CMSC 28100

Introduction to Complexity Theory

Spring 2024
Instructor: William Hoza

The Church-Turing Thesis

- Let L be a language

Church-Turing Thesis:

There exists an "algorithm" / "procedure" for figuring out whether a given string is in L if and only if there exists a Turing machine that decides L.

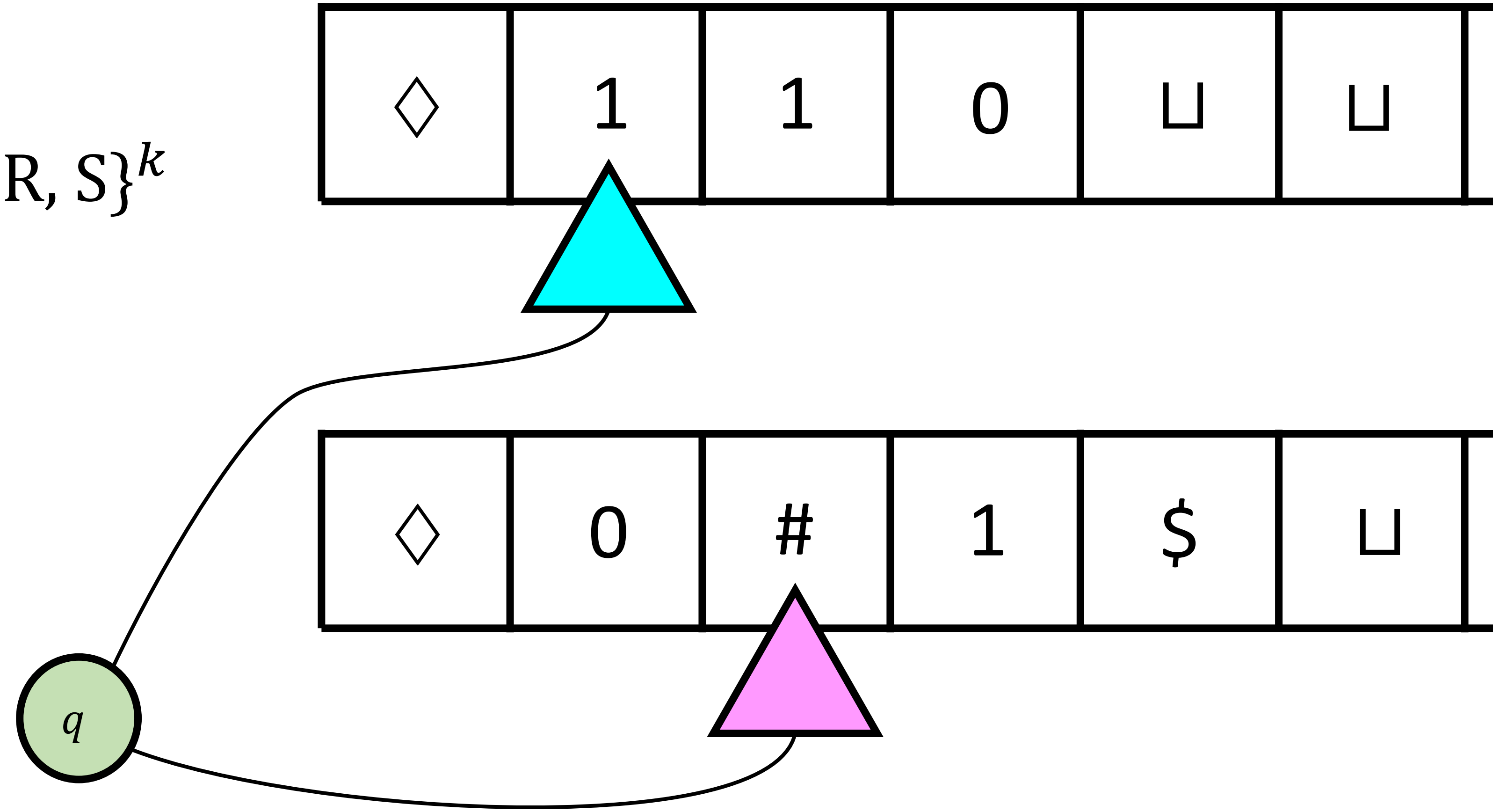
← Intuitive notion

← Mathematically precise notion

Multi-tape Turing machines

- "k-tape TM"
- Transition function:

$$\delta: Q \times \Gamma^k \to Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$$



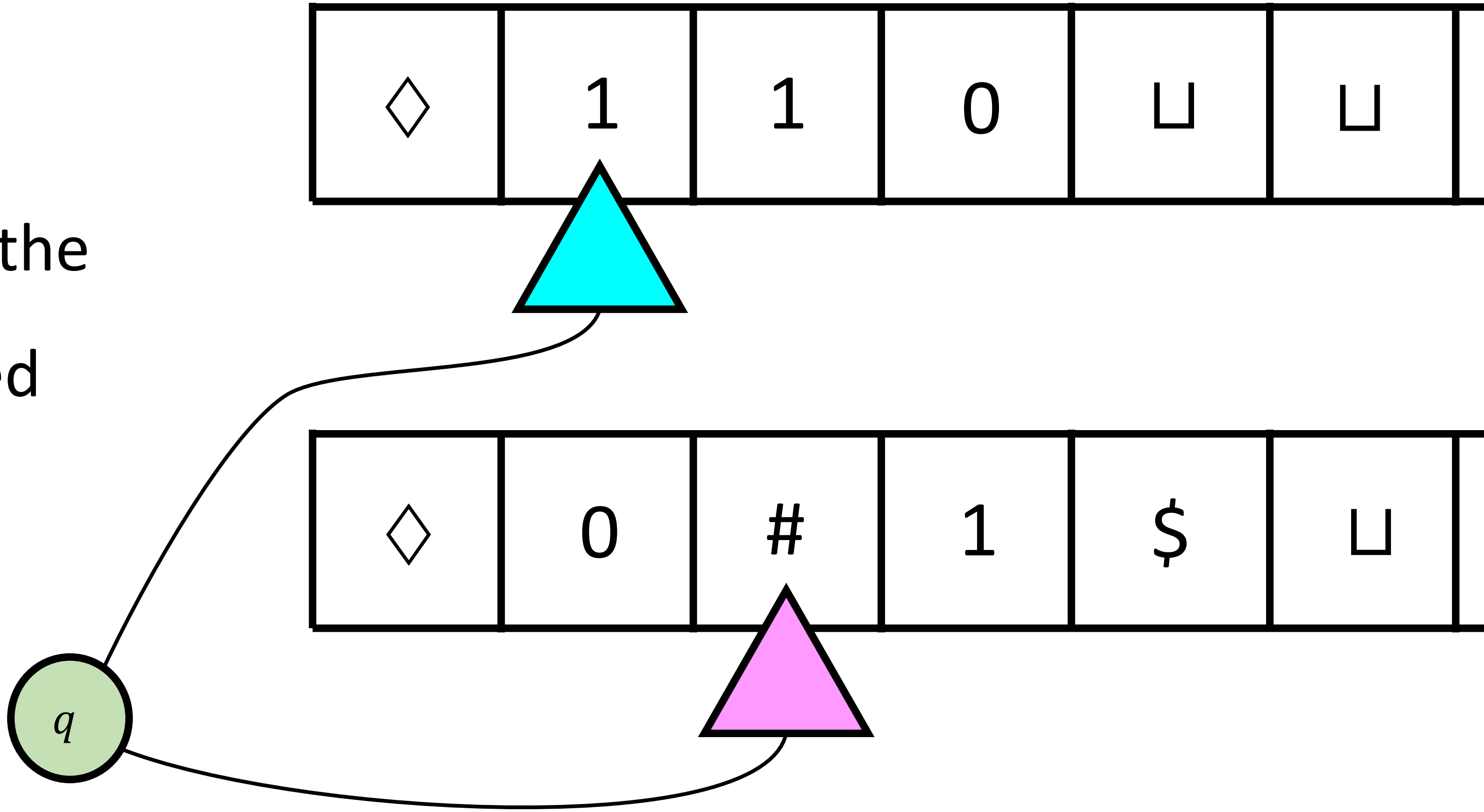
Multi-tape Turing machines

- Let k be any positive integer and let L be a language

Theorem: There exists a k-tape TM that decides L if and only if there exists a 1-tape TM that decides L

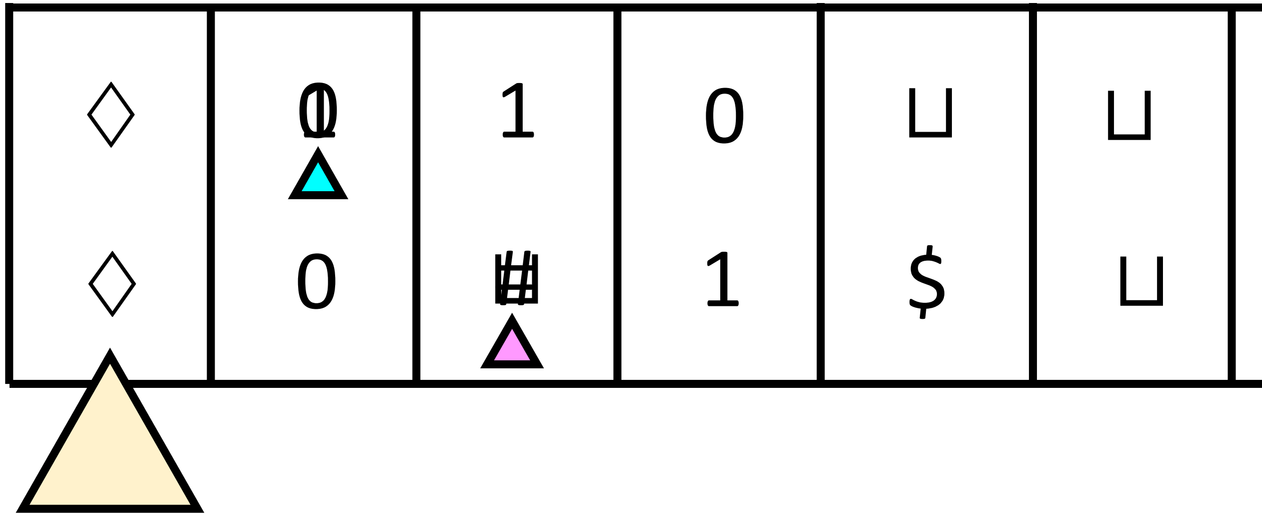
Simulating k tapes with 1 tape

- Idea: Pack a bunch of data into each cell
- Store "simulated heads" on the tape, along with k "simulated symbols" in each cell



Simulating k tapes with 1 tape

- Idea: Pack a bunch of data into each cell
- Store "simulated heads" on the tape, along with k "simulated symbols" in each cell



- The one "real head" will scan back and forth, updating the simulated heads' locations and the simulated tape contents. (Details on the next slides)

Simulating k tapes with 1 tape

- Let $M = (Q, \Sigma, \Gamma, \diamondsuit, \sqcup, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a k-tape Turing machine that decides L
- We will define a 1-tape Turing machine

$$M' = (Q', \Sigma, \Gamma', \diamondsuit, \sqcup, \delta', q_0', q_{\text{accept}}, q_{\text{reject}})$$

that also decides L

Simulating k tapes with 1 tape: Alphabet

- Let $\Lambda = \Gamma \cup \{\underline{b} : b \in \Gamma\}$, i.e., two disjoint copies of Γ
  - Interpretation: An underline indicates the presence of a simulated head
- New alphabet: $\Gamma' = \{\Diamond, \sqcup\} \cup \left\{ \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} : b_1, \dots, b_k \in \Lambda \right\}$
  - Interpretation: One symbol in Γ' is one "simulated column" of M
- Identify each input symbol $b \in \Sigma$ with the new symbol $\begin{pmatrix} b \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix}$, so $\Sigma \subseteq \Gamma'$

Simulating k tapes with 1 tape: Head statuses

- At each moment, each simulated head will have one of the following statuses:
  - "$\rightarrow b, D$" where $b \in \Gamma$ and $D \in \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}$
    - Interpretation: The simulated head needs to write b and move in direction D
  - "👻"
    - Interpretation: The simulated head is not currently depicted on the real tape; the simulated head's location is currently the same as the real head's location
  - "$b \rightarrow$" where $b \in \Gamma$
    - Interpretation: In the next simulated step, the simulated head will read b

Simulating k tapes with 1 tape: Head statuses

- Let Ω be the set of all possible statuses for a single simulated head:

$$\Omega = \{\text{“} \rightarrow b, D\text{”} : b \in \Gamma, D \in \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}\}$$

$$\cup \{\text{“👻”}\}$$

$$\cup \{\text{“}b \rightarrow \text{”} : b \in \Gamma\}$$

Simulating k tapes with 1 tape: States

- New state set:

$$Q' = \{q_{\text{accept}}, q_{\text{reject}}\} \cup \left\{ \begin{pmatrix} s_1 \\ \vdots \\ s_k \end{pmatrix}_{q,D} : s_1, \ldots, s_k \in \Omega; \quad q \in Q; \quad D \in \{\text{L}, \text{R}\} \right\}$$

- Interpretation:
  - Simulated head j has status s_j
  - The simulated machine is in state q
  - The one real head is making a pass over the tape in direction D

Simulating k tapes with 1 tape: Start state

- New start state:

$$q_0' = \begin{pmatrix} \text{“👻”} \\ \vdots \\ \text{“👻”} \end{pmatrix}_{q_0,\mathrm{R}}$$

Simulating k tapes with 1 tape: Transitions

- The new transition function will have the form

$$\delta' : Q' \times \Gamma' \rightarrow Q' \times \Gamma' \times \{\mathrm{L}, \mathrm{R}\}$$

Simulating k tapes with 1 tape: Transitions

- Let $\delta'\left(\begin{pmatrix} s_1 \\ \vdots \\ s_k \end{pmatrix}_{q,D}, \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix}\right) = \left(\begin{pmatrix} s'_1 \\ \vdots \\ s'_k \end{pmatrix}_{q,D}, \begin{pmatrix} b'_1 \\ \vdots \\ b'_k \end{pmatrix}, D\right)$ where s'_j, b'_j are defined by:
- If $s_j =$ "👻": Let $b'_j = \underline{b_j}$ and $s'_j =$ "$b_j \rightarrow$"
- If $s_j =$ "$\rightarrow c_j, S$" and b_j has an underline: Let $b'_j = \underline{c_j}$ and $s'_j =$ "$c_j \rightarrow$"
- If $s_j =$ "$\rightarrow c_j, D$" and b_j has an underline: Let $b'_j = c_j$ and $s'_j =$ "👻"
- In all other cases: Let $b'_j = b_j$ and $s'_j = s_j$

Simulating k tapes with 1 tape: Transitions

- Let $\delta'\left(\begin{pmatrix} s_1 \\ \vdots \\ s_k \end{pmatrix}_{q,\mathrm{R}}, \sqcup\right) = \left(\begin{pmatrix} s_1' \\ \vdots \\ s_k' \end{pmatrix}_{q,\mathrm{L}}, \begin{pmatrix} b_1' \\ \vdots \\ b_k' \end{pmatrix}, \mathrm{L}\right)$ where s_j', b_j' are defined by:
- If $s_j =$ "👻": Let $b_j' = \underline{\sqcup}$ and $s_j' =$ " $\sqcup \rightarrow$ "
- In all other cases: Let $b_j' = \sqcup$ and $s_j' = s_j$

Simulating k tapes with 1 tape: Transitions

- What do we do when we see $\Diamond$? Let $s_1, \dots, s_k \in \Omega$ (head statuses) and let $q \in Q$
- Assume that $\forall j$, either $s_j =$ "$b_j \to$ " or $s_j =$ "👻". In the latter case, let $b_j = \Diamond$
- Let $(q', c_1, \dots, c_k, D_1, \dots, D_k) = \delta(q, b_1, \dots, b_k)$
- If $s_j =$ "$b_j \to$ ", let $s_j' =$ " $\to c_j, D_j$". If $s_j =$ "👻", let $s_j' =$ "👻"
- Let $\delta'\left(\begin{pmatrix} s_1 \\ \vdots \\ s_k \end{pmatrix}_{q,\mathrm{L}}, \Diamond\right) = q'$ if q' is a halting state and $\left(\begin{pmatrix} s_1' \\ \vdots \\ s_k' \end{pmatrix}_{q',\mathrm{R}}, \Diamond, \mathrm{R}\right)$ otherwise

Simulating k tapes with 1 tape

- That completes the definition of M'
- Exercise: Rigorously prove that M' decides L

TMs can simulate all "reasonable" machines

- We could add various other bells and whistles to the basic TM model
  - The ability to observe the two neighboring cells
  - A tape that extends infinitely in both directions
  - A two-dimensional tape
- None of these changes has any effect on the power of the model

The Church-Turing Thesis

- Let L be a language

Church-Turing Thesis:

There exists an “algorithm” / “procedure” for figuring out whether a given string is in L if and only if there exists a Turing machine that decides L.

← Intuitive notion

← Mathematically precise notion

Are Turing machines powerful enough?

- **OBJECTION:** "To encompass all possible algorithms, the model would need to be as powerful as high-level programming languages, such as Python."
- **RESPONSE:** I claim that if there exists a Python script that decides L, then there exists a Turing machine that decides L

```
# Assumption: x, y, z are
# nonnegative integers
def f(x, y, z):
    r = 0
    while (r < y):
        r = r + x
    return (r < z)
```

- We won't actually prove this claim, but let's briefly discuss the process of converting Python code to Turing machines

Step 1: Operate at the level of individual bits

```
# Assumption: x, y, z are
# nonnegative integers
def f(x, y, z):
    r = 0
    while (r < y):
        r = r + x
    return (r < z)
```

⇓

```
# Assumption: x, y, z are
# lists of bits starting with
# the *least* significant
def f(x, y, z):
    r = [0]
    while (lessThan(r, y)):
        addTo(x, r)
    return lessThan(r, z)
```

```
def lessThan(x, y):
    i = max(len(x) - 1, len(y) - 1)
    while i >= 0:
        # Assumption: IndexError ⇒ 0
        if (x[i] < y[i]): return True
        if (x[i] > y[i]): return False
        i = i - 1
    return False

def addTo(x, y):
    c = 0
    for i in range(max(len(x), len(y))):
        # Assumption: IndexError ⇒ 0
        b = x[i] ^ y[i] ^ c
        c = (x[i] & y[i]) | (x[i] & c)
            | (y[i] & c)
        y[i] = b
    y.append(c)
```

Step 2: Eliminate subroutines

```
# Assumption: x, y, z are
# lists of bits starting with
# the *least* significant
def f(x, y, z):
    r = [0]
    while (lessThan(r, y)):
        addTo(x, r)
    return lessThan(r, z)
```

⇒

```
def f(x, y, z):
    r = [0]
    whileCondition = False
    i = max(len(r) - 1, len(y) - 1)
    while i >= 0:
        if (r[i] < y[i]):
            whileCondition = True
            break
        if (r[i] > y[i]):
            whileCondition = False
            break
        i = i - 1
    if (whileCondition):
        c = 0
        for i in range(max(len(x), len(r))):
            b = x[i] ^ r[i] ^ c
            c = (x[i] & r[i]) | (x[i] & c)
                | (r[i] & c)
            r[i] = b
        r.append(c)
        whileCondition = False
        i = max(len(r) - 1, len(y) - 1)
        while i >= 0:
            if (r[i] < y[i]):
                whileCondition = True
                break
            if (r[i] > y[i]):
                whileCondition = False
                break
            i = i - 1
    i = max(len(r) - 1, len(z) - 1)
    while i >= 0:
        if (r[i] < z[i]): return True
        if (r[i] > z[i]): return False
        i = i - 1
    return False
```

Step 3: From code to Turing machines

- Basic idea:
  - Variable ⇒ Tape (assuming the variable holds a list of bits)
  - List index ⇒ Head
  - Line of code ⇒ State

Step 3: From code to Turing machines

```
⋮
3      whileCondition = False
4      i = max(len(r) - 1, len(y) - 1)
5      while i >= 0:
6          if (r[i] < y[i]):
7              whileCondition = True
8              break
9          if (r[i] > y[i]):
10             whileCondition = False
11             break
12         i = i - 1
13     if (whileCondition):
14  …
⋮
```

⇒

- State “4”:
  - If the “r” head and the “y” head both see ⊔, move them both to the left and go to state “5”.
  - Otherwise, move those heads to the right and go to state “4”.
- State “5”:
  - If the “r” head sees 0 or ⊔ and the “y” head sees 1, go to state “14”.
  - If the “r” head sees 1 and the “y” head sees 0 or ⊔, go to state “31”.
  - If the “r” head and the “y” head see ♢, go to state “31”.
  - Otherwise, move those heads to the left and go to state “5”.

Turing machines as a programming language

- You can think of the Turing machine model as a primitive programming language