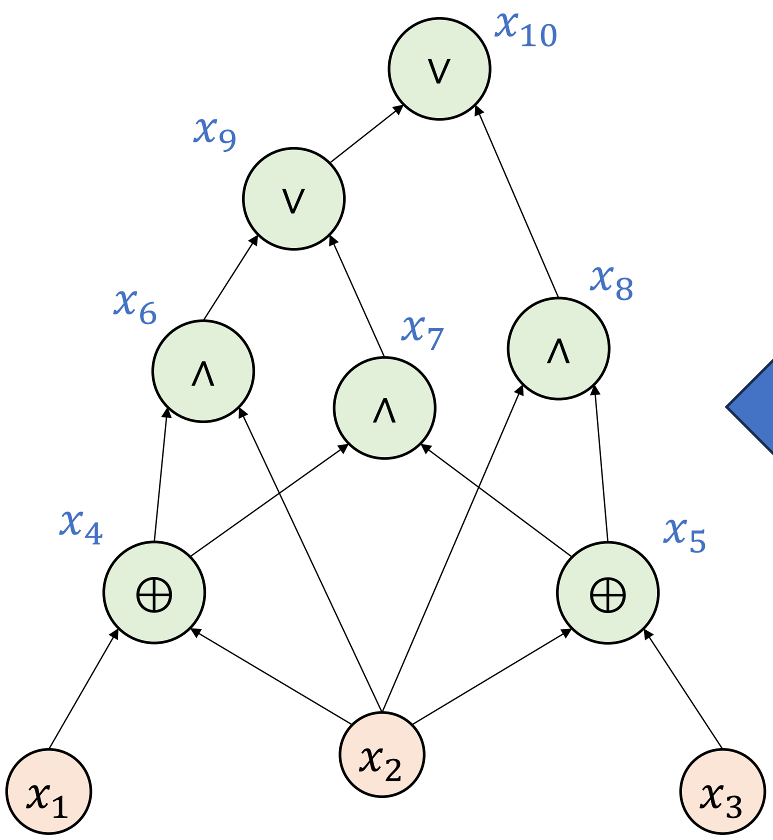
- Each node g computes a function $g: \{0,1\}^n \to \{0,1\}$ defined inductively:
  - If g is labeled x_i, then $g(x) =$ the i-th bit of x
  - If g is labeled 0, then $g(x) \equiv 0$
  - If g is labeled 1, then $g(x) \equiv 1$
  - If g is labeled op and its incoming wires come from f and h, then $g(x) = f(x)\ \mathrm{op}\ h(x)$

Boolean circuits

- Let the output nodes be $g_1, \ldots, g_m$
- As a whole, the circuit computes $C: \{0,1\}^n \to \{0,1\}^m$ defined by

$$C(x) = \big(g_1(x), \ldots, g_m(x)\big)$$

Equivalent: Boolean straight-line programs



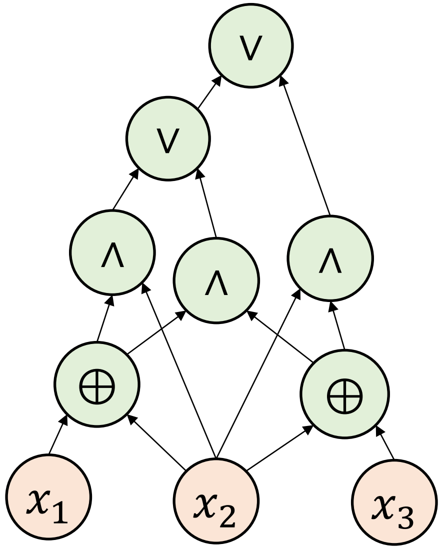
Circuit

- $x_4 \leftarrow x_1 \oplus x_2$
- $x_5 \leftarrow x_2 \oplus x_3$
- $x_6 \leftarrow x_4 \wedge x_2$
- $x_7 \leftarrow x_4 \wedge x_5$
- $x_8 \leftarrow x_2 \wedge x_5$
- $x_9 \leftarrow x_6 \vee x_7$
- $x_{10} \leftarrow x_9 \vee x_8$
- Return x_{10}

Boolean Straight-Line Program

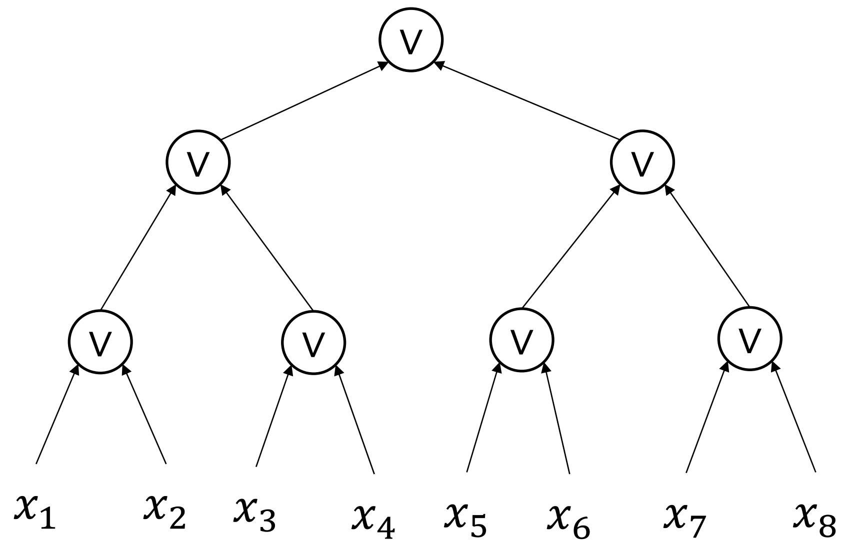
- Each line: Combine two variables, store in new variable
- “Return” at end
- No loops
- No “if” statements
- No branching

Circuit complexity



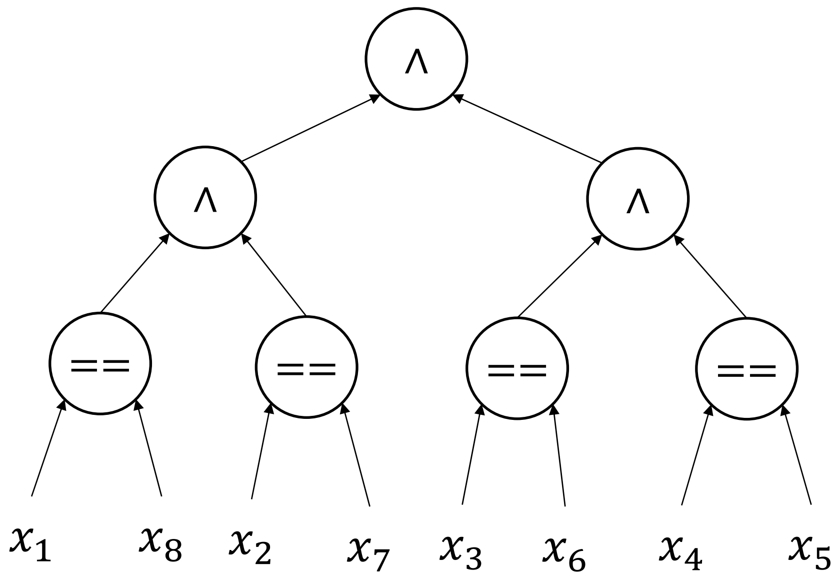
- The size of a circuit is the total number of gates
  - How much "work" does the circuit do?
- Let $f: \{0,1\}^n \to \{0,1\}^m$
- The circuit complexity of f is the size of the smallest circuit that computes f
  - How much work is required to compute f?

Circuit complexity example 1



- Let $f(x) = x_1 \vee x_2 \vee \cdots \vee x_n$
- Circuit complexity: $\Theta(n)$

Circuit complexity example 2



- Define $f: \{0,1\}^n \to \{0,1\}$ by

$f(x) = 1 \Leftrightarrow x$ is a palindrome

- Circuit complexity: $\Theta(n)$

What is the circuit complexity of f?

A: $\Theta(n^2)$

B: $O(1)$

C: $\Theta(n)$

D: $\Theta(2^n)$

Respond at PollEv.com/whoza or text “whoza” to 22333

The power of Boolean circuits

- Recall: Some languages cannot be decided by algorithms
- Are there functions that cannot be computed by circuits?

Theorem: For every $f : \{0,1\}^n \to \{0,1\}$, there exists a Boolean formula that computes f.

Theorem: For every $f: \{0,1\}^n \to \{0,1\}$, there exists a Boolean formula that computes f.

- **Proof (1 slide):** For each $z \in \{0,1\}^n$, construct T_z that is satisfied only by z
  - E.g., $T_{010} = \overline{x}_1 \wedge x_2 \wedge \overline{x}_3$

$$\text{Then } f(x) = \bigvee_{z \in f^{-1}(1)} T_z(x)$$

DNF formulas

- **Definition:** A literal is a variable or its negation (x_i or $\bar{x}_i$)
- **Definition:** A term is a conjunction of literals (AND of literals). Example:

$$T_{010} = \bar{x}_1 \wedge x_2 \wedge \bar{x}_3$$

- **Definition:** A disjunctive normal form (DNF) formula is a disjunction of terms (OR of ANDs of literals). Example:

$$f(x) = (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3)$$

Every function has a DNF formula

- Let $f: \{0,1\}^n \to \{0,1\}$ be any function

Theorem: There is a DNF formula that computes f,

with at most 2^n terms and n literals per term

- **Proof:** For each $z \in \{0,1\}^n$, construct a term T_z that is satisfied only by z

$$\text{Then } f(x) = \bigvee_{z \in f^{-1}(1)} T_z(x)$$

CNF formulas

- **Definition:** A clause is a disjunction of literals (OR of literals). Example:

$$C = \bar{x}_1 \vee x_2 \vee \bar{x}_3$$

- **Definition:** A conjunctive normal form (CNF) formula is a conjunction of clauses (AND of ORs of literals). Example:

$$f(x) = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

Every function has a CNF formula

- Let $f: \{0,1\}^n \to \{0,1\}$ be any function

Theorem: There is a CNF formula that computes f,

with at most 2^n clauses and n literals per clause

- **Proof:** For each $z \in \{0,1\}^n$, construct a clause C_z that is violated only by z
  - E.g., $T_{010} = x_1 \vee \bar{x}_2 \vee x_3$

$$\text{Then } f(x) = \bigwedge_{z \in f^{-1}(0)} C_z(x)$$

Multi-output functions

Corollary: For every $f: \{0,1\}^n \to \{0,1\}^m$, there exists a circuit of size $O(m \cdot n \cdot 2^n)$ that computes f

- **Proof:** Write $f(x) = \big(f_1(x), \ldots, f_m(x)\big)$
- Each f_i can be computed by a circuit of size $O(n \cdot 2^n)$ (DNF/CNF)
- Combine those m circuits into one

Polynomial-size circuits

- Every function has a circuit 🥴
- But the circuit we constructed has exponential size ☹️
- Next: Polynomial-time algorithm $\Rightarrow$ polynomial-size circuits

Circuit complexity of a binary language

- Let $Y \subseteq \{0,1\}^*$
- For each $n \in \mathbb{N}$, we define $Y_n: \{0,1\}^n \to \{0,1\}$ by the rule

$$Y_n(w) = \begin{cases} 1 & \text{if } w \in Y \\ 0 & \text{if } w \notin Y \end{cases}$$

- **Definition:** The circuit complexity of Y is the function $S: \mathbb{N} \to \mathbb{N}$ defined by

  $S(n) =$ the size of the smallest circuit that computes Y_n

- Note: Each circuit only handles a single input length! Different from TMs

Turing machines vs. circuits

- Let M be a Turing machine that decides a language Y
- Let $T(n)$ be M's time complexity; let $S(n)$ be M's space complexity

> **Theorem:** The circuit complexity of Y is $O\big(T(n) \cdot S(n)\big)$.

- Proof (next 6 slides) is based on computation histories

Locality of computation

- Let C be a configuration of the Turing machine M
- We can write $C = c_1 c_2 \dots c_\ell$ for some $c_1, \dots, c_\ell \in \Sigma \cup Q$
- Then $\mathrm{NEXT}(C) = c_1' c_2' \dots c_\ell'$ for some $c_1', \dots, c_\ell' \in \Sigma \cup Q$
- **Exercise 10a:** If $2 \leq i \leq \ell - 2$, then

$$c_i' = \begin{cases} \text{the third symbol of } \mathrm{NEXT}(\sqcup\, c_{i-1} c_i c_{i+1} c_{i+2}) & \text{if } c_{i-1} \in Q \text{ or } c_i \in Q \text{ or } c_{i+1} \in Q \\ c_i & \text{otherwise} \end{cases}$$

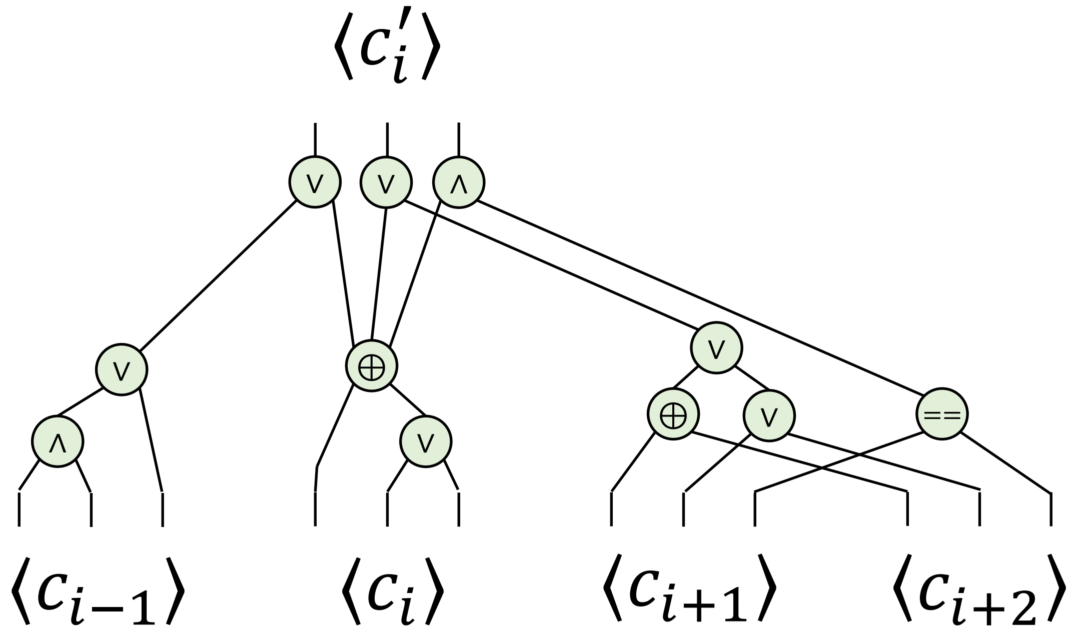
For simplicity, assume the head is not at beginning/end

Encoding configurations in binary

- Let C be a configuration of a TM M, say $C = u_1 u_2 \dots u_k q v_1 v_2 \dots v_m$
- Each symbol/state $b \in \Sigma \cup Q$ can be encoded in binary as $\langle b \rangle \in \{0,1\}^r$ for some $r = O(1)$
- We define $\langle C \rangle = \langle u_1 \rangle \langle u_2 \rangle \cdots \langle u_k \rangle \langle q \rangle \langle v_1 \rangle \cdots \langle v_m \rangle$

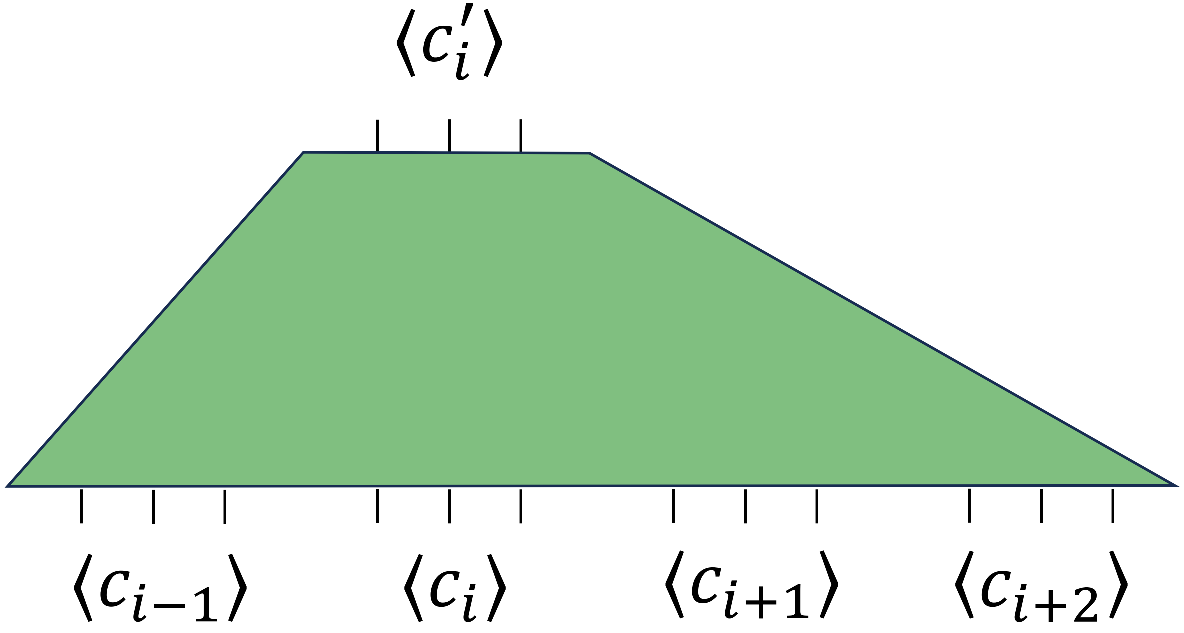
TM ⇒ Circuit

- There is a circuit C_M that computes $\langle c_i' \rangle$ given $\langle c_{i-1} \rangle$, $\langle c_i \rangle$, $\langle c_{i+1} \rangle$, $\langle c_{i+2} \rangle$

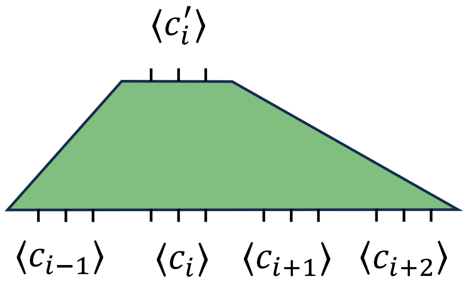


TM ⇒ Circuit

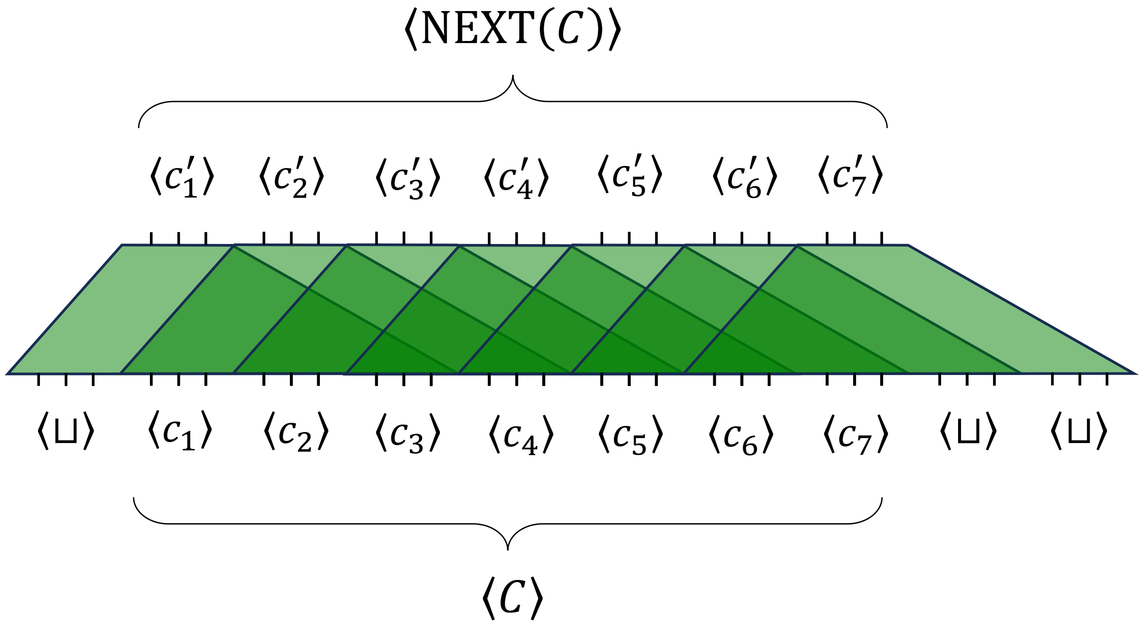
- There is a circuit C_M that computes $\langle c_i' \rangle$ given $\langle c_{i-1} \rangle, \langle c_i \rangle, \langle c_{i+1} \rangle, \langle c_{i+2} \rangle$



TM ⇒ Circuit



- There is a circuit C_M that computes $\langle c_i' \rangle$ given $\langle c_{i-1} \rangle, \langle c_i \rangle, \langle c_{i+1} \rangle, \langle c_{i+2} \rangle$
- Now let's combine many copies of C_M in parallel:



TM ⇒ Circuit

- Size: $O(S(n) \cdot T(n))$
- Assume WLOG:
  - $\langle 0 \rangle = 0^r$ and $\langle 1 \rangle = 10^{r-1}$
  - M halts in starting cell
  - $\mathrm{NEXT}(C) = C$ if C is a halting configuration
  - $\langle q_{\text{accept}} \rangle = 1^r$
  - $\langle q_{\text{reject}} \rangle = 01^{r-1}$