Computational Complexity

计算复杂性

张方国

中山大学计算机学院

isszhfg@mail.sysu.edu.cn

Lecture 3. Computational Models(Cont.)

Non-uniform Models

- Boolean Circuits
- Machines That Take Advice
- Restricted Models

Non-uniform

By a non-uniform model of computation we mean a model in which for each possible input length one considers a different computing device. That is, there is no "uniformity" requirement relating devices that correspond to different input lengths. Furthermore, this collection of devices is infinite by nature, and this collection may not even have a finite description.

In contrast to the standard (or uniform) TM model where the same TM is used on all the infinitely many input sizes, a non-uniform model allows a different algorithm to be used for each input size.

Non-uniform

In complexity theory, non-uniform models of computation are studied either towards the development of lower bound techniques or as simplified upper bounds on the ability of efficient algorithms.

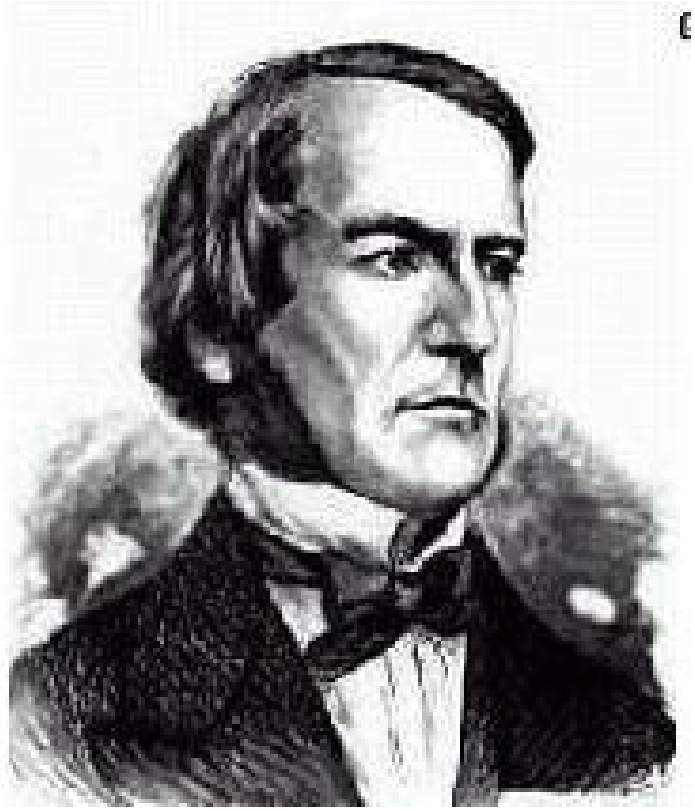
电路本身的重要性和电路复杂性的研究（如分支程序等）为非一致性模型的研究提供了新的空间

Boolean Circuits(布尔电路或布尔线路)

Boolean circuit is a generalization of Boolean formulae and a simplified model of the silicon chips used to make modern computers. It is a natural model for non-uniform computation, which crops up often in complexity theory.

This model is more adequate for the study of the evolution of computation(i.e., development of lower bound techniques).

Boolean Algebra developed by George Boole(1815 – 1864) to solve mathematical logic problems (1847)（The Mathematical Analysis of Logic）.

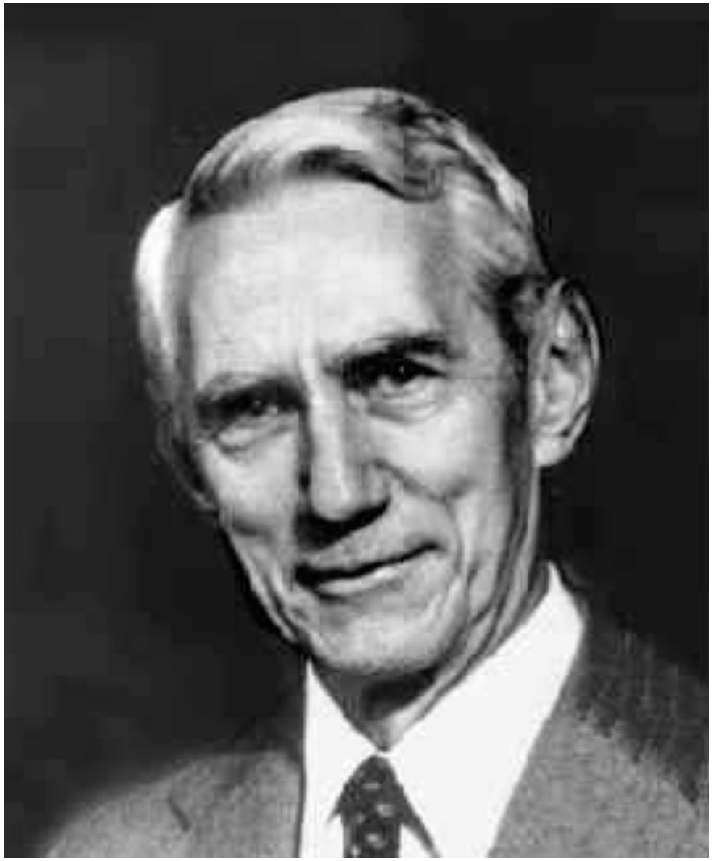


1854年，George Boole出版了《思维规律的研究》，(An Investigation of The Laws of Thought)，在这本书中布尔介绍了现在以他的名字命名的布尔代数(用数学的方法解决逻辑问题)。

Boolean variable: value is 0 or 1.

Boolean function: function whose inputs and outputs are 0, 1.

Claude E Shannon (1916-2001) first applied to digital circuits (1939).

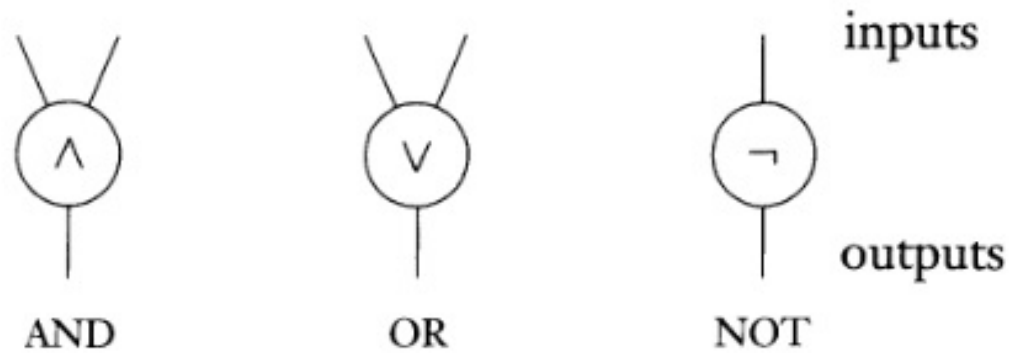


布尔代数与开关电路的结合：布尔电路

1937年，Shannon在MIT提交硕士论文《继电器和开关电路符号分析》，首次提出了可用于设计和分析逻辑电路的系统方法，是数字信号处理的里程碑之一，获1940年Afred Noble优秀论文奖。

Formal Definition of Boolean Circuits

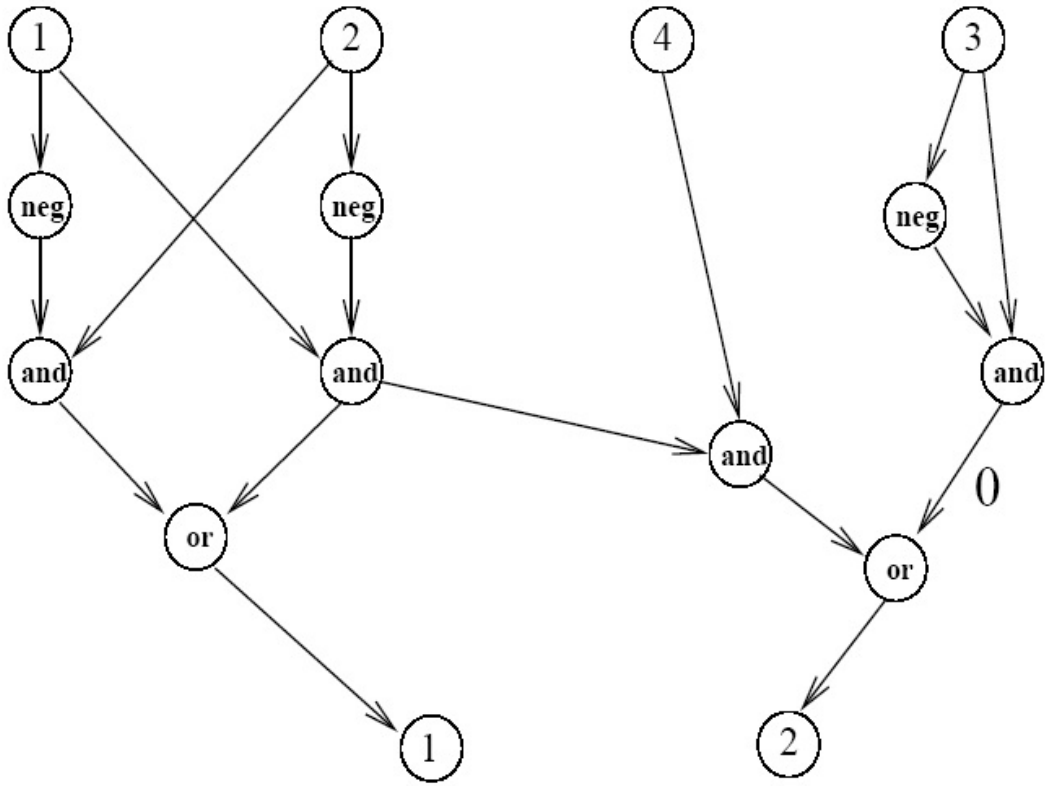
A ***Boolean circuit*** is a collection of ***gates*** and ***inputs*** connected by ***wires***. Cycles aren't permitted. Gates take three forms: AND gates, OR gates, and NOT gates, as shown schematically in the following figure.



A Boolean circuit is a directed acyclic graph with labels on the vertices. The graph's vertices are of three types: sources(源), sinks(底), and internal vertices(nodes)(内部顶点).

- Internal vertices are vertices having in-coming and out-going edges. In the context of Boolean circuits, internal vertices are called gates. Each gate is labeled by a Boolean operation: and($\wedge$), or($\vee$) and neg($\neg$) (Not).
- The graph sources (i.e., vertices with no in-going edges) are called **input terminals**. Each input terminal is labeled by a natural number(which is to be thought of the index of an input variable).

- The graph sinks (i.e., vertices with no out-going edges) are called output terminals.Each output terminal is labeled by a natural number such that if the circuit has m output terminals then they are labeled $1, 2, \ldots, m$.



Definition (Boolean Circuit)

A Boolean circuit with n different input labels and m output terminals induces (and indeed computes) a function from $\{0,1\}^n$ to $\{0,1\}^m$ defined as follows: For any fixed string $x \in \{0,1\}^n$, we iteratively define the value of vertices in the circuit such that the input terminals are assigned the corresponding bits in $x = x_1 \cdots x_n$ and the values of other vertices are determined in the natural manner. That is:

- An input terminal with label $i \in \{1, 2, \ldots, n\}$ is assigned the i^{th} bit of x (i.e., the value x_i).

- If the children of a gate(of in-degree d) that is labeled $\wedge$ have values $v_1, v_2, \ldots, v_d$, then the gate is assigned the value $\wedge_{i=1}^{d} v_i$. The value of a gate labeled $\vee$(or $\neg$) is determined analogously.

有效布尔电路的构造和属性有实践重要性，因为它们是计算机的构建块。

The value of the circuit on input x (i.e., the output computed by the circuit on input x) is $y = y_1 \cdots y_m$, where y_i is the value assigned by the foregoing process to the output terminal labeled i.

一个布尔线路计算一个布尔函数。等价的布尔线路计算同一个函数。

We note that there exists a polynomial-time algorithm that, given a circuit C and a corresponding input x outputs the value of C on input x. This algorithm determines the values of the circuit's vertices, going from the circuit's input terminals to its output terminals.

Definition (Circuit families)

We say that a family of circuits $(C_n)_{n\in\mathbb{N}}$ computes a function $f : \{0,1\}^* \to \{0,1\}^*$ if for every n the circuit C_n computes the restriction of f to strings of length n. In other words, for every $x \in \{0,1\}^*$, it must hold that $C_{|x|}(x) = f(x)$.

The vertices labeled with $\wedge$ and $\vee$ have fan-in (i.e., number of incoming edges) equal to 2 and the vertices labeled with $\neg$ have fan-in 1.

Fan-in is the number of inputs of an electronic logic gate(逻辑门的扇入是指其所有输入的个数).

A related concept is fan-out（扇出）, which is the number of logic inputs that a given logic output drives.

We will be most interested in circuits in which each gate has at most two in-coming edges.

Such circuits are called circuits of **bounded fan-in**. In contrast, other studies are concerned with circuits of **unbounded fan-in**, where each gate may have an arbitrary number of incoming edges.

Circuit size as a complexity measure

Given a Boolean circuit C, the size of C is the number of **gates** in C, and the depth of C is the length of the longest path from any input to the output. （定义其所有逻辑门的个数为线路规模，布尔线路的深度是指其从变量到输出门的最长路径的长度。

电路复杂性研究主要研究"计算一给定的布尔函数"的电路大小和深度界限。除了它们的实践价值外，这种界限紧密的联系到关于图灵机计算的重要问题。

When considering a family of circuits $(C_n)_{n\in\mathbb{N}}$ that computes a function $f : \{0,1\}^* \to \{0,1\}^*$, we are interested in the size of C_n as a function of n. Specifically, we say that this family has size complexity $s : \mathbb{N} \to \mathbb{N}$ if for every n the size of C_n is $s(n)$.

The circuit complexity of a function f denoted s_f, is the infimum of the size complexity of all families of circuits that compute f. Alternatively, for each n we may consider the size of the smallest circuit that computes the restriction of f to n bit strings (denoted f_n), and set $s_f(n)$ accordingly. We stress that non-uniformity is implicit in this definition, because no conditions are made regarding the relation between the various circuits used to compute the function on different input lengths.

Some simple facts about circuit complexity of functions

- Most importantly, any Boolean function can be computed by some family of circuits, and thus the circuit complexity of any function is well defined. Furthermore, each function has at most exponential circuit complexity.
- Some functions have polynomial circuit complexity. In particular, any function that has time complexity t has circuit complexity $poly(t)$. Furthermore, the corresponding circuit family is uniform.

  Simulation of Turing Machines by Circuits.

Some simple facts about circuit complexity of functions

- Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by some circuit of size 4×2^n.

  Let $s(n)$ be the minimal size of a circuit necessary to compute all $f : \{0,1\}^n \to \{0,1\}$. Assume $n > 0$. Given a function $f : \{0,1\}^n \to \{0,1\}$, consider the functions $f_0, f_1 : \{0,1\}^{n-1} \to \{0,1\}$ where $f_b(x_1, ..., x_{n-1}) = f(x_1, ..., x_{n-1}, b)$. Then

  $$f(x_1, ..., x_n) = \overline{(x_n} \wedge f_0(x_1, ..., x_{n-1}, 0)) \vee (x_n \wedge f_0(x_1, ..., x_{n-1}, 1))$$

  Therefore, $s(n) \leq 2 \times s(n-1) + 4$. Solving this recursion yields $s(n) \leq 2^n(s(0) + 4) - 4 \leq 4 \times 2^n$.

Some simple facts about circuit complexity of functions

- Almost all Boolean functions have exponential circuit complexity. Specifically, the number of functions mapping $\{0,1\}^n$ to $\{0,1\}$ that can be computed by some circuit of size s is smaller than s^{2s}.

所以只有极少数的布尔函数可以被小规模电路所计算。我们至今未能证明任何具体的布尔函数的布尔电路规模必为超线性函数。已知最好下界是$3n - o(n)$。

“几乎所有的n元布尔函数f均满足$C(f) = \Omega(2^n/n)$”

令s为正整数。我们首先计算含n个变量规模为s的所有布尔电路的个数。因为每个逻辑门或为$\vee$或为$\wedge$，且所有门的扇入不大于2，所以每个门恰好有2个输入。因为每个输入都来自于一个逻辑门（共有s种选择），或是一个文字（共有2n种选择），或是一个布尔常数（有2种选择），所以每个输入有$s + 2n + 2$种选择。由此推出，含n个变量且规模为s的布尔线路的总个数不超过$(2(2 + s + 2n)^2)^s$。因此，所有含n个变量且规模不大于s的布尔线路的总个数最多为$s(2(2 + s + 2n)^2)^s$。

令$s = 2^n/(8n)$，则$s(2(2 + s + 2n)^2)^s \ll 2^{2^n}$。因为我们总共有$2^{2^n}$个n元布尔函数，所以当n趋于无穷大时，n元布尔函数可被规模小于$s = 2^n/(8n)$ 的布尔线路所计算的概率趋于0.

AC^0 and NC^1

The **depth** of a circuit is the layer in which the output gate is found. Alternatively, it is the length of the longest directed path in the underlying graph.

The class AC^0 consists of those decision problems L that admit polynomial-size circuit families $\{C_0, C_1, ...\}$ such that the depth of C_i is at most c for come constant c.

The class NC^1 consists of those decision problems L that admit circuit families $\{C_0, C_1, ...\}$ such that every gate in every circuit in the family has fan-in 2 and the depth of C_i is at most $O(\log n)$.

The fact that NC^1 circuit families have polynomial size is automatic: If every gate has fan-in 2 and the depth is $O(\log n)$, then the circuit size is at most $2^{O(\log n)} = n^{O(1)}$.

$$AC^0 \subseteq NC^1$$

The class NC: For every d, a language L is in NC^d if L can be decided by a family of circuits $\{C_n\}$ where C_n has $poly(n)$ size and depth $O(\log^d n)$. The class NC is $\cup_{i \geq 1} NC^i$.

A problem has efficient parallel algorithms iff it is in NC.

Simulation of Turing Machines by Circuits

N. Pippenger and M.J. Fischer. Relations among complexity measures. Journal of the ACM, Vol.26(2), pages 361-381, 1979.

Theorem

Let M be a Turing Machine. For every $n \geq 0$, the machine defines a Boolean function: $f_{M,n} : \{0,1\}^n \rightarrow \{0,1\}$ (The function $f_{M,n}$ maps each string over $\{0,1\}^n$ accepted by M to 1 and all the rest to 0). Let $T(n)$ be a bound on the number of steps, that M executes on inputs of length n. There exists a series of circuits: $C_1, C_2, \ldots$ that computes the functions: $f_{M,1}, f_{M,2}, \ldots$ such that the size of the n^{th} circuit is $O(T(n)logT(n))$.

Proof: Let M be a quintuple $M = (\Sigma, Q, q_0, q_1, \delta)$, where

- Σ is a finite alphabet
- Q is a finite set of states
- $q_0 \in Q$ is the initial state
- $q_1 \in Q$ is the accepting state
- $\delta : Q \times \Sigma \longrightarrow Q \times \Sigma \times \{L, R\}$ is the transition function.

Each state in Q will be represented by $\lceil \log |Q| \rceil$ bits, and each letter of Σ by $\lceil \log |\Sigma| \rceil$ bits. Using this representation, δ can be seen as a Boolean function of a *constant* number of bits, and may be implemented by a circuit of constant size, as shown in Figure 3.1, which is labeled δ.

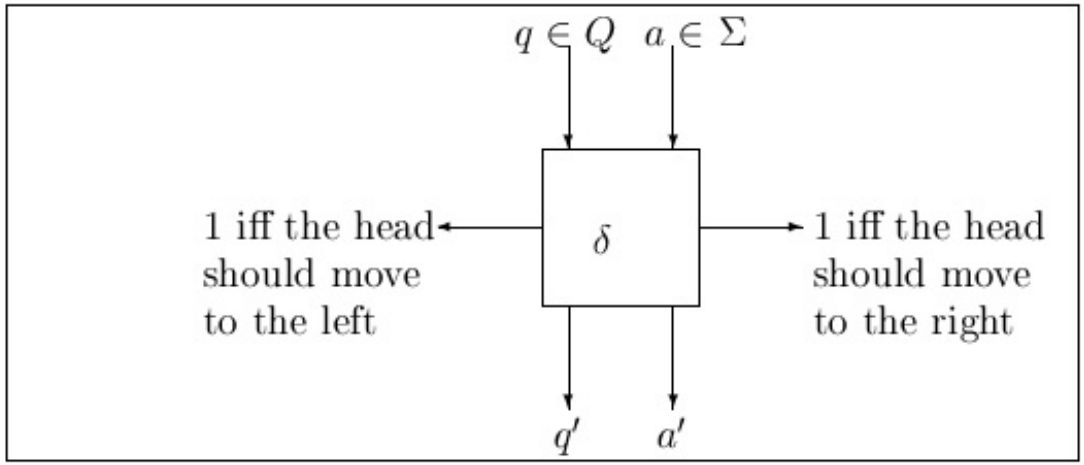


Figure 3.1: The circuit δ

The circuit δ is a basic building block in a circuit that simulates M. Let $x_1\, x_2 \ldots, x_n$ be binary variables representing the first n binary digits in the representation of the input written on the tape of M, before it starts executing. The circuit simulating M will receive as input the variables x_i, and compute the function calculated by M. It will have $T(n)$ different layers, one for each time unit $0 \leq t \leq T(n)$. The binary values at each layer of the circuit encode a 'configuration' of the machine (its state, tape content, and the location of its head). It is clearly sufficient to include in such a configuration, the content of the tape cells $-T(n) \leq i \leq T(n)$ only. For each point of time, $0 \leq t \leq T(n)$, for every cell of the tape $-T(n) \leq i \leq T(n)$, we keep a binary array, shown in Figure 3.2.

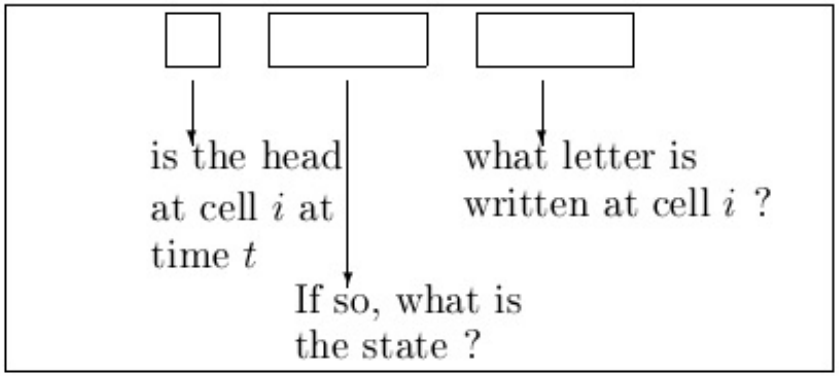


Figure 3.2: The binary array

The circuit computes these arrays, according to the following layers:

First layer: For $t = 0$, these arrays are composed of the variables $x_1 x_2 \ldots, x_n$, and constants.

Intermediate layers: Given the arrays for some time t, the arrays for time $t + 1$, are constructed in parallel, using γ circuits, similar to δ (see Figure 3.3).

Each γ circuit gets as input one of the arrays of time t, and creates, at its output, the corresponding array of time $t + 1$. If the head is in the cell, for which the γ circuit

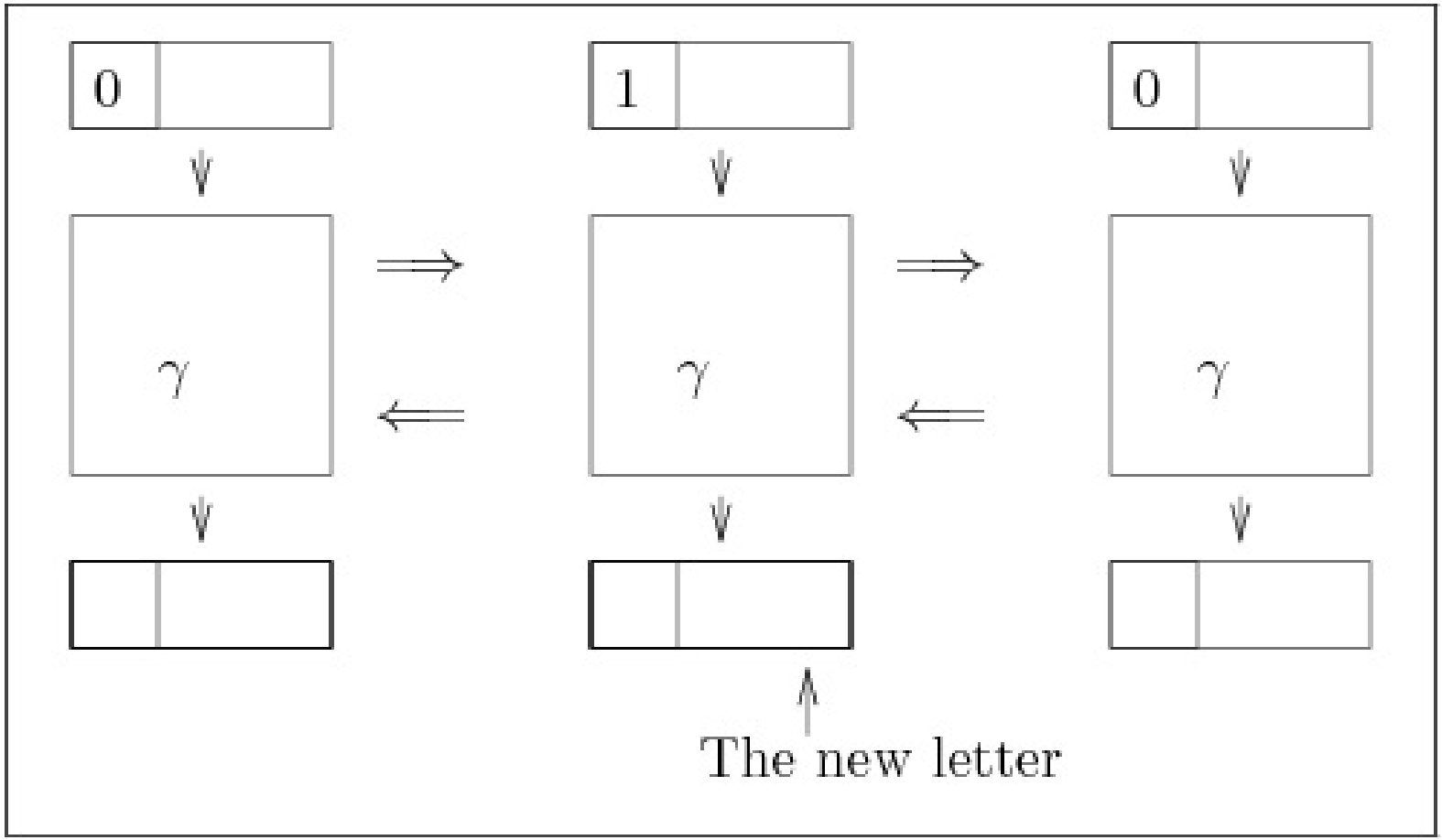


Figure 3.3: A circuit to simulate M

is 'responsible', the array is changed according to δ, and the appropriate neighboring γ circuit is notified of the head movement. This neighbor then updates its array accordingly. All other γ circuits output their input array, without change.

A difference between the original transition function of M, and the circuit γ, is that once M reaches q_1, the accepting state, it halts. The circuit, on the other hand, cannot stop, so when a γ circuit gets as input an array, in which the encoded state is q_1, the array is left unchanged, and the head is not moved.

Last layer: The last layer in the construction is a circuit, that checks whether the state, encoded at the array where the head bit is 1, is q_1.

The size of the circuit, thus constructed, is $O(T^2(n))$. It is interesting, that its depth is $O(T(n))$, the same as M's time complexity. □

Uniform families: A family of polynomial size circuits $(C_n)_{n\in\mathbb{N}}$ is called uniform if given n one can construct the circuit C_n in $poly(n)$-time.

Note that if a function is computable by a uniform family of polynomial-size circuits then it is computable by a polynomial-time algorithm.

This algorithm first constructs the adequate circuit(which can be done in polynomial time by the uniformity hypothesis), and then evaluate this circuit on the given input(which can be done in time that is polynomial in the size of the circuit).

Turing machines that take advice

In Computational complexity theory, an **advice string** is an extra input to a Turing machine which is allowed to depend on the length n of the input, but not on input itself.

Turing machines that "take advice" : such a machine has, for each n, an advice string α_n, which it is allowed to use in its computation whenever the input has size n.

Definition (taking advice)

We say that algorithm A computes the function f using advice of length $l : \mathbb{N} \to \mathbb{N}$ if there exists an infinite sequence $(\alpha_n)_{n\in\mathbb{N}}$ such that:

- For every $x \in \{0,1\}^*$, it holds that $A(\alpha_{|x|}, x) = f(x)$.
- For every $n \in \mathbb{N}$, it holds that $|\alpha_n| = l(n)$.

The sequence$(\alpha_n)_{n\in\mathbb{N}}$ is called the advice sequence.

Theorem (the power of advice)

There exist functions that can be computed using one-bit advice but cannot be computed without advice.

Proof:

Restricted models

- Boolean formulae

  A Boolean formula is a circuit in which all non-sink vertices have out-degree 1, which means that the underlying graph is a tree, and the formula as an expression can be read by traversing the tree ,and registering the vertices' labels in the order traversed

- Formulae in CNF and DNF

  conjunctive normal form(CNF) 合取范式

$$\varphi(x_1, x_2, ..., x_n) = \bigwedge_{i=1}^{m} (\bigvee_{j=1}^{k_i} l_{i,j})$$

  disjunctive normal form (DNF) 析取范式

$$\varphi(x_1, x_2, ..., x_n) = \bigvee_{i=1}^{m} (\bigwedge_{j=1}^{k_i} l_{i,j})$$

- Constant-depth circuits

  Circuits have a natural structure(as graphs). One natural parameter regarding this structure is the depth of a circuit, which is defined as the longest directed path from any source to any sink.

- Monotone circuits（单调线路）

  A circuit is said to be monotone if it consists of only AND and OR gates.

定理：布尔函数f可被单调线路所计算当且仅当f为单调递增。

A boolean function f is monotone if $f(x) \leq f(y)$ as long as $x_i \leq y_i$ for all i.

已经可以证明，存在某些函数（例如CLIQUE问题-Razbarov定理），任何计算它的的单调线路的规模必有指数下界。

单调电路域一般布尔电路还是有很大区别的：我们已知存在布尔函数f可被多项式规模的布尔电路所计算，但f只能在指数规模的单调线路下可计算。

Thank You Very Much!