CMSC 28100

Introduction to Complexity Theory

Autumn 2025
Instructor: William Hoza

The nature of this course

- We will study:
  - The mathematical/philosophical foundations of computer science
  - The ultimate limits of computation
- We will develop conceptual tools for reasoning about computation
- Expect lots of proofs and very little programming

Who this course is designed for

- CS students, math students, and anyone who is curious
- Prerequisites:
  - Experience with mathematical proofs
  - CMSC 27200 or CMSC 27230 or CMSC 37000, or MATH 15900 or MATH 15910 or MATH 16300 or MATH 16310 or MATH 19900 or MATH 25500

Who this course is designed for

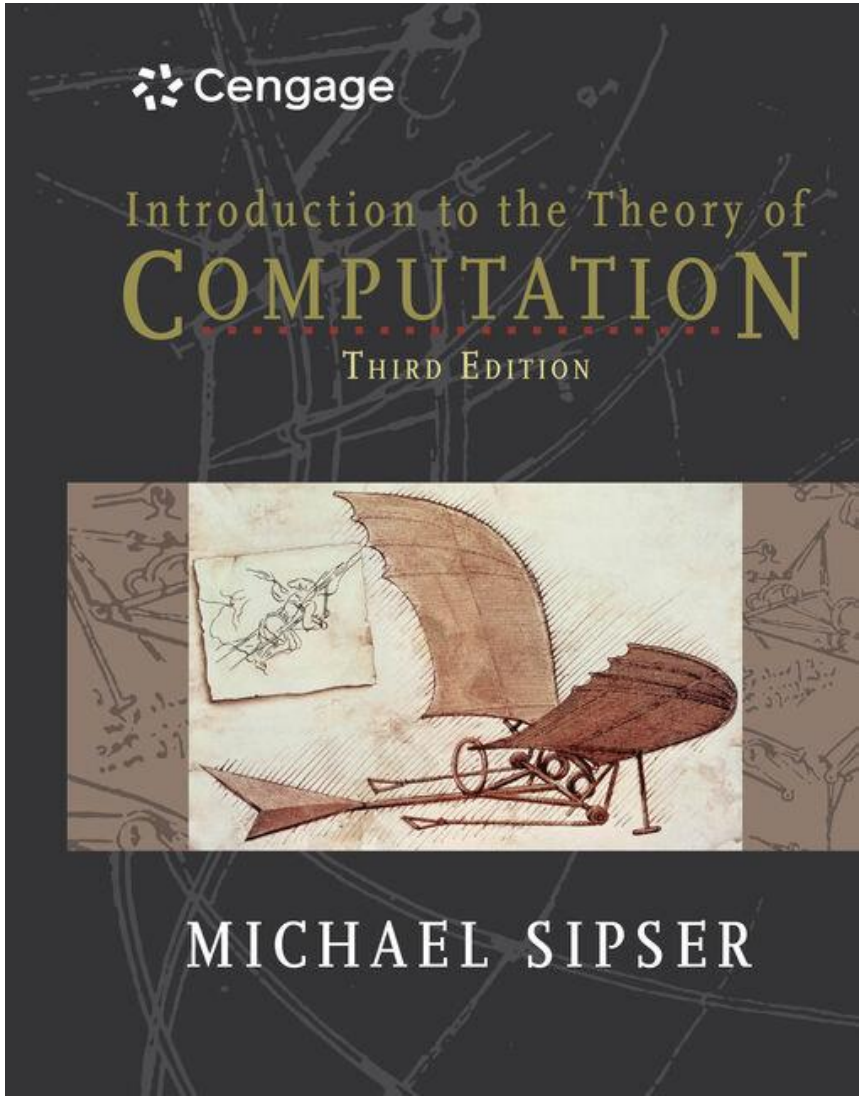
- It’s okay if you don’t consider yourself “theory-oriented”
- You belong here
- My job:
  - Give you resources so you can learn and succeed
  - Persuade you that complexity theory is worthy of your attention

Class participation

- Please ask questions!
  - “What does that notation mean?”
  - “I forget what a _____ is. Can you remind me?”
  - “How do we know _____?”
  - “I’m lost. Can you explain that again?”

Textbook

- Classic
- Popular
- High-quality
- Not free 😟



Assessment

- 29 homework exercises
  - Exercises 1-3 are due **Friday, October 3**
- Midterm exam in class on Friday, October 24
- Final exam at the end of the quarter

My office hours

- Fridays, 9am to 11am, JCL 205
- Confused/curious? 👉 I'll try to help you learn
- Stuck on the homework? 👉 I'll try to think of a good hint
- Have a complaint? 👉 I'll listen and try to make things better

Teaching assistants

- Mirza Mehmedagic
    - Office hours: Thursdays, 11am to noon, JCL 205
- Zelin Lv
    - Office hours: Thursdays, 3pm to 4pm, JCL 205

Student meet-up time

- Thursdays, 2pm to 3pm, JCL Common Area A
  - Immediately before Zelin's office hours
- Find study partners
- Discuss course topics
- Collaborate on homework

https://canvas.uchicago.edu/courses/66278

- Discussions
- Announcements

- Course policies
- Slides

- Submitting your homework solutions
- Grades
- Feedback on your work

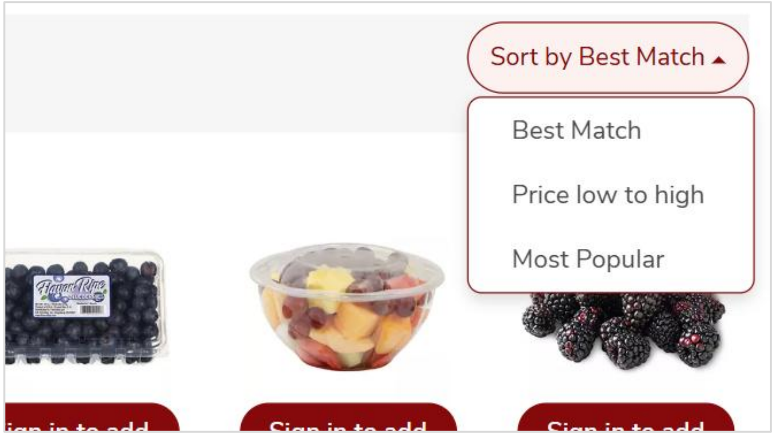
- Exercises
- Practice exams
- Official solutions

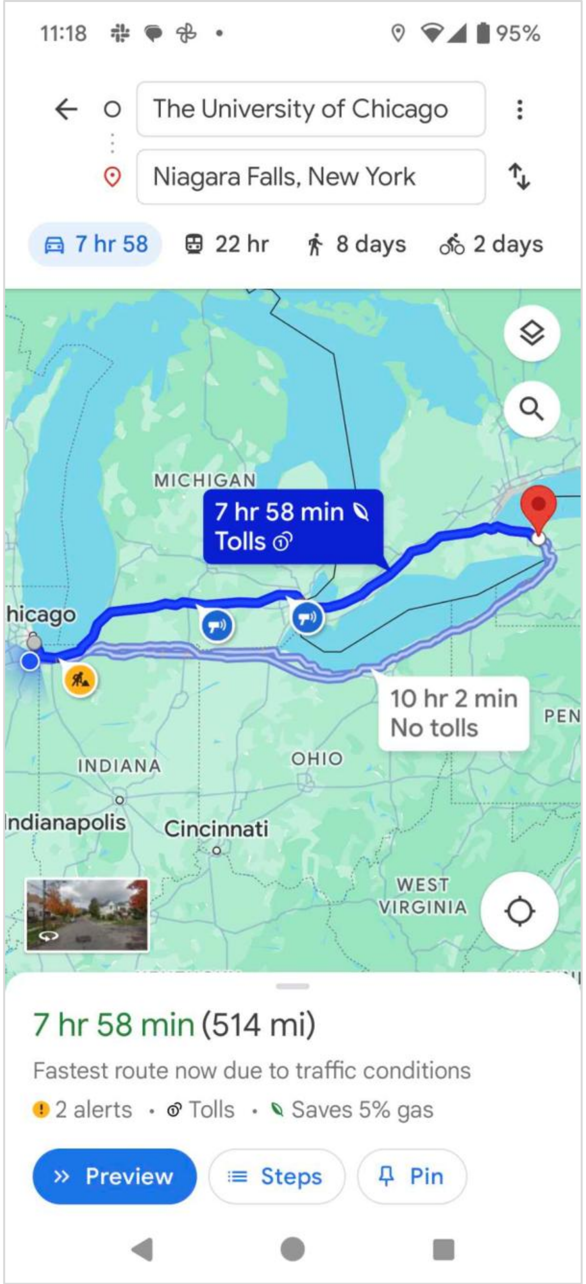
The central question of this course:

Which problems can be solved through computation?

Examples

- Many problems can be solved through computation:
    - Multiplication
    - Sorting
    - Shortest path





- Are there any problems that cannot be solved through computation?

Impossibility proofs

- We will take a mathematical approach to this question

1. Formulate precise mathematical models
    - “Computation”
    - “Problem”
    - “Solve”
2. Write rigorous mathematical proofs of impossibility

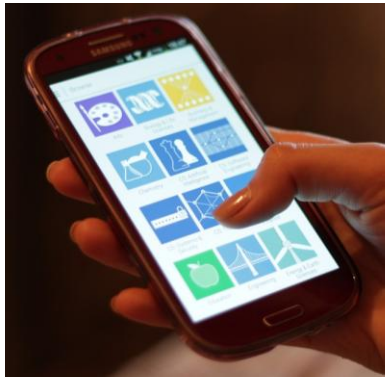
Which problems

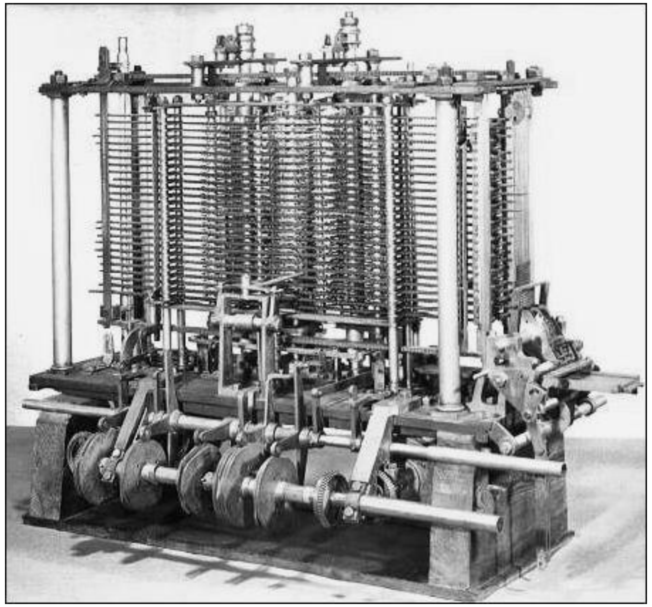
can be solved

through computation?

Computation

- Computers: Modern technology?
- Computation is ancient
- Can be performed by:
  - A human being with paper and a pencil
  - A smartphone
  - A steam-powered machine

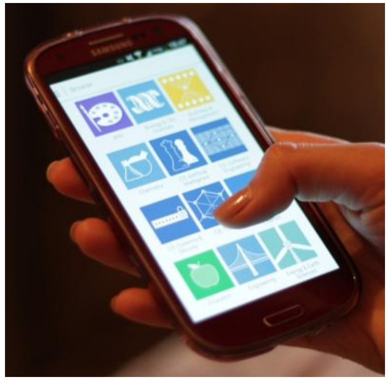




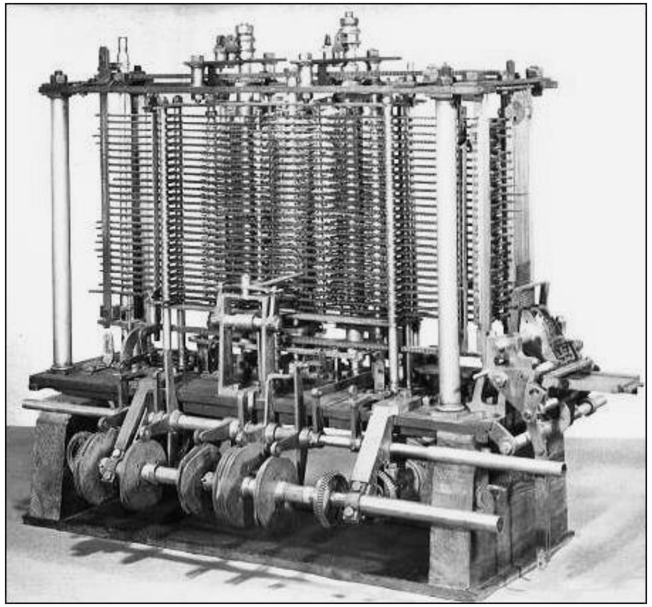
- We want a mathematical model that describes all of these and transcends any one technology

Computation

- Note: Humans can do all the same computations that smartphones/laptops do
  - (less quickly/reliably)

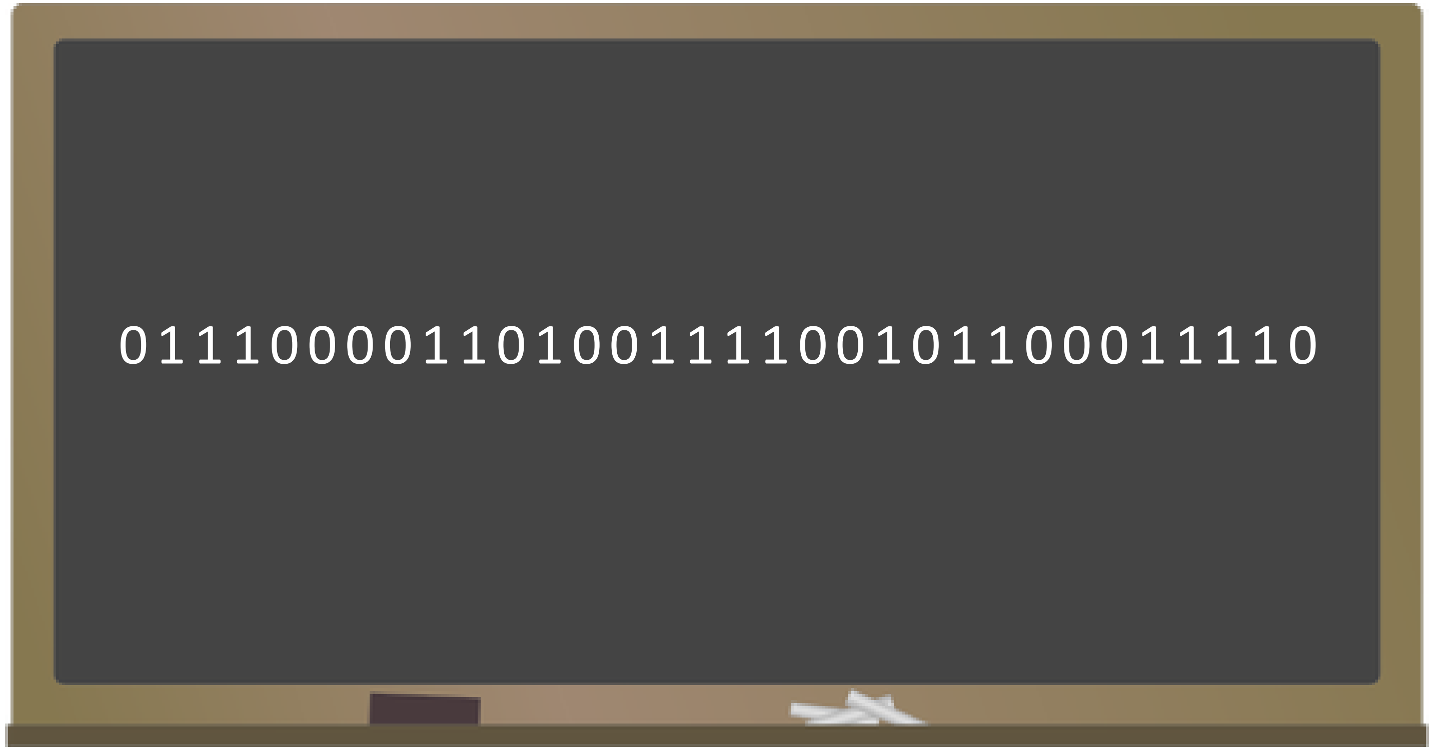


- Consequence: We can study computation without understanding electronics 🥹



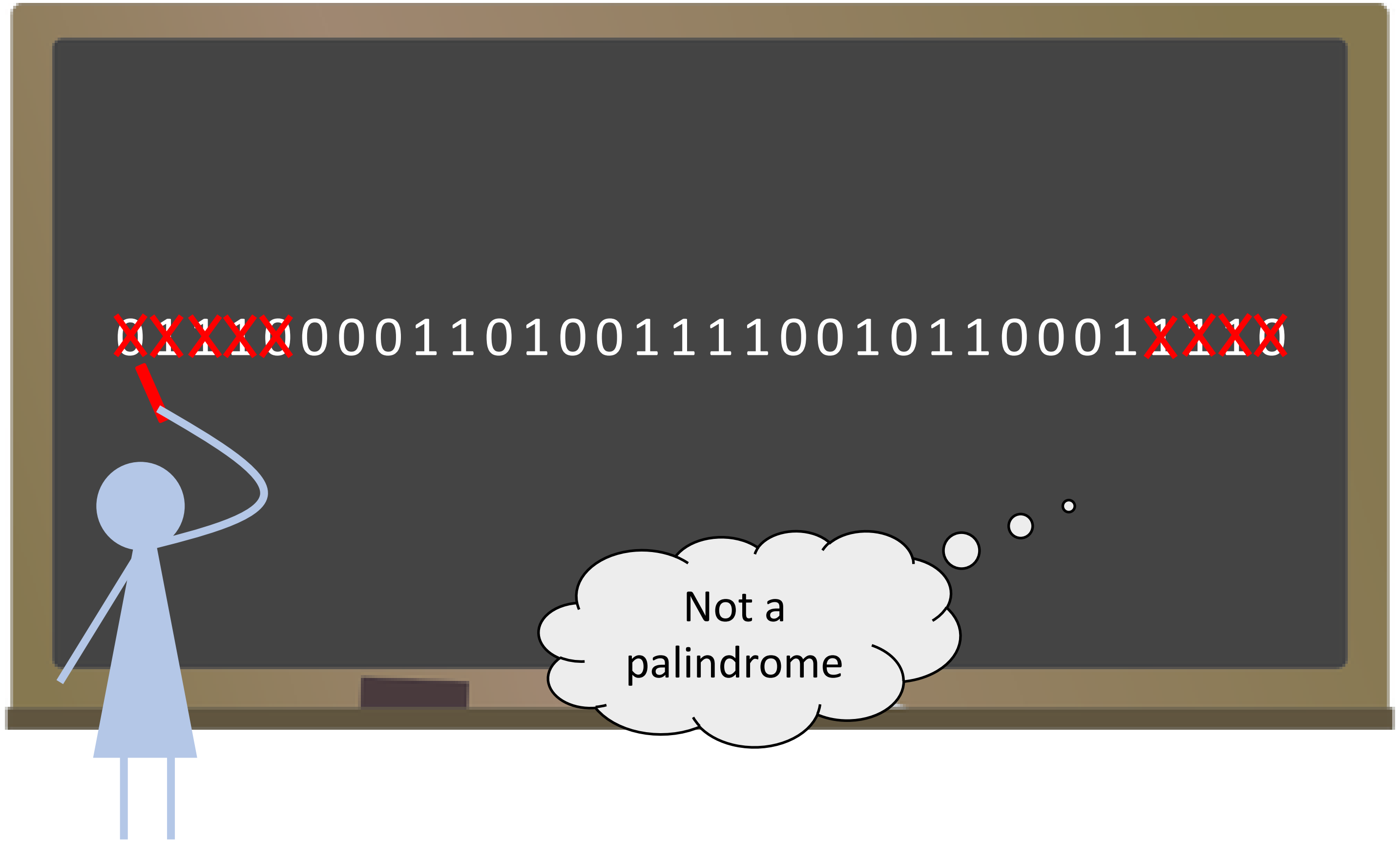
- Computation is a familiar, everyday, human act

Ex: Palindromes



- Suppose a long string of bits is written on a blackboard
- Our job: Figure out whether the string is a "palindrome," i.e., whether it is the same forwards and backwards
- What should we do?

Ex: Palindromes



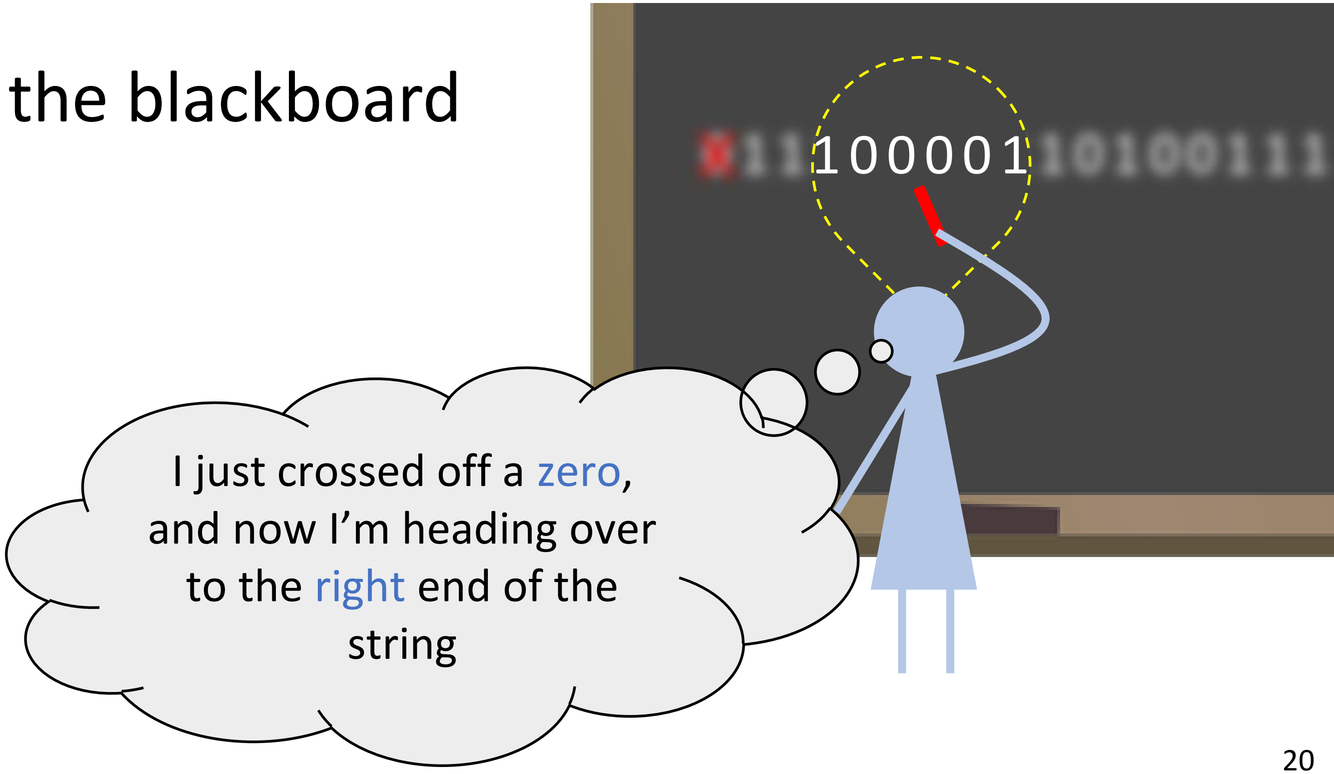
- Idea: Compare and cross off the first and last symbols
- Repeat until we find a mismatch or everything is crossed off

Local decisions

- In each step, how do we know what to do next?

1. We keep track of some information ("state") in our mind
2. We look at the local contents of the blackboard (one symbol is sufficient)

- We can describe the algorithm using "transition function" (next slide)



and we see this symbol…

If we're in this state…

	0	**1**	**~~0~~**	**~~1~~**	**(blank)**
q_0					Move right Go to state q_1
q_1	Cross off 0 Move right Go to state q_2	Cross off 1 Move right Go to state q_3	Output YES	Output YES	Output YES
q_2	Move right	Move right	Move left Go to state q_4	Move left Go to state q_4	Move left Go to state q_4
q_3	Move right	Move right	Move left Go to state q_5	Move left Go to state q_5	Move left Go to state q_5
q_4	Cross off 0 Move left Go to state q_6	Output NO	Output YES	Output YES	
q_5	Output NO	Cross off 1 Move left Go to state q_6	Output YES	Output YES	
q_6	Move left	Move left	Move right Go to state q_1	Move right Go to state q_1	

and we see this symbol…

If we're in this state…

then we should do this

	0	1	~~0~~	~~1~~	(blank)
q_0					Move right Go to state q_1
q_1	Cross off 0 Move right Go to state q_2	Cross off 1 Move right Go to state q_3	Output YES	Output YES	Output YES
q_2	Move right	Move right	Move left Go to state q_4	Move left Go to state q_4	Move left Go to state q_4
q_3	Move right	Move right	Move left Go to state q_5	Move left Go to state q_5	Move left Go to state q_5
q_4	Cross off 0 Move left Go to state q_6	Output NO	Output YES	Output YES	
q_5	Output NO	Cross off 1 Move left Go to state q_6	Output YES	Output YES	
q_6	Move left	Move left	Move right Go to state q_1	Move right Go to state q_1	

If we're in this state…

and we see this symbol…

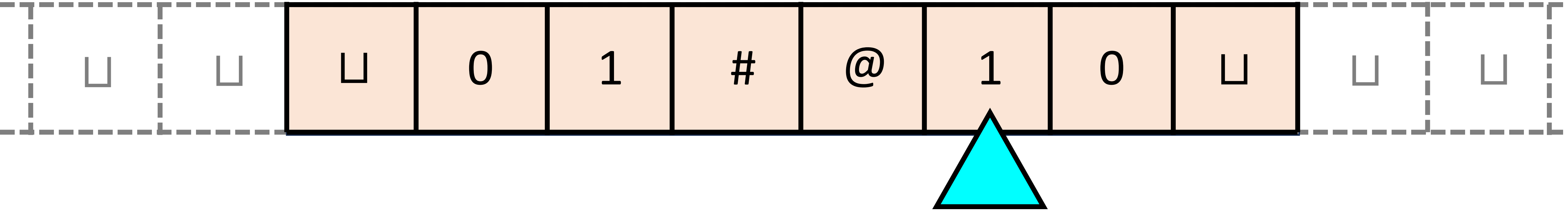
then we should do this

	0	1	~~0~~	~~1~~	(blank)
q_0					Move right Go to state q_1
q_1	Cross off 0 Move right Go to state q_2	**Cross off 1 Move right Go to state q_3**	Output YES	Output YES	Output YES
q_2	Move right	Move right	Move left Go to state q_4	Move left Go to state q_4	Move left Go to state q_4
q_3	Move right	Move right	Move left Go to state q_5	Move left Go to state q_5	Move left Go to state q_5
q_4	Cross off 0 Move left Go to state q_6	Output NO	Output YES	Output YES	
q_5	Output NO	Cross off 1 Move left Go to state q_6	Output YES	Output YES	
q_6	Move left	Move left	Move right Go to state q_1	Move right Go to state q_1	

The Turing machine model

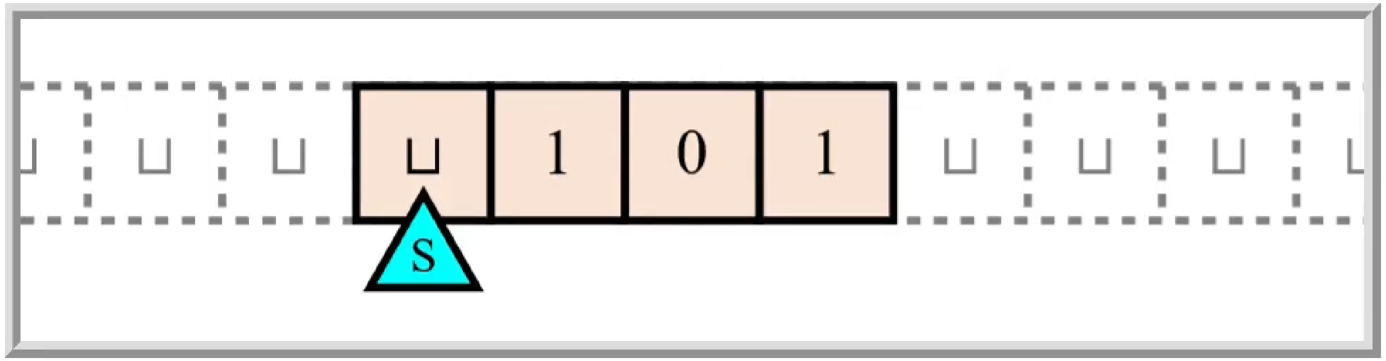
- Turing machines: A mathematical model of human computation
- In a nutshell, a Turing machine is any algorithm that can be described by a transition function like the one we just saw

The Turing machine model



- There is a "tape" that is divided into "cells"
- Each cell has one symbol written in it
- There is a "head" pointing at one cell
- The machine can be in one of finitely many internal "states"

Turing machines



- In each step, the machine decides
  - What to write
  - Which direction to move the head (left or right)
  - The new state
- Decision is based only on current state and observed symbol
- New cells are automatically "created" when needed

Transition function

- Mathematically, a Turing machine is described by a transition function

$$\delta: Q \times \Sigma \to Q \times \Sigma \times \{\mathrm{L}, \mathrm{R}\}$$

- Here Q is the set of states and Σ is the set of symbols
- $\delta(q, b) = (q', b', D)$ means:
  - If we are in the state q and we read the symbol b…
  - Write b' (replacing b), move the head in direction D ($\mathrm{L} =$ left, $\mathrm{R} =$ right), and go to state q'

Input

- One Turing machine represents one algorithm
- For us, the input to a Turing machine will always be a finite string over the binary alphabet

Symbols and alphabets

- An “alphabet” Σ is any nonempty, finite set of “symbols”
  - $\Sigma = \{0, 1\}$
  - $\Sigma = \{\sqcup, 0, 1, \cancel{0}, \cancel{1}\}$
  - $\Sigma = \{A, B, C, \ldots, Z\}$
  - $\Sigma = \{$ 😍, ⚾, 🐍, ⏰, 🍕 $\}$

Strings

- Let Σ be an alphabet
- A string over Σ is a finite se
- The length of a string x is the number of symbols, denoted $|x|$
- If n is a nonnegative integer, then Σ^n is the set of length-n strings over Σ
- Example: If $\Sigma = \{0, 1\}$, then

$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

If $|\Sigma| = m$, then what is $|\Sigma^0|$?

A: $|\Sigma^0| = 0$

B: $|\Sigma^0| = m$

C: $|\Sigma^0| = 1$

D: $|\Sigma^0|$ is not well-defined

Respond at PollEv.com/whoza or text "whoza" to 22333

The empty string

- If Σ is any alphabet, then $|\Sigma^0| = 1$
- There is one string of length zero: the empty string, denoted ϵ
  - Denoted "" in popular programming languages
- $\Sigma^0 = \{\epsilon\}$

Arbitrary-length strings

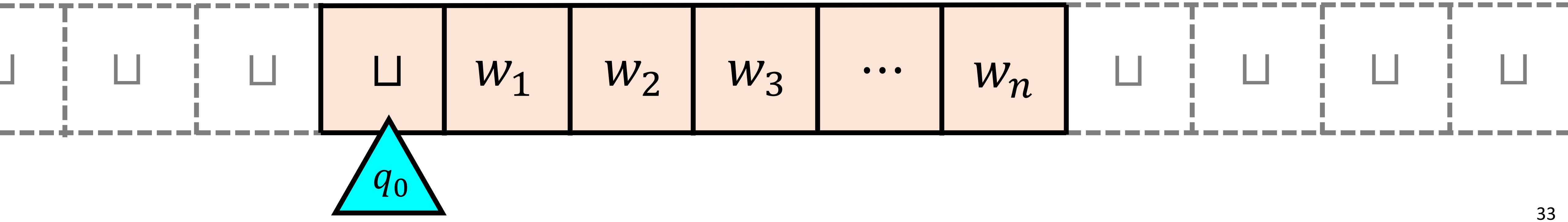
- Let Σ be an alphabet
- We define Σ^* to be the set of strings over Σ of any finite length:

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

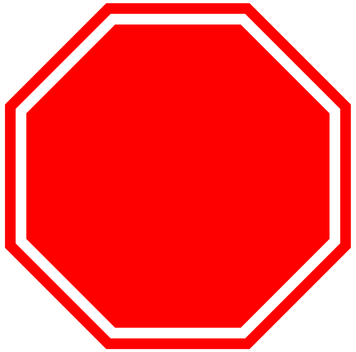
- Example: $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\}$

Turing machine initialization

- The tape initially contains an input string $w \in \{0,1\}^*$ (one bit per cell)
- The head is initially in a cell to the left of the input
  - This cell contains a special "blank symbol" $\sqcup$
- The machine is initially in a special "start state" q_0

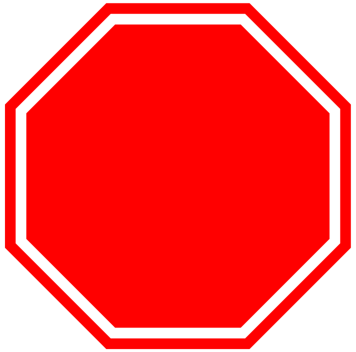


Halting



- After initialization, we repeatedly apply the transition function
- Whenever necessary, a new cell is created containing $\sqcup$
- Eventually, the machine might reach a "halting state"
  - There are two halting states: q_{accept} and q_{reject}
- In this case, the computation halts

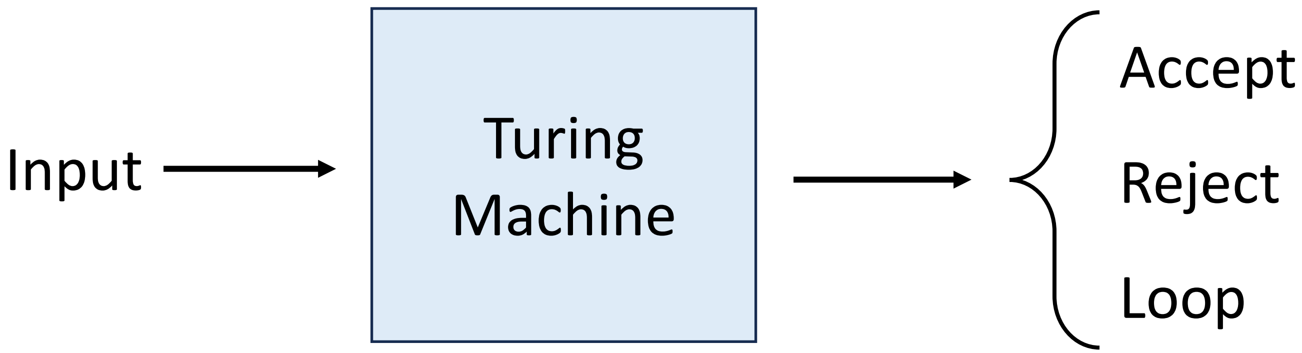
Accepting and rejecting



- If the machine reaches q_{accept}, we say it accepts the input
  - Analogy: "`return True`"
- If the machine reaches q_{reject}, we say it rejects the input
  - Analogy: "`return False`"

Looping

- It is also possible that the machine runs forever without halting
- In this case, we say the machine “loops”



Which problems can be solved through computation?

Languages

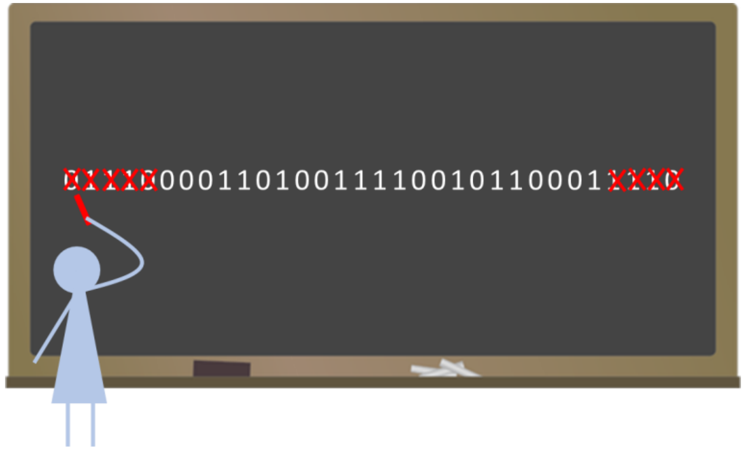
- A binary language is a set $Y \subseteq \{0, 1\}^*$
- Each language $Y \subseteq \{0, 1\}^*$ represents a distinct computational problem:

  "Given $w \in \{0, 1\}^*$, figure out whether $w \in Y$"

Deciding a language

- Let M be a Turing machine and let $Y \subseteq \{0,1\}^*$
- We say that M decides Y if
  - M accepts every $w \in Y$, and
  - M rejects every $w \in \{0,1\}^* \setminus Y$
- This is a mathematical model of what it means to "solve a problem"

Example: Palindromes



- **Informal problem statement:** "Given $w \in \{0,1\}^*$, determine whether w is the same forward and backward."
- **The same problem, formulated as a language:**

$$\mathrm{PALINDROMES} = \{w \in \{0,1\}^* : w \text{ is the same forward and backward}\}$$

- There exists a Turing machine that decides $\mathrm{PALINDROMES}$

Example: Parity

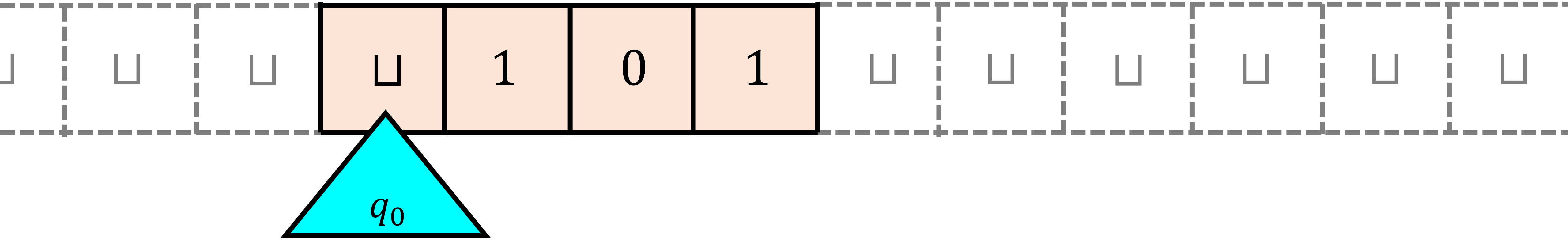
- **Informal problem statement:** "Given $w \in \{0,1\}^*$, determine whether the number of ones in w is even or odd."
- **The same problem, formulated as a language:**

$$\mathrm{PARITY} = \{w \in \{0,1\}^* : w \text{ has an odd number of ones}\}$$

- Exercise: Design a Turing machine that decides PARITY

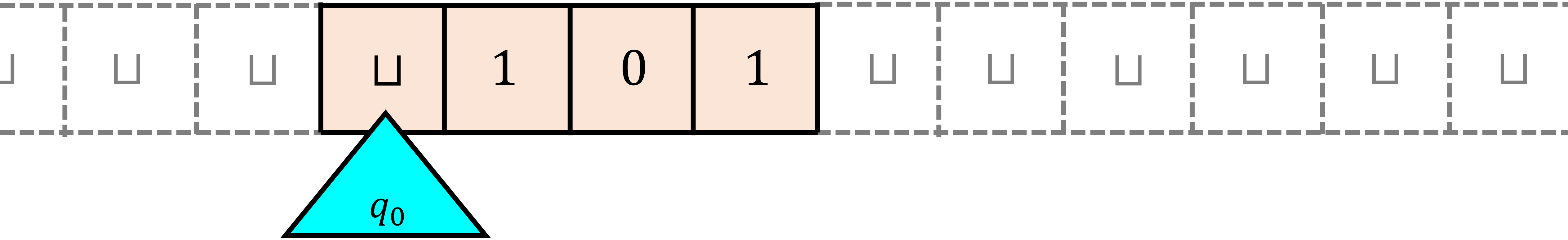
Example: A Turing machine that decides PARITY

	0	1	$\sqcup$
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



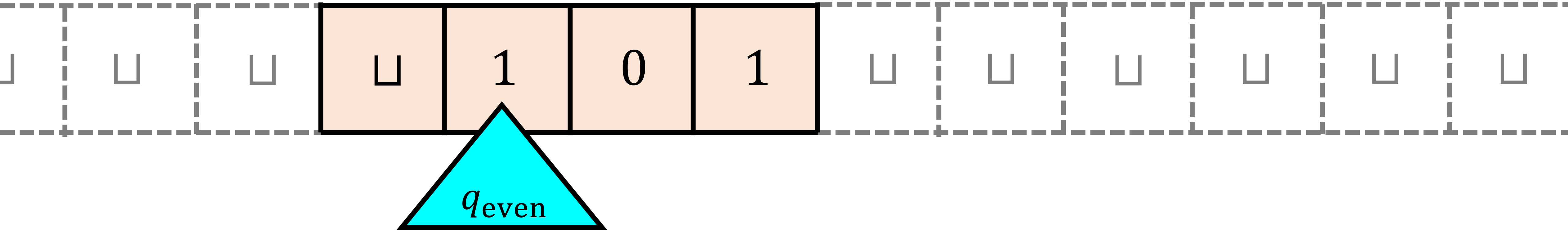
Example: A Turing machine that decides PARITY

	0	1	⊔
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



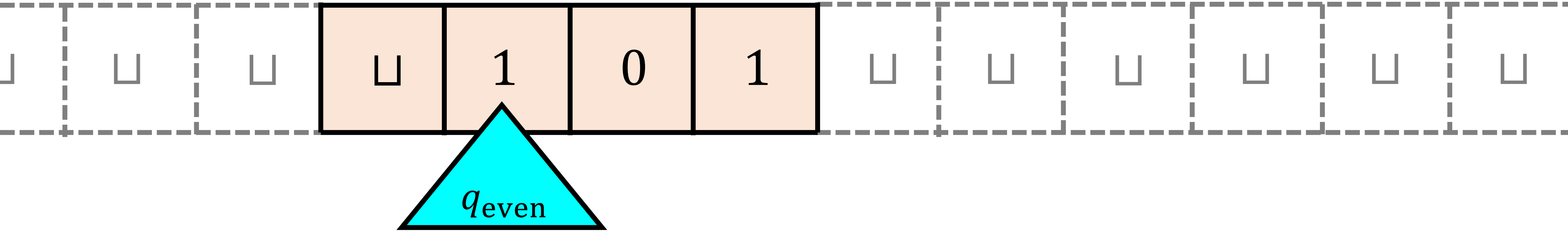
Example: A Turing machine that decides PARITY

	0	1	$\sqcup$
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



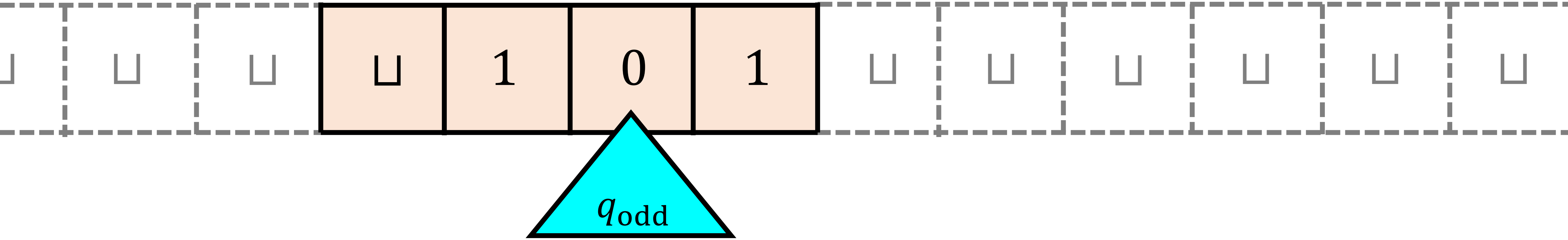
Example: A Turing machine that decides PARITY

	0	1	⊔
q_0			$(q_{\text{even}}, ⊔, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, ⊔, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, ⊔, \text{R})$
q_{accept}			
q_{reject}			



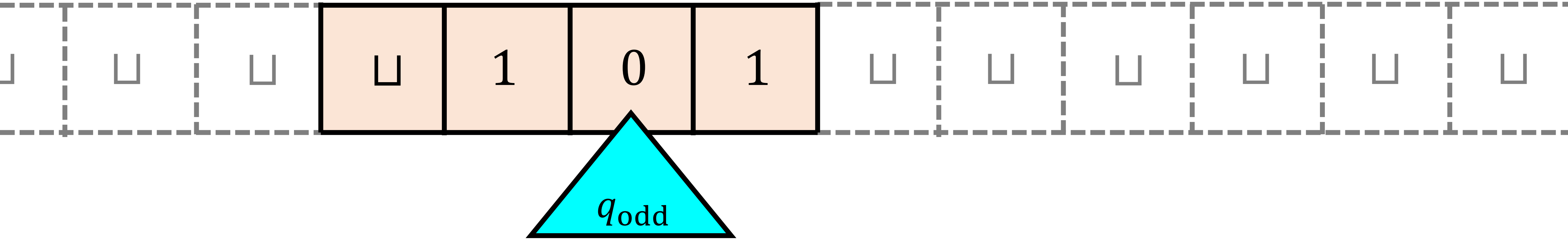
Example: A Turing machine that decides $\mathbf{PARITY}$

	0	1	⊔
q_0			$(q_{\text{even}}, ⊔, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, ⊔, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, ⊔, \text{R})$
q_{accept}			
q_{reject}			



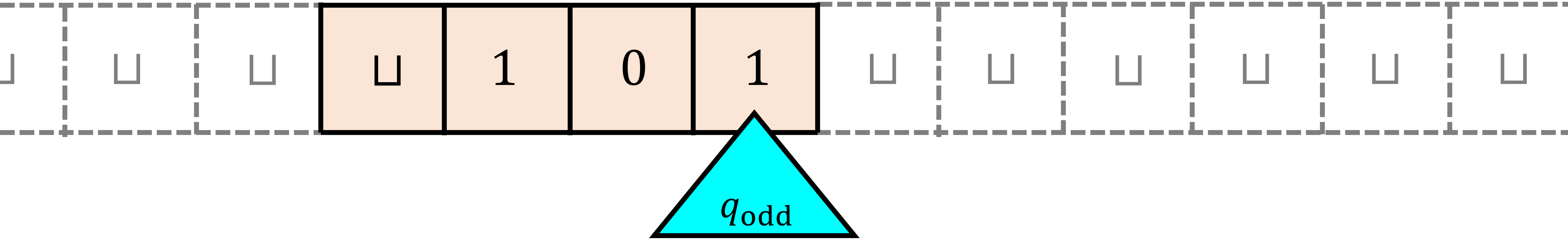
Example: A Turing machine that decides PARITY

	0	1	⊔
q_0			$(q_{\text{even}}, ⊔, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, ⊔, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, ⊔, \text{R})$
q_{accept}			
q_{reject}			



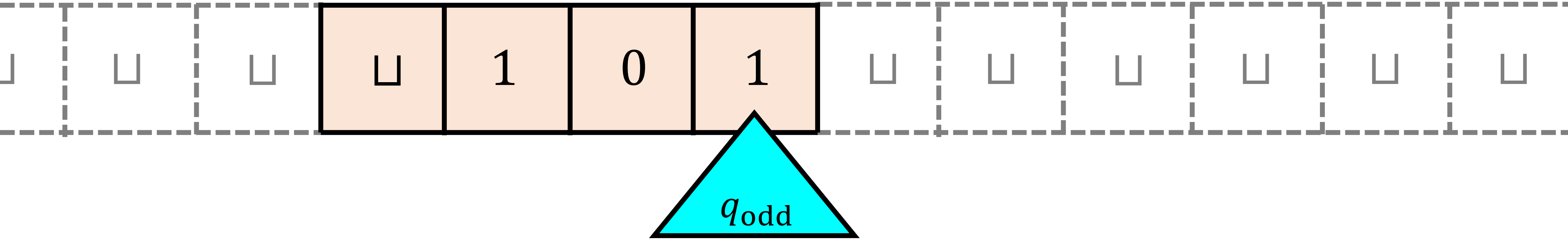
Example: A Turing machine that decides PARITY

	$\mathbf{0}$	$\mathbf{1}$	$\sqcup$
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



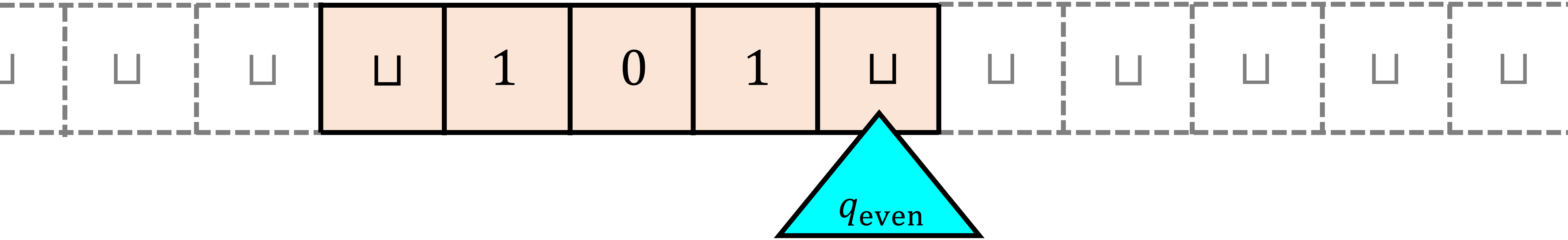
Example: A Turing machine that decides PARITY

	0	1	$\sqcup$
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



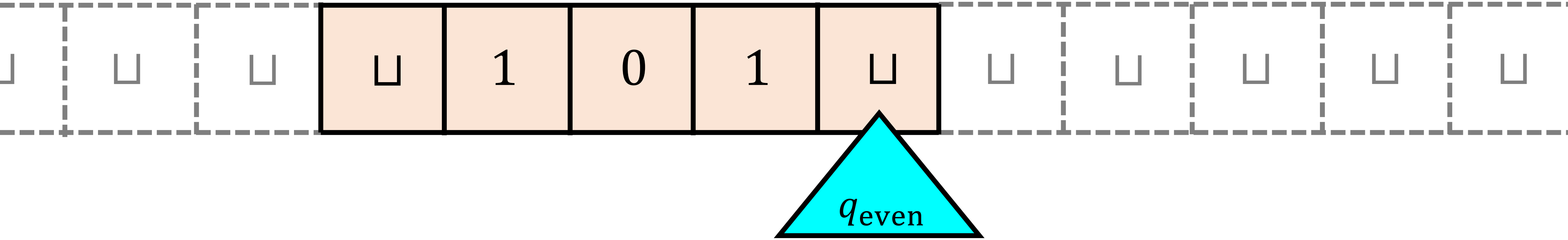
Example: A Turing machine that decides PARITY

	0	1	$\sqcup$
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



Example: A Turing machine that decides PARITY

	0	1	⊔
q_0			$(q_{\text{even}}, \sqcup, \text{R})$
q_{even}	$(q_{\text{even}}, 0, \text{R})$	$(q_{\text{odd}}, 1, \text{R})$	$(q_{\text{reject}}, \sqcup, \text{R})$
q_{odd}	$(q_{\text{odd}}, 0, \text{R})$	$(q_{\text{even}}, 1, \text{R})$	$(q_{\text{accept}}, \sqcup, \text{R})$
q_{accept}			
q_{reject}			



Example: A Turing machine that decides PARITY

	0	1	$\sqcup$
q_0			$(q_{\mathrm{even}}, \sqcup, \mathrm{R})$
q_{even}	$(q_{\mathrm{even}}, 0, \mathrm{R})$	$(q_{\mathrm{odd}}, 1, \mathrm{R})$	$(q_{\mathrm{reject}}, \sqcup, \mathrm{R})$
q_{odd}	$(q_{\mathrm{odd}}, 0, \mathrm{R})$	$(q_{\mathrm{even}}, 1, \mathrm{R})$	$(q_{\mathrm{accept}}, \sqcup, \mathrm{R})$
q_{accept}			
q_{reject}