

CS 483 - Introduction to the Theory of Computation

Lecture 10

Non-Context-Free Languages Revisited

How does one compute the language

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}?$$

Non-Context-Free Languages Revisited

How does one compute the language

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}?$$

Answer 1: Use two stacks! (2PDA; see homework)

Non-Context-Free Languages Revisited

How does one compute the language

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}?$$

Answer 1: Use two stacks! (2PDA; see homework)

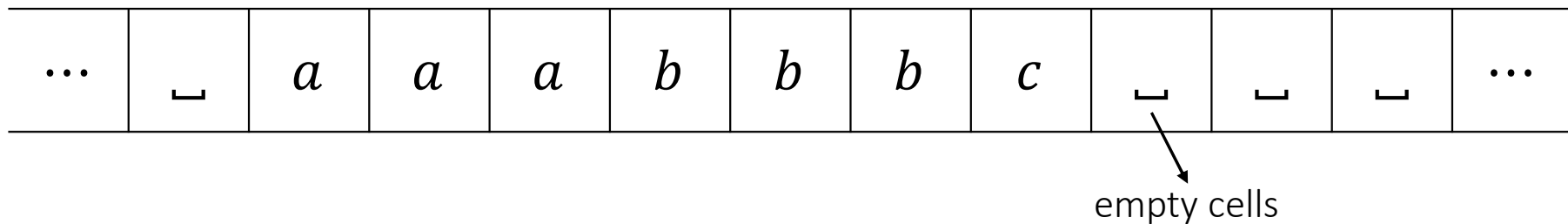
Answer 2: Actually store n .

This requires a **memory** that can be arbitrarily accessed.

Modeling the Memory: Tapes

A **tape** contains:

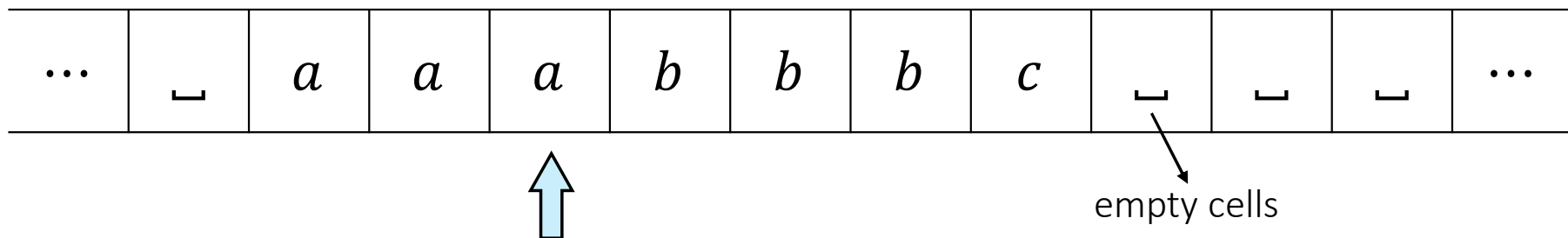
- A two-way infinite sequence of **cells**, that each holds a symbol or is empty;



Modeling the Memory: Tapes

A **tape** contains:

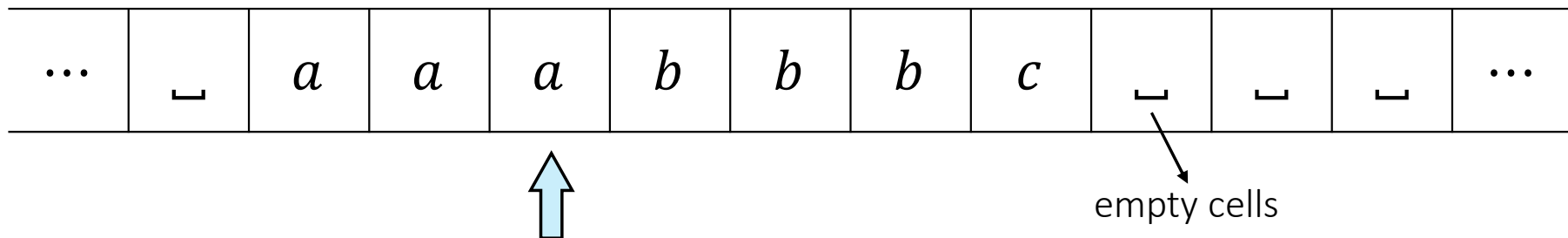
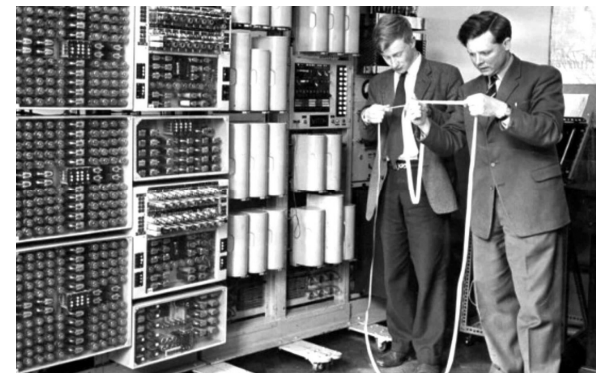
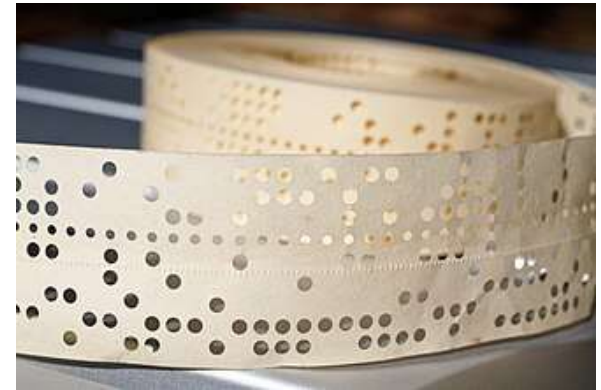
- A two-way infinite sequence of **cells**, that each holds a symbol or is empty;
- A pointer called **head**, that can move left or right on the tape.



Modeling the Memory: Tapes

A **tape** contains:

- A two-way infinite sequence of **cells**, that each holds a symbol or is empty;
- A pointer called **head**, that can move left or right on the tape.



Single-Tape Turing Machine

A **Turing machine (TM)** is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- $\Gamma \supseteq \Sigma \sqcup \{_ \}$ is another finite set of symbols (tape alphabet);

Single-Tape Turing Machine

A **Turing machine (TM)** is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

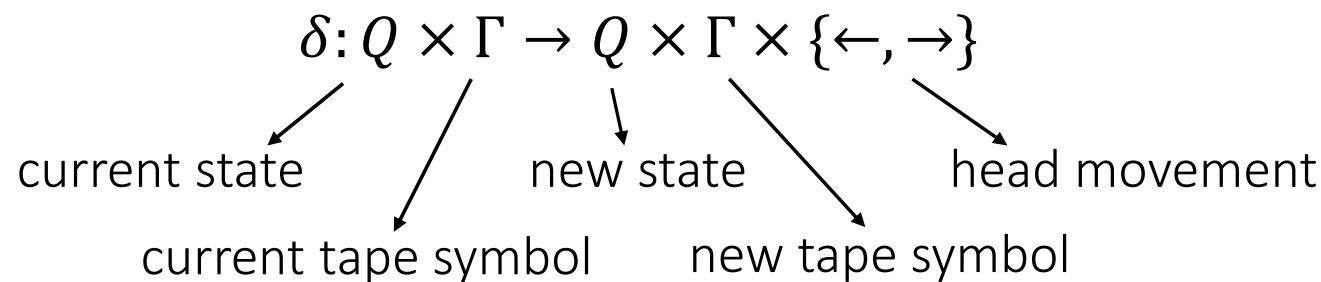
- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- $\Gamma \supseteq \Sigma \sqcup \{_ \}$ is another finite set of symbols (tape alphabet);
- δ is the transition function:

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

Single-Tape Turing Machine

A **Turing machine (TM)** is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

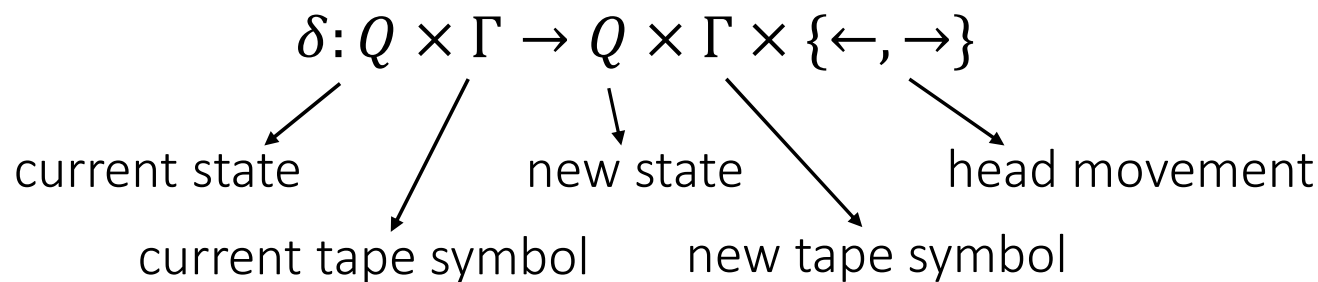
- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- $\Gamma \supseteq \Sigma \sqcup \{_ \}$ is another finite set of symbols (tape alphabet);
- δ is the transition function:



Single-Tape Turing Machine

A **Turing machine (TM)** is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- $\Gamma \supseteq \Sigma \sqcup \{_ \}$ is another finite set of symbols (tape alphabet);
- δ is the transition function:



- Initial state q_0 , accepting state q_{acc} , rejecting state $q_{rej} \neq q_{acc}$.

Single-Tape Turing Machine

On input $w = w_1 \cdots w_n \in \Sigma^*$:

- The tape is initialized to contain w in n consecutive cells, with the head pointing at w_1 ; the state is q_0 ;

Single-Tape Turing Machine

On input $w = w_1 \cdots w_n \in \Sigma^*$:

- The tape is initialized to contain w in n consecutive cells, with the head pointing at w_1 ; the state is q_0 ;
- At every step, if the state is q and the cell pointed by the head contains symbol $s \in \Gamma$ and $\delta(q, s) = (q', s', D)$:
 - The new state is q' ;
 - The cell pointed by the head rewrites the symbol to s' ;
 - The head moves one cell in direction D

Single-Tape Turing Machine

On input $w = w_1 \cdots w_n \in \Sigma^*$:

- The tape is initialized to contain w in n consecutive cells, with the head pointing at w_1 ; the state is q_0 ;
- At every step, if the state is q and the cell pointed by the head contains symbol $s \in \Gamma$ and $\delta(q, s) = (q', s', D)$:
 - The new state is q' ;
 - The cell pointed by the head rewrites the symbol to s' ;
 - The head moves one cell in direction D
- The computation **halts** if the state is q_{acc} (accept) or q_{rej} (reject).

Example: Palindrome

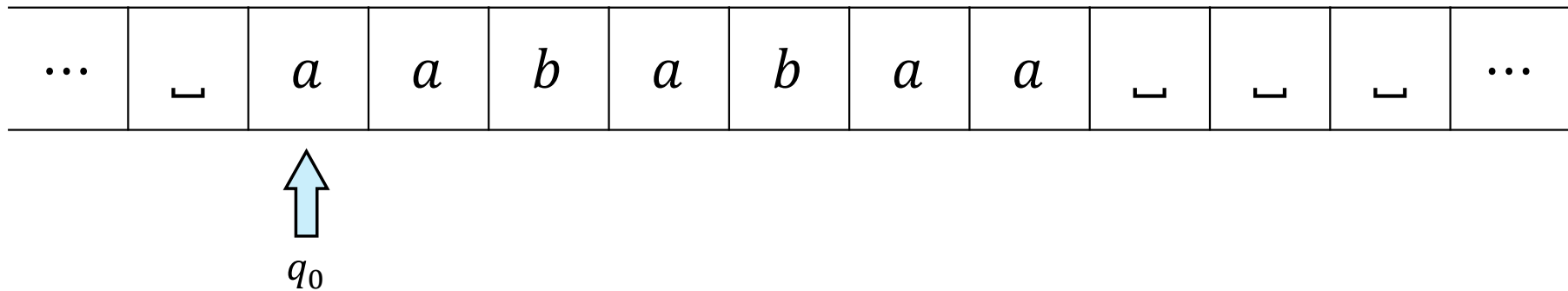
$$L = \{w = w^R \mid w \in \{a, b\}^*\}$$

q_0 : record the first symbol

$$\delta(q_0, a) = (q_a, \sqcup, \rightarrow)$$

$$\delta(q_0, b) = (q_b, \sqcup, \rightarrow)$$

$$\delta(q_0, \sqcup) = (q_{\text{acc}}, \sqcup, \rightarrow)$$



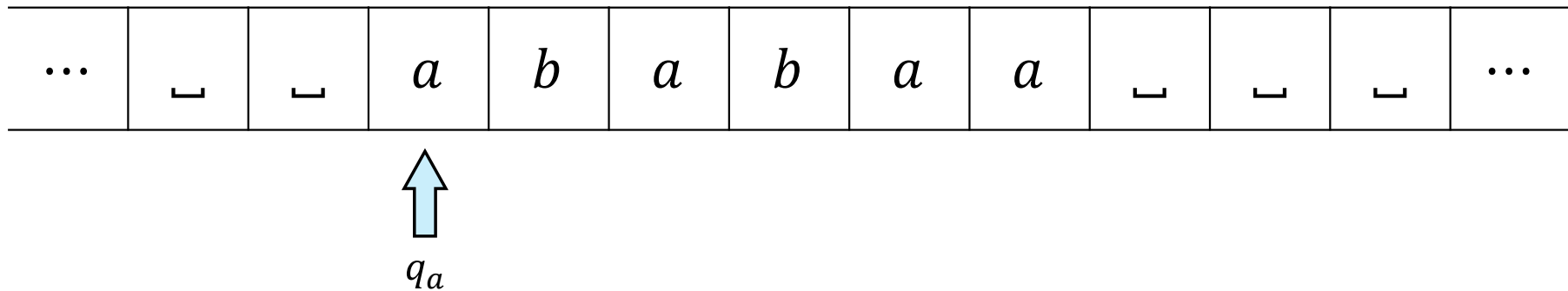
Example: Palindrome

q_a, q_b : find the last unmatched symbol

$$\delta(q_a, a) = (q_a, a, \rightarrow)$$

$$\delta(q_a, b) = (q_a, b, \rightarrow)$$

$$\delta(q_a, \sqcup) = (q'_a, \sqcup, \leftarrow)$$



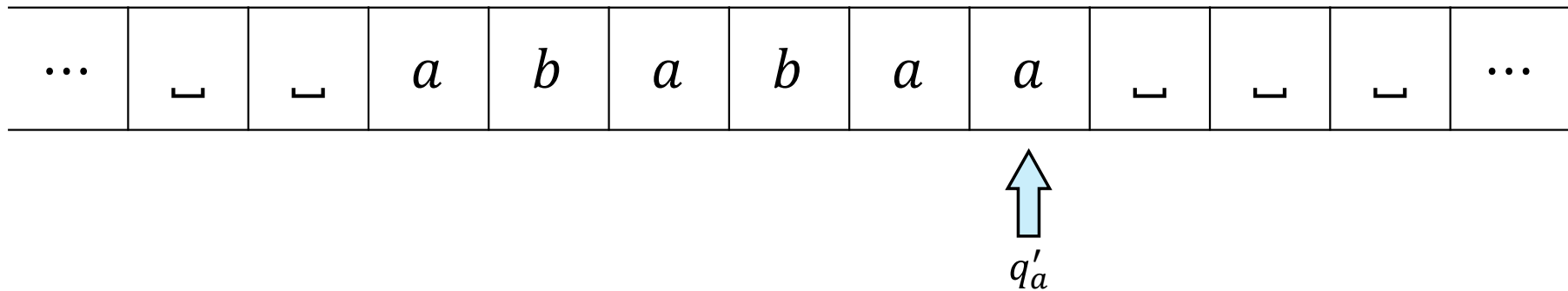
Example: Palindrome

q'_a, q'_b : check if the last symbol matches with the first symbol

$$\delta(q'_a, a) = (q_1, \sqcup, \leftarrow)$$

$$\delta(q'_a, b) = (q_{\text{rej}}, b, \leftarrow)$$

$$\delta(q'_a, \sqcup) = (q_{\text{acc}}, \sqcup, \leftarrow)$$



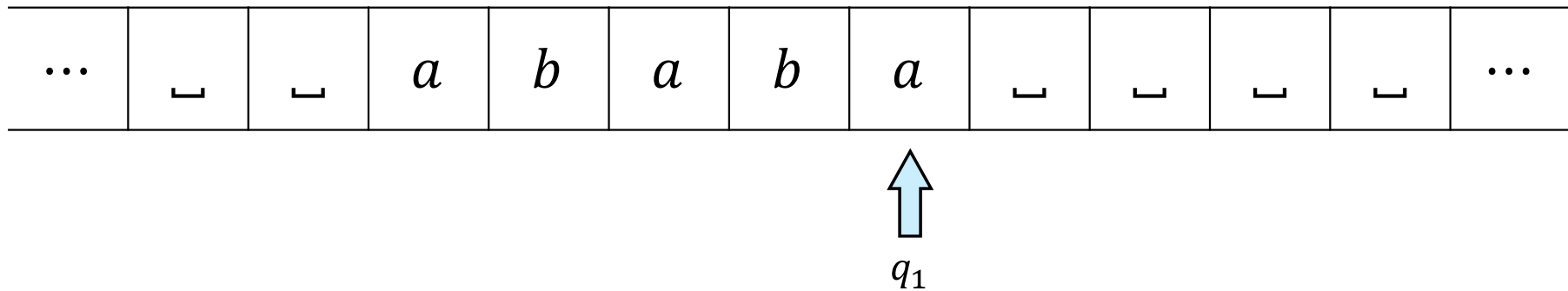
Example: Palindrome

q_1 : find the first remaining symbol

$$\delta(q_1, a) = (q_1, a, \leftarrow)$$

$$\delta(q_1, b) = (q_1, b, \leftarrow)$$

$$\delta(q_1, \sqcup) = (q_0, \sqcup, \rightarrow)$$



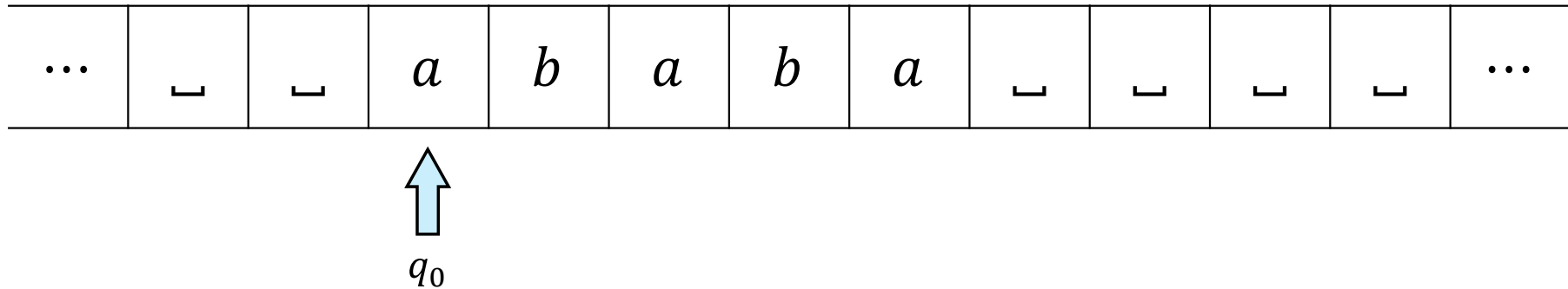
Example: Palindrome

q_1 : find the first remaining symbol

$$\delta(q_1, a) = (q_1, a, \leftarrow)$$

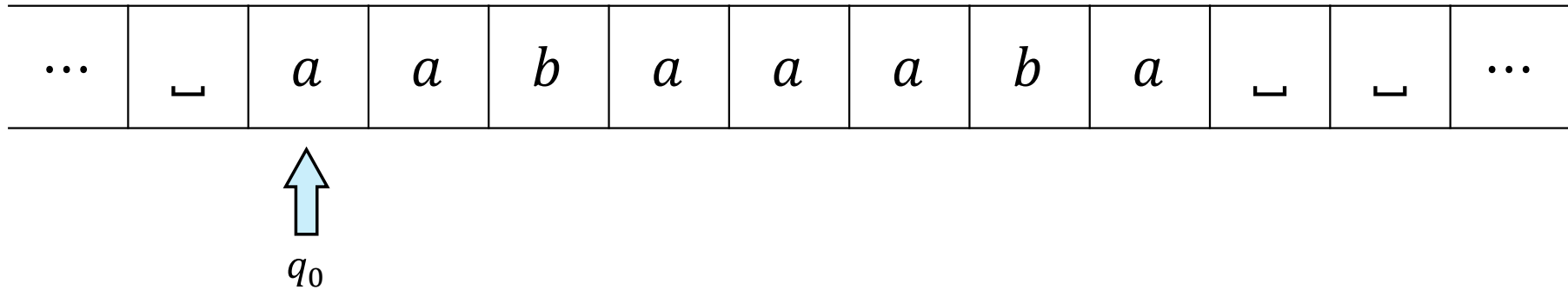
$$\delta(q_1, b) = (q_1, b, \leftarrow)$$

$$\delta(q_1, \sqcup) = (q_0, \sqcup, \rightarrow)$$



Example: Repeated Strings

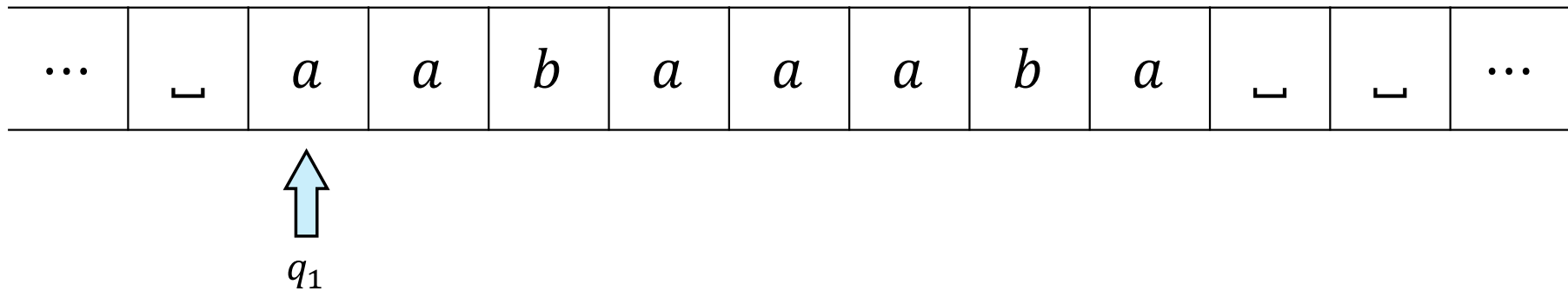
$$L = \{ww \mid w \in \{a, b\}^*\}$$



Example: Repeated Strings

$$L = \{ww \mid w \in \{a, b\}^*\}$$

Stage 1: Check the input has even length $2n$

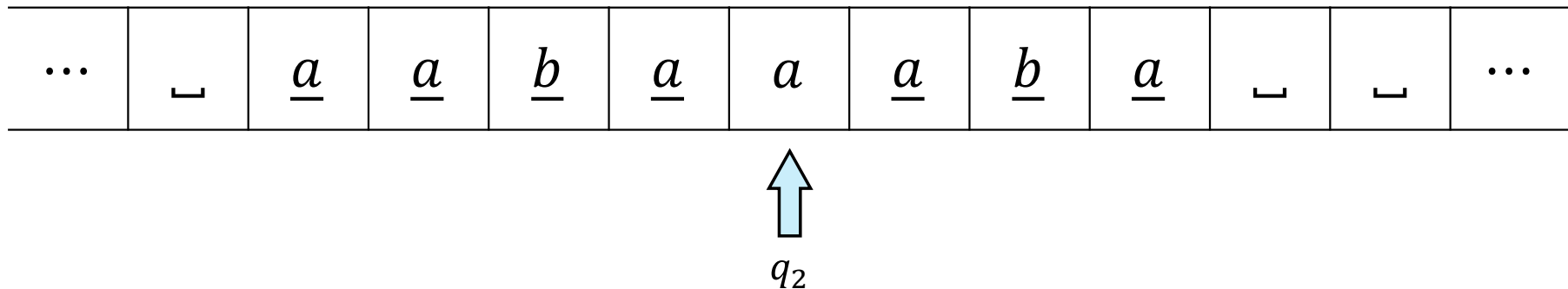


Example: Repeated Strings

$$L = \{ww \mid w \in \{a, b\}^*\}$$

Stage 1: Check the input has even length $2n$

Stage 2: Find the $(n + 1)$ -th symbol



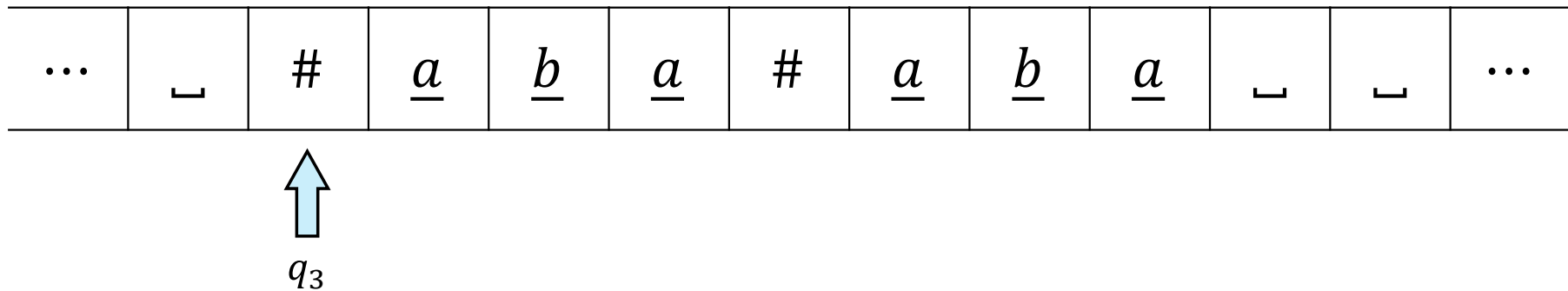
Example: Repeated Strings

$$L = \{ww \mid w \in \{a, b\}^*\}$$

Stage 1: Check the input has even length $2n$

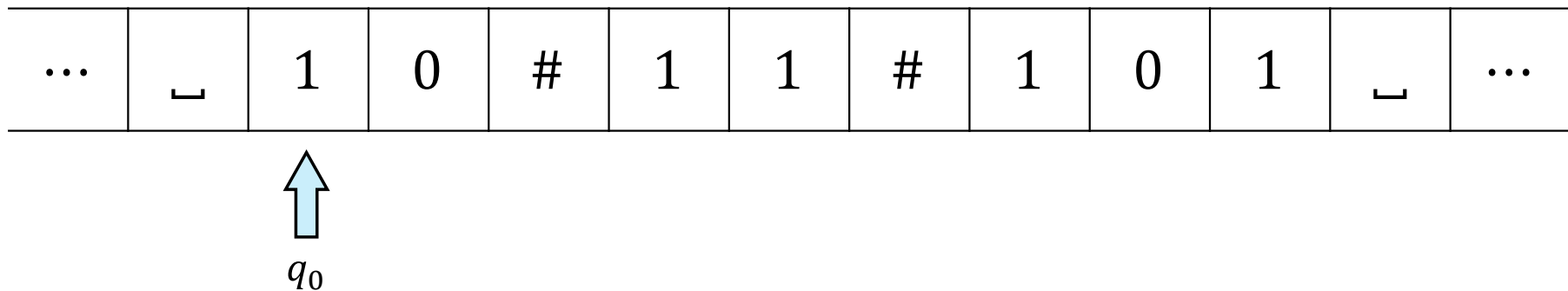
Stage 2: Find the $(n + 1)$ -th symbol

Repeat Stage 3: Match the symbols in the second half with the first half



Example: Binary Addition

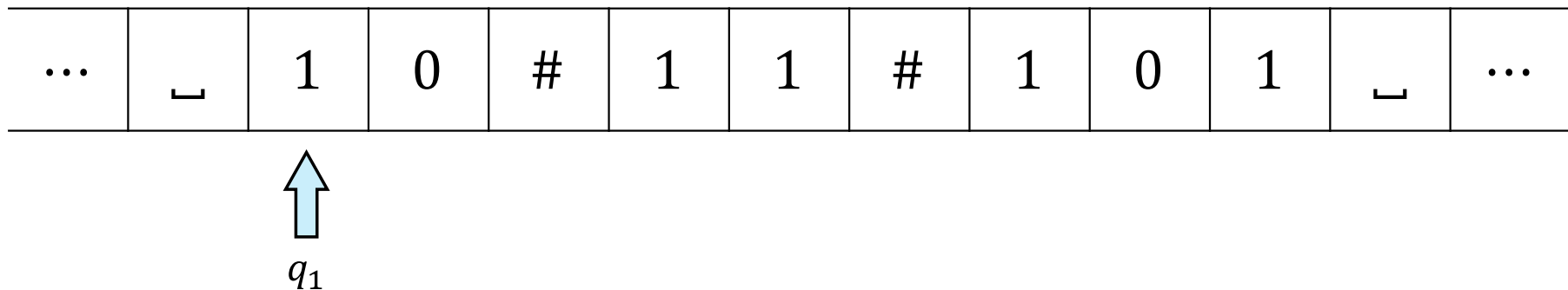
$$L = \{x\#y\#z \mid x, y, z \in \{0,1\}^*, x + y = z \text{ as binary numbers}\}$$



Example: Binary Addition

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

Stage 1: Check the input is of form $\{0,1\}^*\#\{0,1\}^*\#\{0,1\}^*$



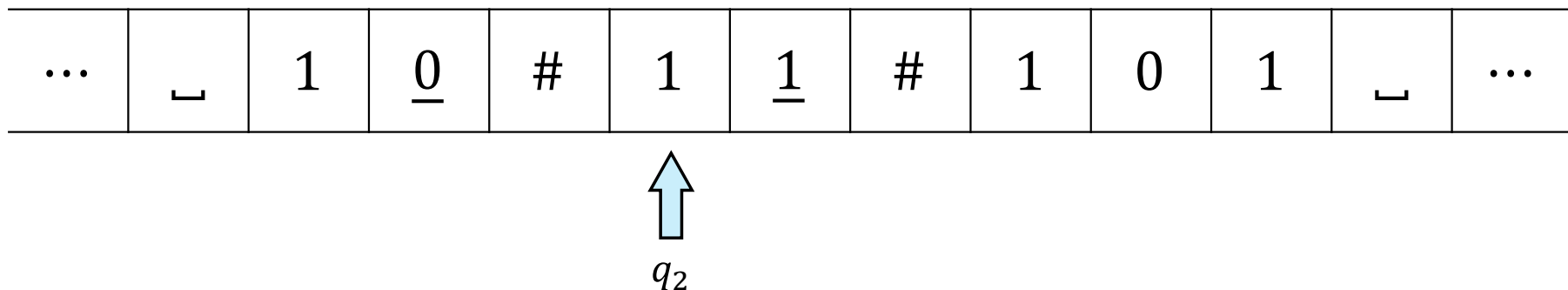
Example: Binary Addition

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

Stage 1: Check the input is of form $\{0,1\}^*\#\{0,1\}^*\#\{0,1\}^*$

Repeat Stage 2:

- Find the last 1 in x and the corresponding bit in y



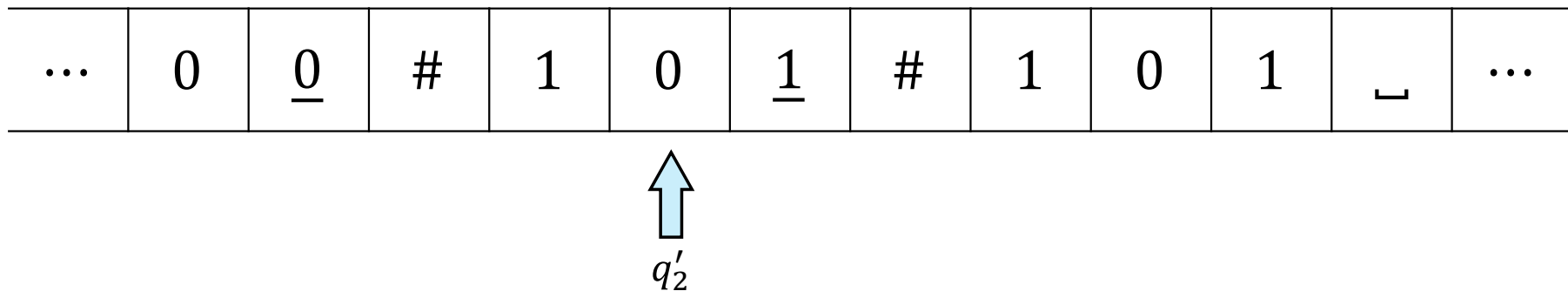
Example: Binary Addition

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

Stage 1: Check the input is of form $\{0,1\}^*\#\{0,1\}^*\#\{0,1\}^*$

Repeat Stage 2:

- Find the last 1 in x and the corresponding bit in y
- Set the 1 to 0 in x and add the 1 to y (set 011..1 to 100..0)



Example: Binary Addition

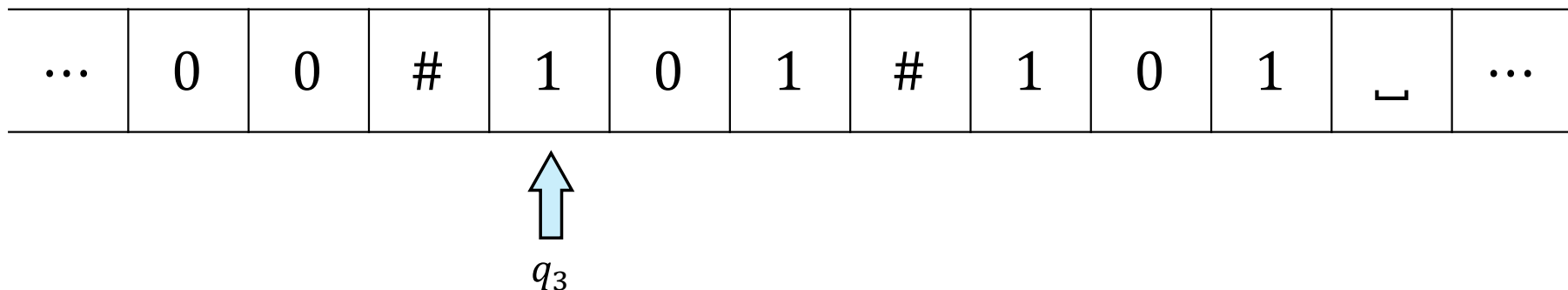
$$L = \{x\#y\#z \mid x, y, z \in \{0,1\}^*, x + y = z \text{ as binary numbers}\}$$

Stage 1: Check the input is of form $\{0,1\}^* \# \{0,1\}^* \# \{0,1\}^*$

Repeat Stage 2:

- Find the last 1 in x and the corresponding bit in y
- Set the 1 to 0 in x and add the 1 to y (set 011..1 to 100..0)

Stage 3: Check $y = z$



Single-Tape Turing Machine

The language **accepted/recognized** by a Turing machine M is

$$L(M) = \{w \in \Sigma^* \mid M \text{ reaches } q_{\text{acc}} \text{ on input } w\}$$

and we say the language is **recognizable**. In this case M **does not necessarily halt** on inputs $w \notin L(M)$.

Single-Tape Turing Machine

The language **accepted/recognized** by a Turing machine M is

$$L(M) = \{w \in \Sigma^* \mid M \text{ reaches } q_{\text{acc}} \text{ on input } w\}$$

and we say the language is **recognizable**. In this case M **does not necessarily halt** on inputs $w \notin L(M)$.

If M halts on every input in Σ^* , we say M **decides/computes** the language $L(M)$: M is a decider, and $L(M)$ is decidable/computable.

Single-Tape Turing Machine

We can define a partial function $M: \Sigma^* \rightarrow \{0,1\}$ for Turing machine M :

$$M(w) = \begin{cases} 1 & \text{if } M \text{ reaches } q_{\text{acc}} \text{ on input } w \\ 0 & \text{if } M \text{ reaches } q_{\text{rej}} \text{ on input } w \\ \perp \text{ (not defined)} & \text{if } M \text{ does not halt on input } w \end{cases}$$

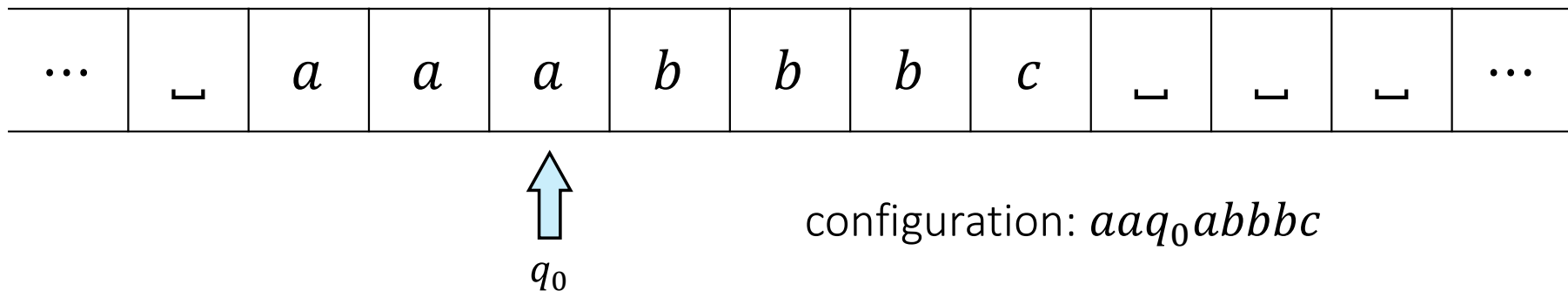
When M is a decider, $M(w)$ is a total function.

Configurations

A **configuration** of Turing machine M is a snapshot of M at a certain step.

We can express a configuration using a string $uqv \in (\Gamma \sqcup Q)^*$:

- uv is the current tape content (minus leading and following $\sqcup$)
- q is the current state
- the head points at the first symbol of v



Configurations

The execution of $M(w)$ can be expressed as a sequence of configurations:

$$C_0, C_1, \dots, C_t \in (\Gamma \sqcup Q)^*$$

where $C_0 = q_0w$ and C_i **yields** C_{i+1} :

Configurations

The execution of $M(w)$ can be expressed as a sequence of configurations:

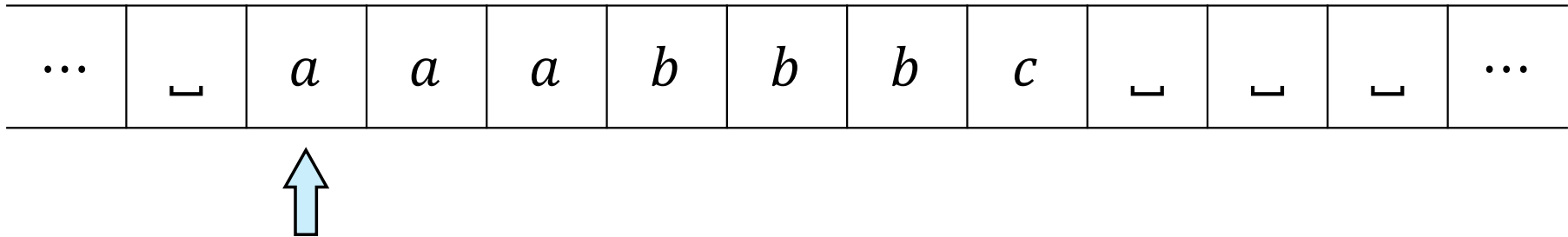
$$C_0, C_1, \dots, C_t \in (\Gamma \sqcup Q)^*$$

where $C_0 = q_0w$ and C_i **yields** C_{i+1} : if $C_i = \dots abqcd \dots$ then

- $C_{i+1} = \dots aq'bc'd \dots$, if $\delta(q, c) = (q', c', \leftarrow)$
- $C_{i+1} = \dots abc'q'd \dots$, if $\delta(q, c) = (q', c', \rightarrow)$

Closure Properties

Are recognizable languages closed under intersection?

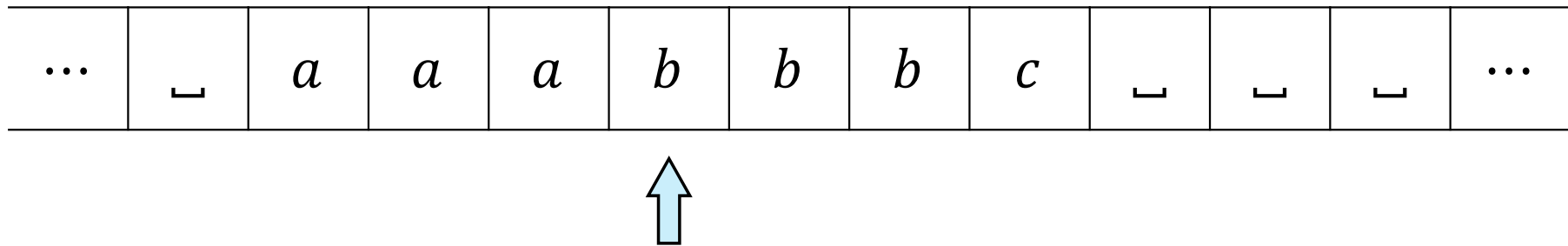


Closure Properties

Are recognizable languages closed under intersection?

Idea: $w \in L = L(M_1) \cap L(M_2)$ if and only if both M_1 and M_2 accepts x .

Can we first run $M_1(w)$ and then $M_2(w)$?

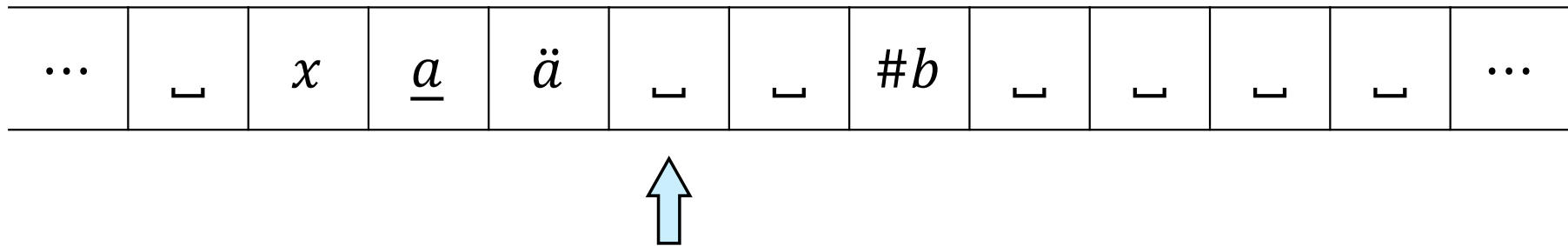


Closure Properties

Are recognizable languages closed under intersection?

Idea: $w \in L = L(M_1) \cap L(M_2)$ if and only if both M_1 and M_2 accepts x .

Can we first run $M_1(w)$ and then $M_2(w)$?



Closure Properties

Are recognizable languages closed under intersection?

Idea: $w \in L = L(M_1) \cap L(M_2)$ if and only if both M_1 and M_2 accepts x .

Can we first run $M_1(w)$ and then $M_2(w)$?

We can save a copy of the input w by using $\Gamma' = \Gamma \times \Gamma$.

...	␣	x	<u>a</u>	$\ddot{a}$	␣	␣	$\#b$	␣	␣	␣	␣	...
...	␣	a	a	a	b	b	b	c	␣	␣	␣	...

