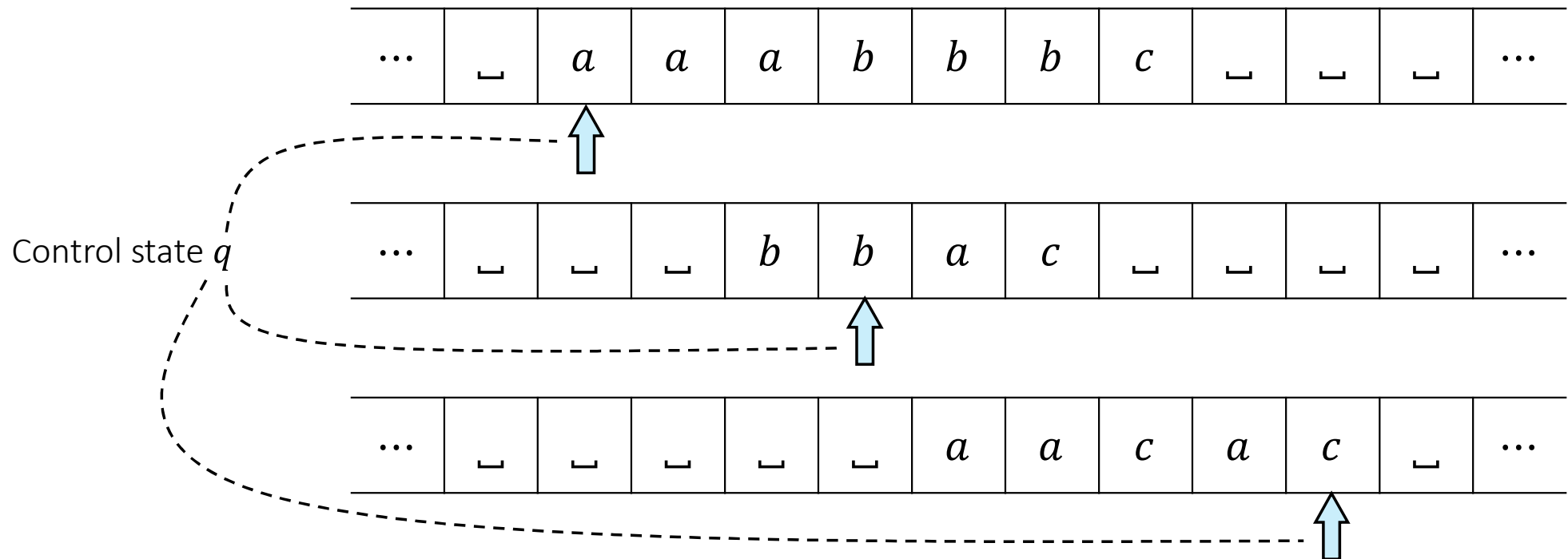


CS 483 - Introduction to the Theory of Computation

Lecture 11

Multi-Tape Turing Machine

Turing machine can be generalized to use multiple tapes, each equipped with a head that moves independently.



Multi-Tape Turing Machine

For a constant k , a **k -tape Turing machine** $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ is defined similarly as a single-tape TM, but with transition function

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\leftarrow, \rightarrow\}^k$$

Initially the first tape contains the input w and the rest are empty.

Multi-Tape Turing Machine

For a constant k , a **k -tape Turing machine** $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ is defined similarly as a single-tape TM, but with transition function

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\leftarrow, \rightarrow\}^k$$

Initially the first tape contains the input w and the rest are empty.

At every step, the transition

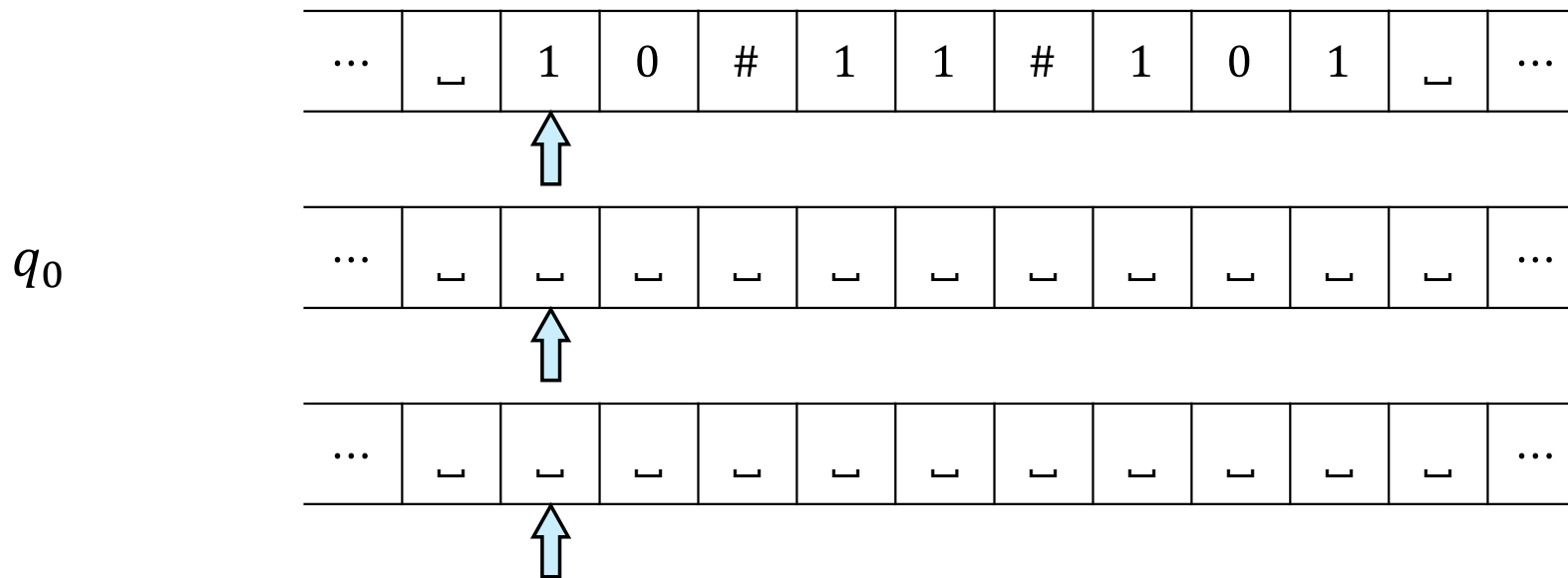
$$\delta(q, s_1, \dots, s_k) = (q', s'_1, \dots, s'_k, D_1, \dots, D_k)$$

means that if the current state is q and the head on the i -th tape points at symbol s_i , it is rewritten to s'_i and the head moves in direction D_i , while the new state is q' .

Example: Binary Addition

$$L = \{x\#y\#z \mid x, y, z \in \{0,1\}^*, x + y = z \text{ as binary numbers}\}$$

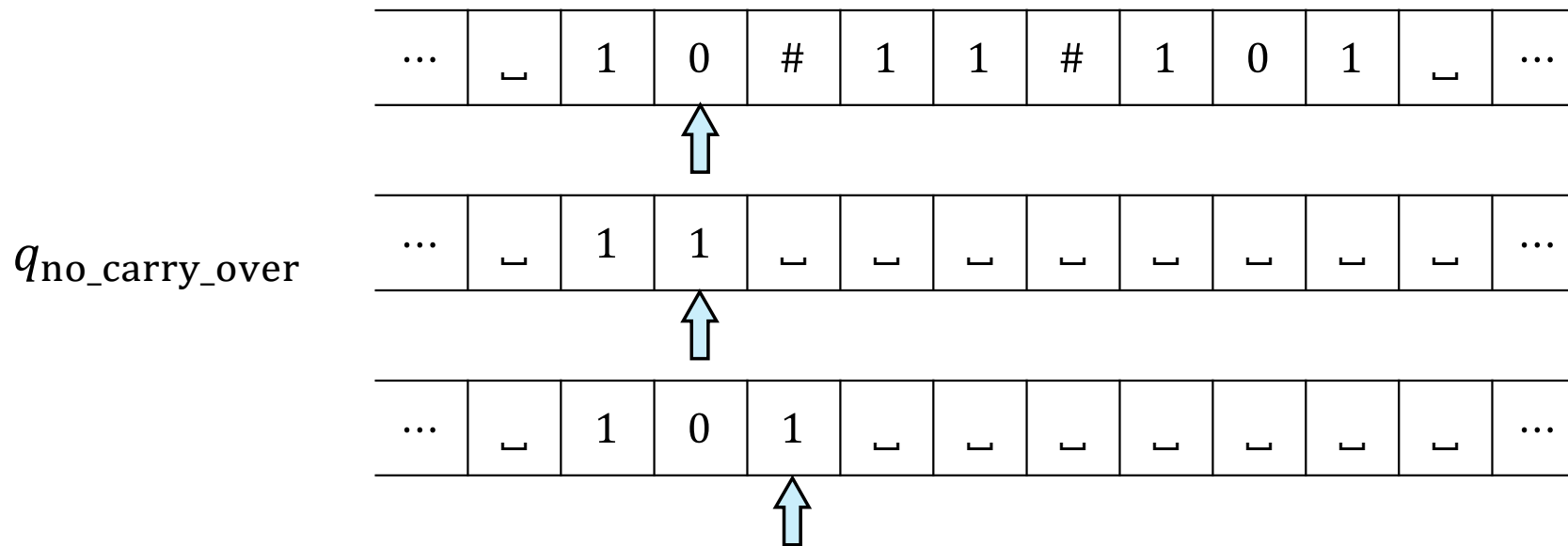
Stage 1: Copy y and z to tape 2 and 3;



Example: Binary Addition

$$L = \{x\#y\#z \mid x, y, z \in \{0,1\}^*, x + y = z \text{ as binary numbers}\}$$

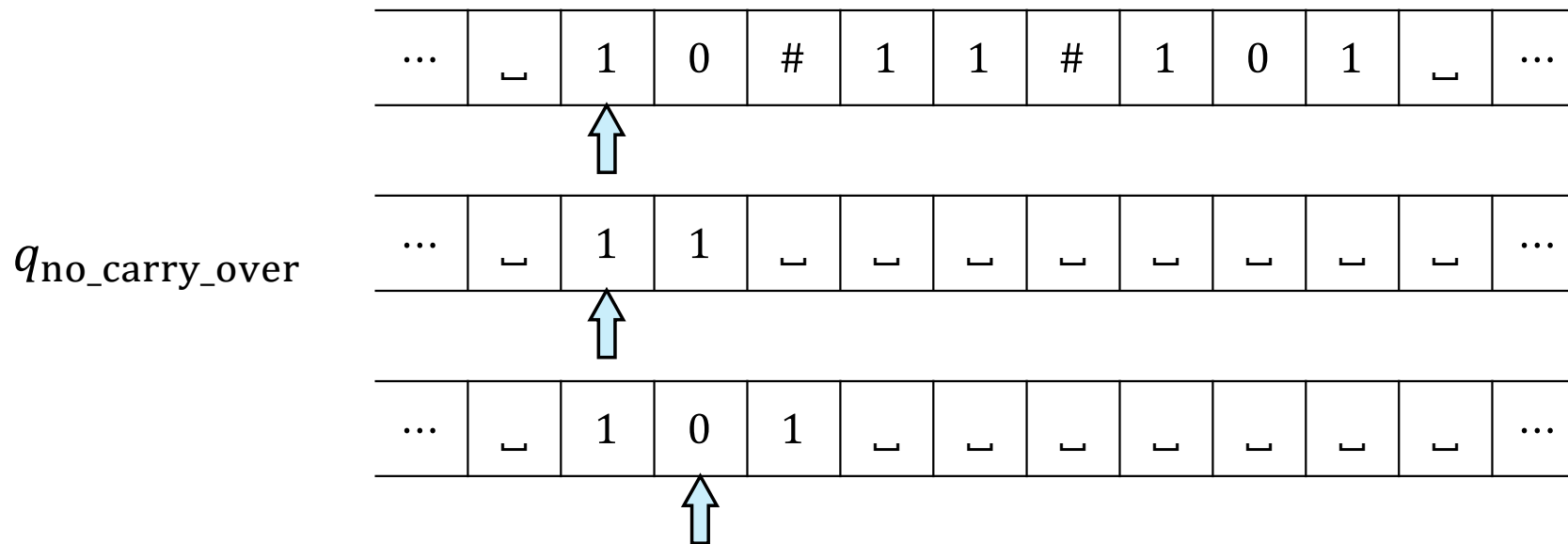
Stage 2: Check each bit from right to left (use two states for carry over).



Example: Binary Addition

$$L = \{x\#y\#z \mid x, y, z \in \{0,1\}^*, x + y = z \text{ as binary numbers}\}$$

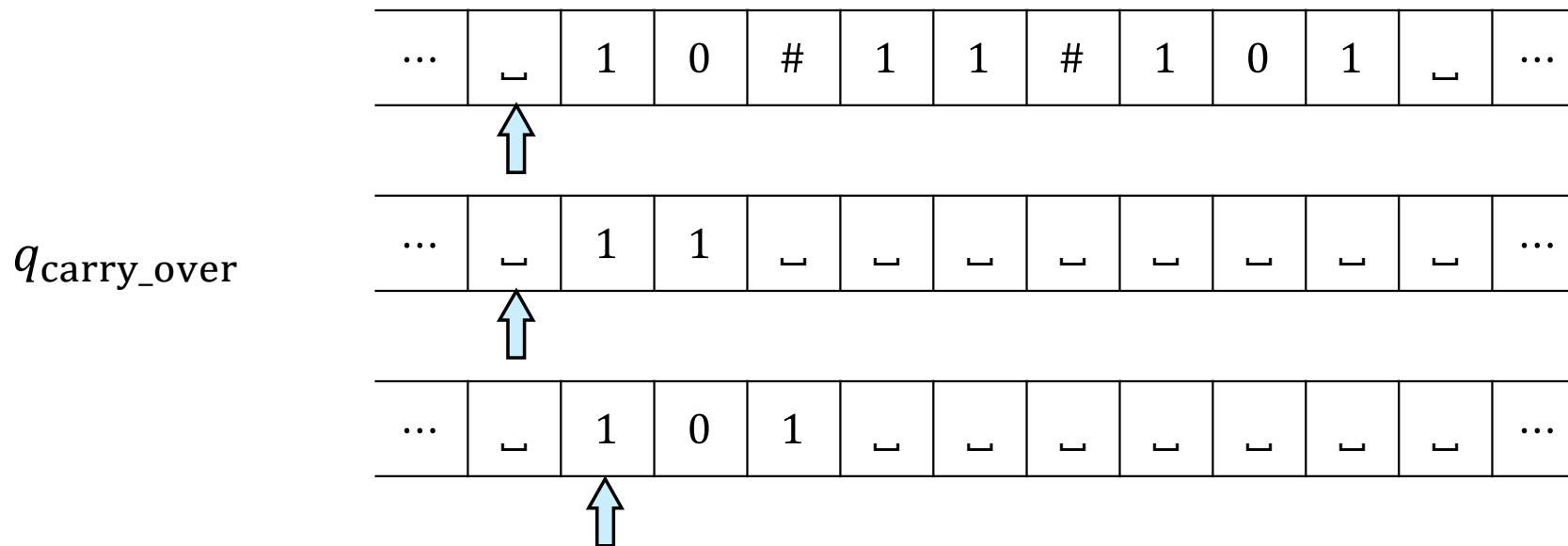
Stage 2: Check each bit from right to left (use two states for carry over).



Example: Binary Addition

$$L = \{x\#y\#z \mid x, y, z \in \{0,1\}^*, x + y = z \text{ as binary numbers}\}$$

Stage 2: Check each bit from right to left (use two states for carry over).



Power of Multi-Tape Turing Machine

Claim: Every program in assembly language can be simulated by a multi-tape Turing machine.

Power of Multi-Tape Turing Machine

Claim: Every program in assembly language can be simulated by a multi-tape Turing machine.

- Assembly language operands: **REG1, REG2, ... ,REGk, MEMORY[]**
 - Each register is simulated by a tape of finite length;
 - The memory is simulated by a infinite tape divided into words.

Power of Multi-Tape Turing Machine

Claim: Every program in assembly language can be simulated by a multi-tape Turing machine.

- Assembly language operands: **REG1, REG2, ... ,REGk, MEMORY[]**
 - Each register is simulated by a tape of finite length;
 - The memory is simulated by a infinite tape divided into words.
- Assembly language instructions: **mov, add, sub, mul, xor, ...**
- Assembly language control flow: labels and **jmp**
 - Each instruction or label corresponds to a unique state

Power of Multi-Tape Turing Machine

Claim: Every program in assembly language can be simulated by a multi-tape Turing machine.

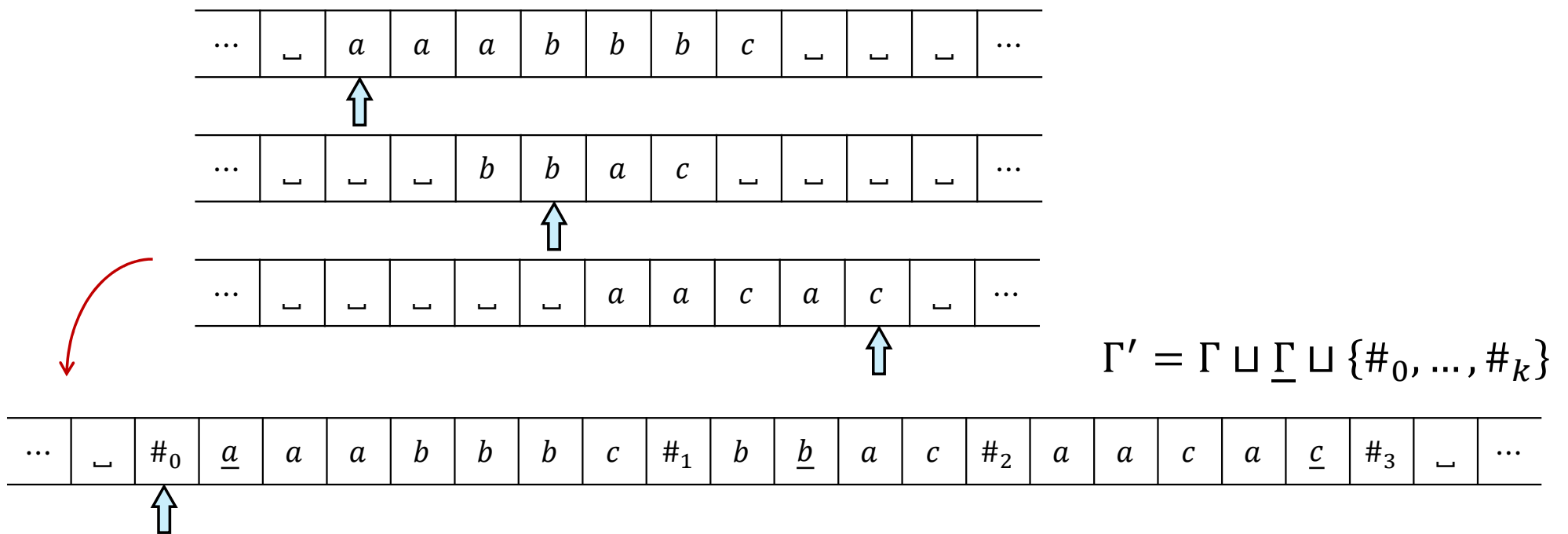
Corollary: Every program in your favorite programming language can be simulated by a multi-tape Turing machine.

Power of Single-Tape Turing Machine

Theorem. Every multi-tape Turing machine M can be simulated by a single-tape Turing machine M' .

Power of Single-Tape Turing Machine

Theorem. Every multi-tape Turing machine M can be simulated by a single-tape Turing machine M' .



Power of Single-Tape Turing Machine

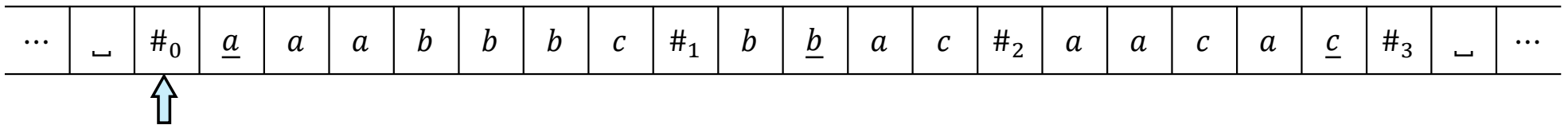
To simulate one transition of M :

$$\delta(q, s_1, \dots, s_k) = (q', s'_1, \dots, s'_k, D_1, \dots, D_k)$$

M' enters the state $(q', s'_1, \dots, s'_k, D_1, \dots, D_k, 0)$ and scans to the right.

Whenever it reaches a marked symbol it:

- Rewrites the symbol;
- Moves the mark;
- Potentially shifts the other tape content by one cell;
- Increase the last component and continue the scan.



Church-Turing Thesis

The **output** of a Turing machine M on input w is

$$M(w) = \begin{cases} \text{content on the last tape} & \text{when } M \text{ halts} \\ \perp \text{ (not defined)} & \text{if } M \text{ does not halt} \end{cases}$$

This defines a partial function $M: \Sigma^* \rightarrow \Gamma^*$. A partial function F is **computable** if there exists a Turing machine M such that $F(w) = M(w)$ for every w in the domain of F .

Church-Turing Thesis

*Every physically realizable computational model
can be simulated by a Turing machine.*

or equivalently,

*A function is physically computable if and only if
it is computable by a Turing machine.*

Church-Turing thesis is **NOT** a mathematical statement! “Physically realizable” or “physically computable” are not formally defined.

Nondeterministic Turing Machine

A **nondeterministic Turing machine (NTM)** $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ is defined similarly as a normal TM, but with transition function

$$\delta: Q \times \Gamma^k \rightarrow 2^{Q \times \Gamma^k \times \{\leftarrow, \rightarrow\}^k}$$

Nondeterministic Turing Machine

A **nondeterministic Turing machine (NTM)** $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ is defined similarly as a normal TM, but with transition function

$$\delta: Q \times \Gamma^k \rightarrow 2^{Q \times \Gamma^k \times \{\leftarrow, \rightarrow\}^k}$$

On input $w \in \Sigma^*$, the NTM

- Accepts, if **there exists** a sequence of transitions reaching q_{acc} .
- Rejects, if **every** sequence of transitions reaches q_{rej} .

Nondeterministic Turing Machine

The set of inputs **accepted/recognized** by M is $L(M)$. If M either accepts or rejects on every input, we say $L(M)$ is **decided/computed** by M .

Theorem. A language is recognizable (resp. decidable) if and only if it is recognized (resp. decided) by a nondeterministic Turing machine.

Nondeterministic Turing Machine

The set of inputs **accepted/recognized** by M is $L(M)$. If M either accepts or rejects on every input, we say $L(M)$ is **decided/computed** by M .

Theorem. A language is recognizable (resp. decidable) if and only if it is recognized (resp. decided) by a nondeterministic Turing machine.

The “only if” direction is trivial. We will prove the “if” direction by simulating an NTM M with a TM M' .

Nondeterministic Turing Machine

M' use **breadth-first search** on the non-deterministic choices of M .

Nondeterministic Turing Machine

M' use **breadth-first search** on the non-deterministic choices of M .

Description of $M'(w)$:

- For every $t = 0, 1, 2, \dots$
 - For every $1 \leq a_1, a_2, \dots, a_t \leq |Q \times \Gamma^k \times \{\leftarrow, \rightarrow\}^k|$:
 - Run $M(w)$ for t steps, with the i -th transition using the a_i -th choice.
 - Accepts if it accepted.
 - Rejects if $M(w)$ rejected in t steps for all choices.

Closure Properties

Decidable languages are closed under:

Closure Properties

Decidable languages are closed under:

- intersection: Assume L_1 is decided by M_1 and L_2 is decided by M_2 .

Description of $M(w)$ that decides $L_1 \cap L_2$:

- Run $M_1(w)$ and then $M_2(w)$; accept if both accepted.
- Otherwise reject.

Closure Properties

Decidable languages are closed under:

- intersection
- union: Assume L_1 is decided by M_1 and L_2 is decided by M_2 .

Description of $M(w)$ that decides $L_1 \cup L_2$:

- Run $M_1(w)$ and accept if it accepted;
- Run $M_2(w)$ and accept if it accepted;
- Otherwise reject.

Closure Properties

Decidable languages are closed under:

- intersection
- union
- concatenation: Assume L_1 is decided by M_1 and L_2 is decided by M_2 .

Description of $M(w)$ that decides L_1L_2 :

- For every $i = 0, 1, \dots, n = |w|$:
 - Run $M_1(w_1 \dots w_i)$ and then $M_2(w_{i+1} \dots w_n)$; accept if both accepted.
- Otherwise reject.

Closure Properties

Decidable languages are closed under:

- intersection
- union
- concatenation
- complement: Assume L is decided by M .

Description of $M'(w)$ that decides $\bar{L}$:

- Run $M(w)$; reject if it accepted, and accept if it rejected.

Closure Properties

Decidable languages are closed under:

- intersection
- union
- concatenation
- complement
- ...

Closure Properties

Computable functions are closed under composition:

Assume F is computed by M_1 and G is computed by M_2 .

Description of $M(w)$ that computes $G(F(w))$:

- Run $M_1(w)$ and let v be its output;
- Run $M_2(v)$ and output its output.

Closure Properties

Are recognizable languages closed under the following operations?

- intersection ✓
- union: Assume L_1 is recognized by M_1 and L_2 is recognized by M_2 .

Description of $M(w)$ that decides $L_1 \cup L_2$:

- For each $t = 0, 1, 2, \dots$
 - Run $M_1(w)$ for t steps and accept if it accepted;
 - Run $M_2(w)$ for t steps and accept if it accepted.

Closure Properties

Are recognizable languages closed under the following operations?

- intersection ✓
- union ✓
- concatenation

Closure Properties

Are recognizable languages closed under the following operations?

- intersection ✓
- union ✓
- concatenation ✓
- complement

Closure Properties

Are recognizable languages closed under the following operations?

- intersection ✓
- union ✓
- concatenation ✓
- complement ✗

Counterexample: $A_{TM} = \{(M, w) \mid \text{Turing machine } M \text{ accepts } w\}$