Lecture 4: Turing machines

Mika Göös

EPFL School of Computer and Communication Sciences

Lecture 4

Recall: **Pumping Lemma**

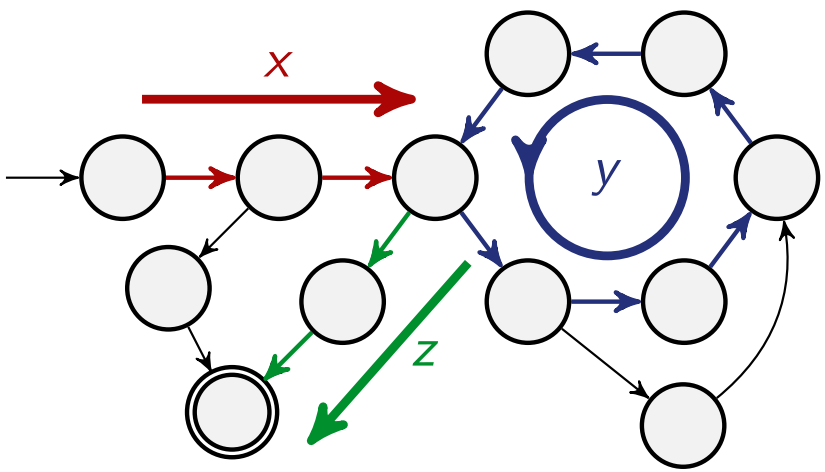
(and how to prove that a language is not regular)

Pumping Lemma and its Proof Sketch

If A is a regular language, then there is a number p (the pumping length) such that, for every string s in A of length at least p, there exists a division of s into three pieces, $s = xyz$ s.t.

1. for each $i \geq 0$, $xy^i z \in A$
2. $|y| \geq 1$, and
3. $|xy| \leq p$.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ s.t. $L(M) = A$
- Consider a string $s \in A$ s.t. $|s| \geq |Q| = p$
- Stop once in a state for the 2nd time, say at times j, k
- $s = xyz$ where x is the first j letters, y is letter $j + 1$ to k, z is from $k + 1$ to end



$$F = \{ww \mid w \in \{0,1\}^*\}$$

F is not regular!

Proof: (by contradiction)

- Assume F is regular, let p be its pumping length
- Pick $s = 0^p1^p0^p1^p \in F$

  All strings don't work! Fun part is guessing which string to pick
- Pumping lemma: $s = xyz, |xy| \leq p, |y| \geq 1, xy^iz \in F$ for all $i \geq 0$

  Pumping lemma tells us there is such a decomposition – *we can't choose it!* Your reasoning should work for any decomposition
- Since $|xy| \leq p$ and $|y| \geq 1$, $y = 0^k$ for some $k > 0$
- According to pumping lemma, $xy^2z \in F$
- $xy^2z = 0^{p+k}1^p0^p1^p \notin F$
- **Contradiction!**

Part I of course

Lec 1: DFA and Regular Languages

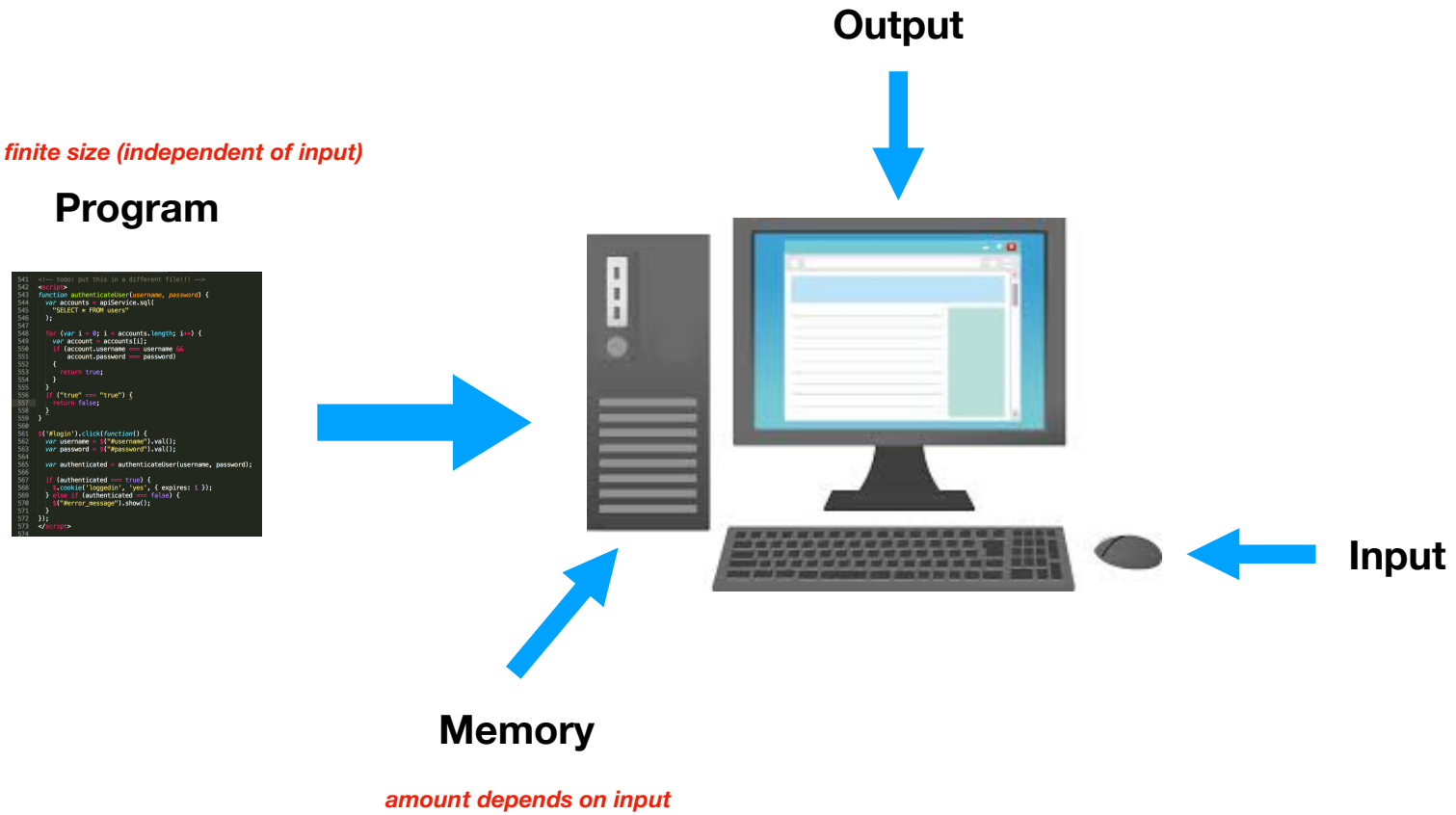
Lec 2: NFA and its equivalence to DFA

Lec 3: Non-regular languages and the Pumping Lemma

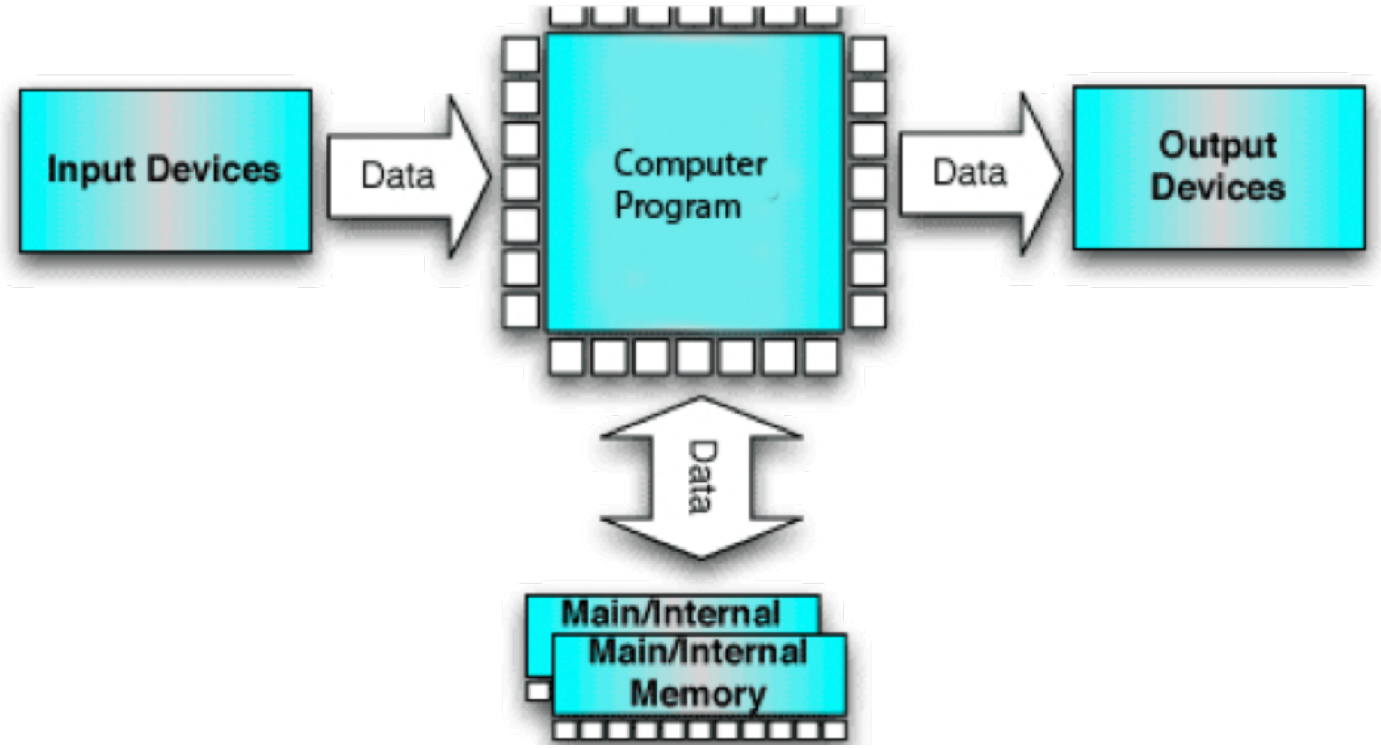
- $\{w \in \{0,1\}^* : w$ has the same number of 0's and 1's$\}$ cannot be recognized by DFAs
- This seems like a problem with DFAs

What's missing?

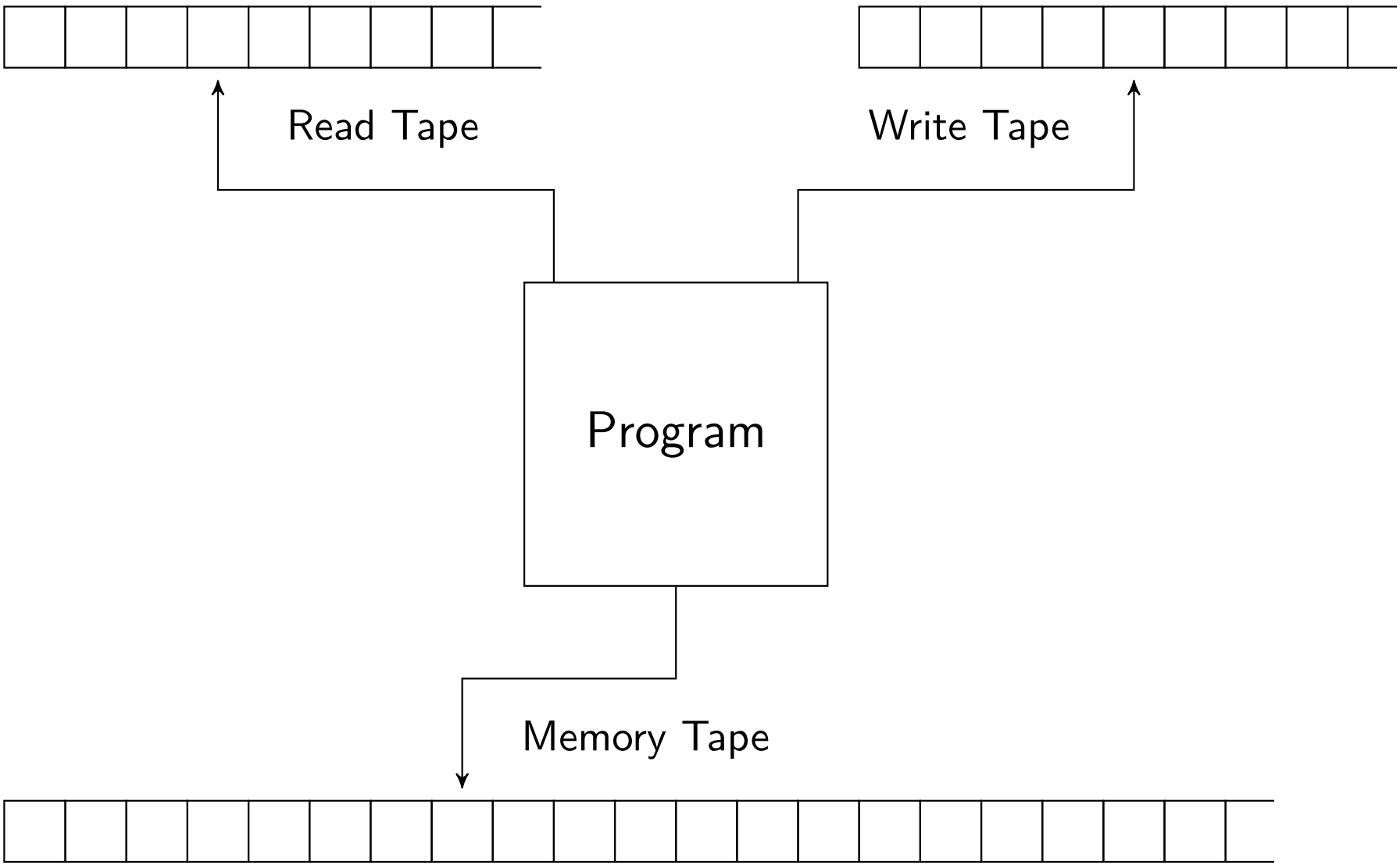
What is a computer?



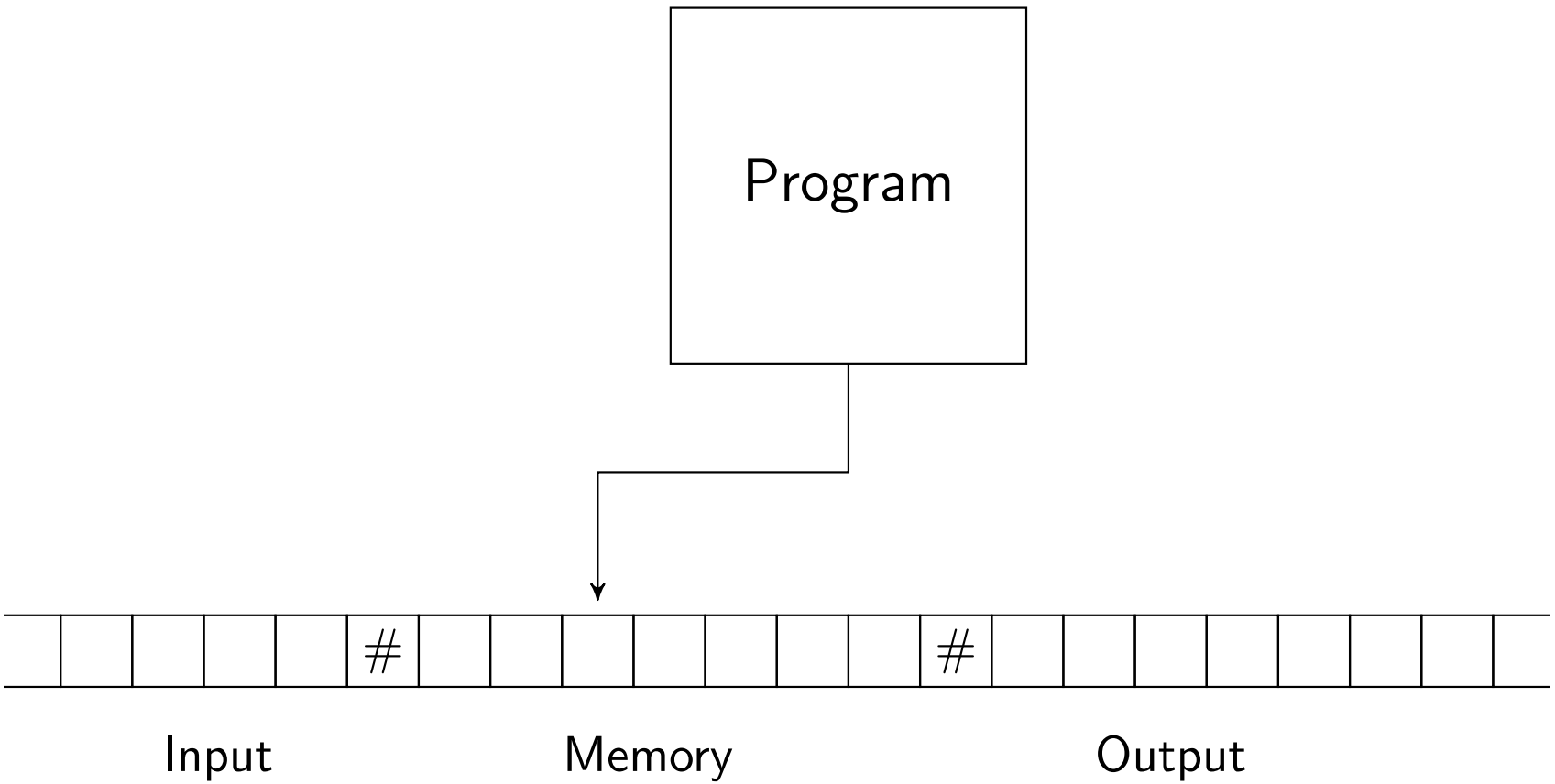
Abstractly


Input Devices
Data
Computer Program
Data
Output Devices
Data
Main/Internal
Main/Internal Memory

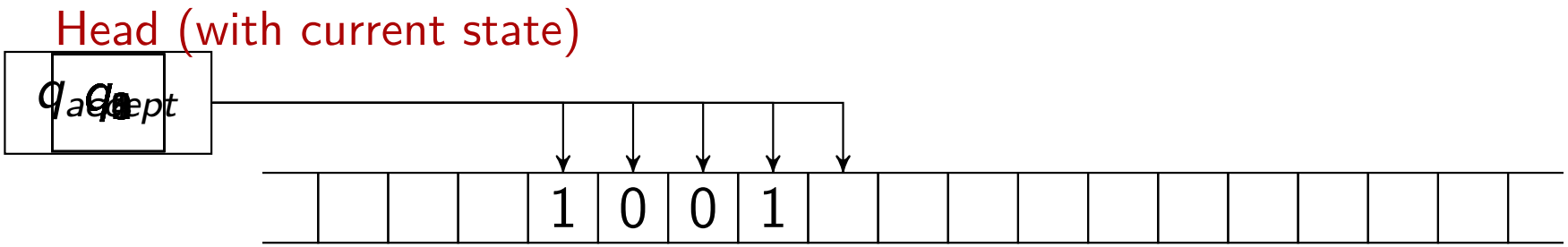
More abstractly

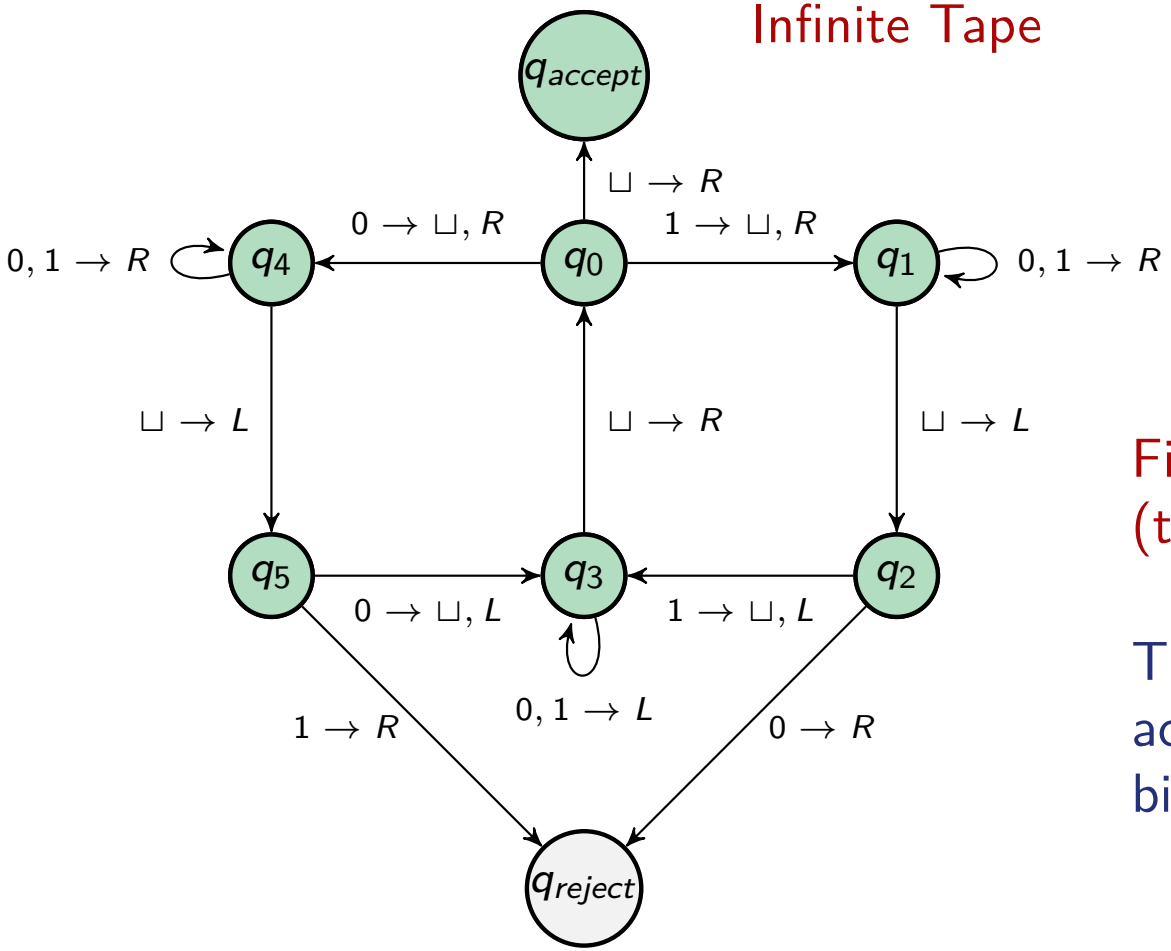


Single tape seems enough. . .



The Turing Machine

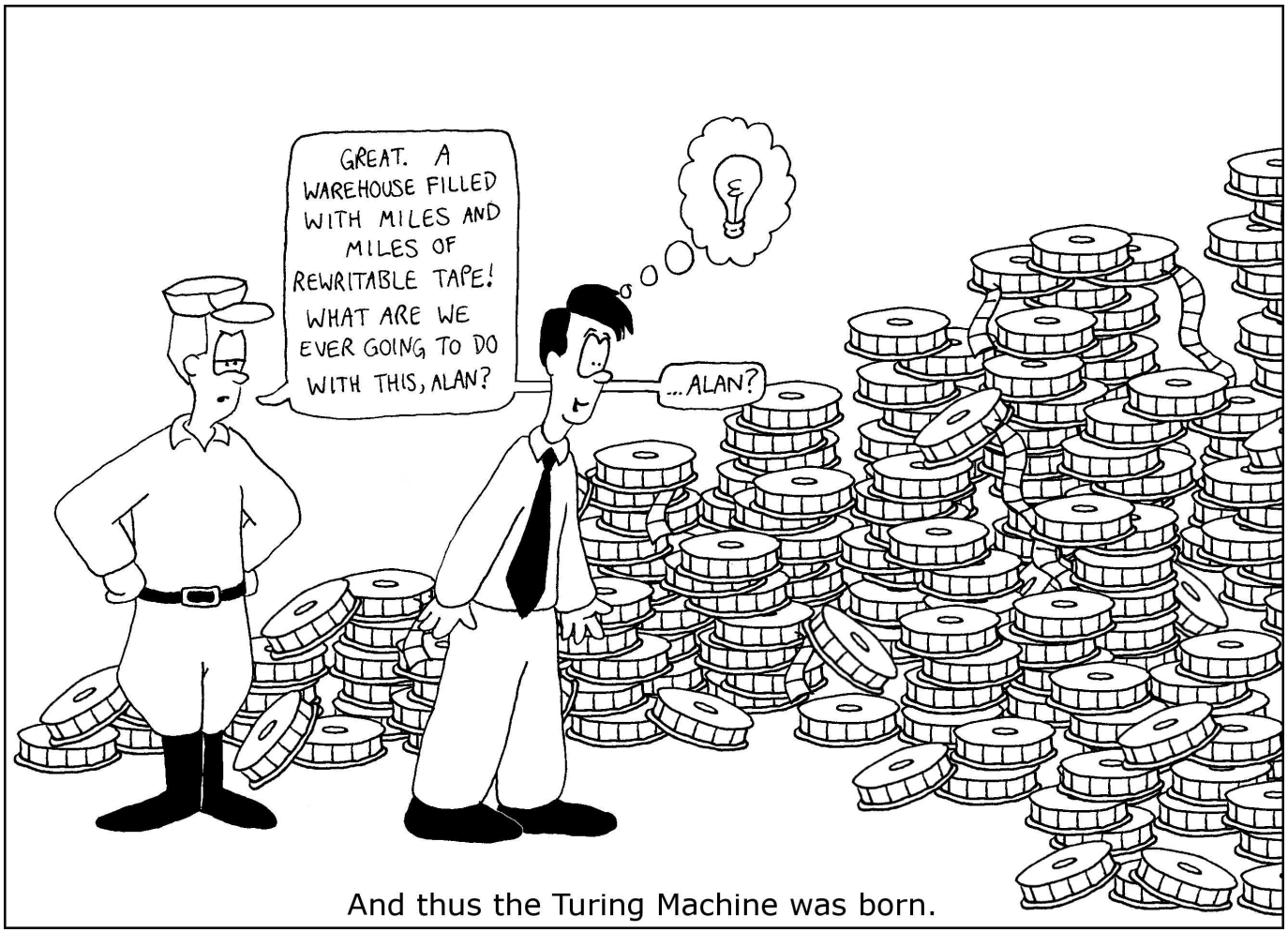




Finite state control (the algorithm)

This Turing machine accepts even length binary palindromes

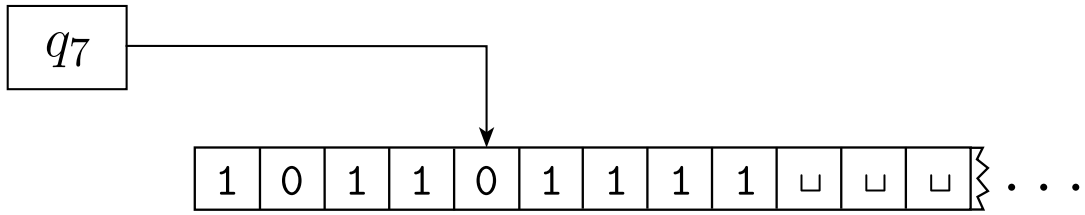
Finite size program, larger and larger instances $\Rightarrow$ Infinite Tape!



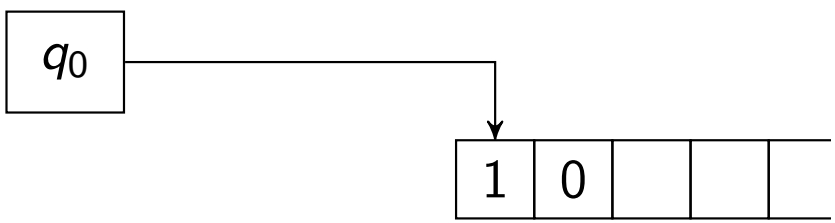
Finite automata vs Turing machines

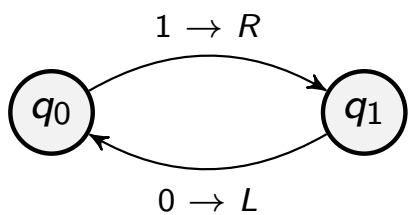
1. A Turing machine can both write on the tape and read from it
2. The read-write head can move both to the left and to the right
3. The tape is infinite
4. The special states for rejecting and accepting take effect immediately

Turing Machine



- Infinite tape
- Tape alphabet contains input alphabet plus ⊔ (blank symbol) plus maybe more symbols
- Head has states (corresponding to the finite control automata)
- Exactly **one** Accept state and exactly **one** Reject state *(where computation immediately ends)*
- Remaining states *"computation in progress"*
- May never reach an accept state. May never halt!





Formal Definition of a TM

A **Turing Machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the blank symbol $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{accept} \neq q_{reject}$.

$\{w : w$ has an equal number of 0's and 1's$\}$

Easy (efficient) to write an algorithm to count number of 1s and 0's — let's try to implement on a TM

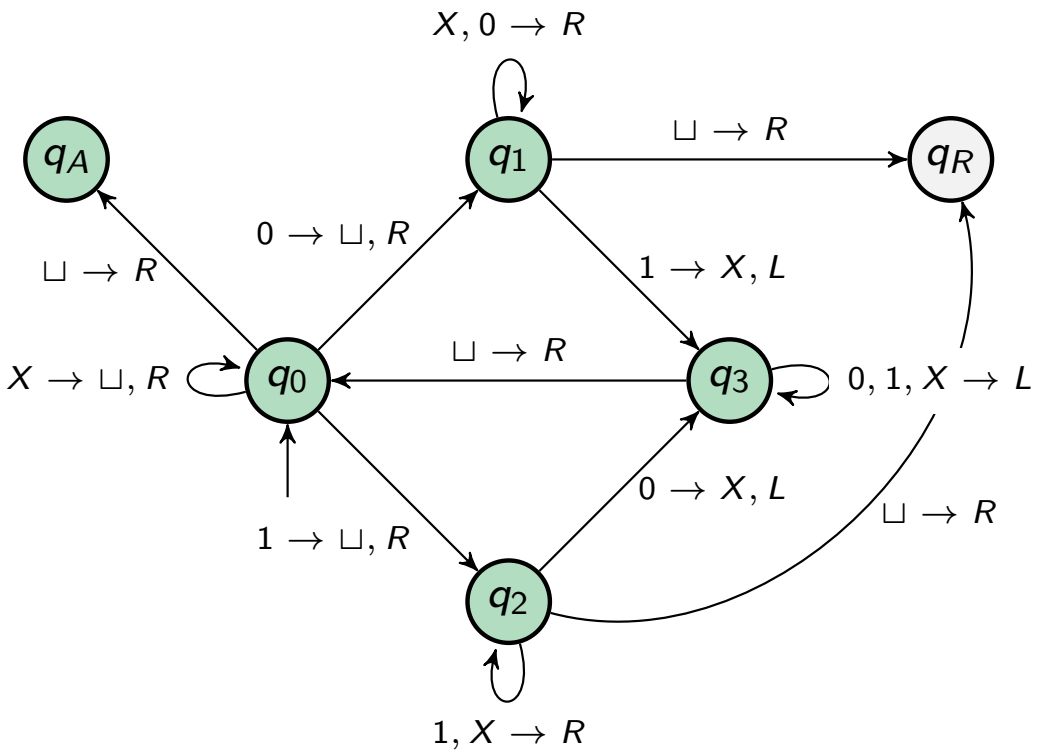
(In this Part II of the course, we do not care about running time. . .)

Our approach: check for each 0 (or 1) there is a corresponding 1 (or 0)

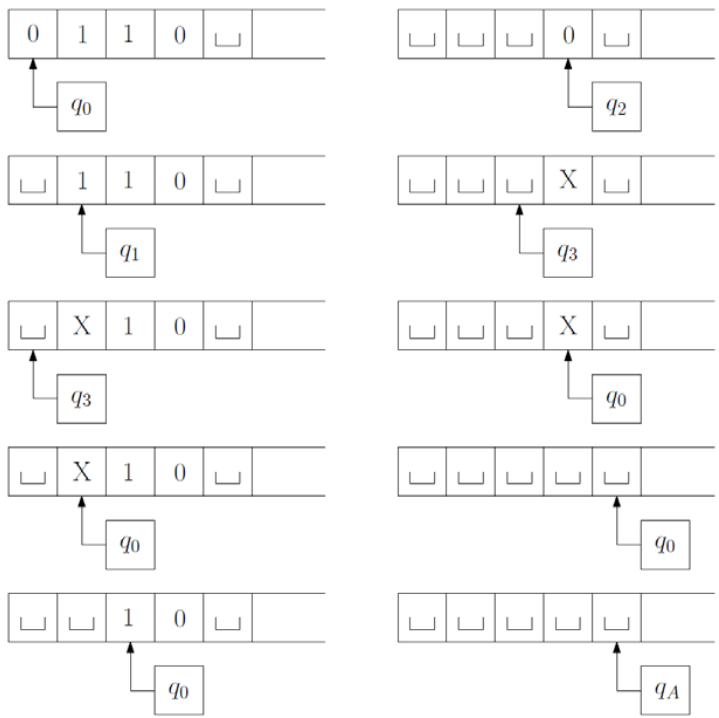
- Scan the input from left to right
- Whenever we encounter an uncrossed 0 or 1, we "remove" it and proceed right to find a corresponding 1 or 0 that we cross
- We keep doing this (2) until either we cross off all the letters (accept) or we fail to find a pair for one of the letters (reject)

$\{w : w \text{ has an equal number of 0's and 1's}\}$

$Q = \{q_0, q_1, q_2, q_3, q_A, q_R\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1, \sqcup, X\}$ – X crossed off letter



Execution of the TM on 0110



Configurations of a TM

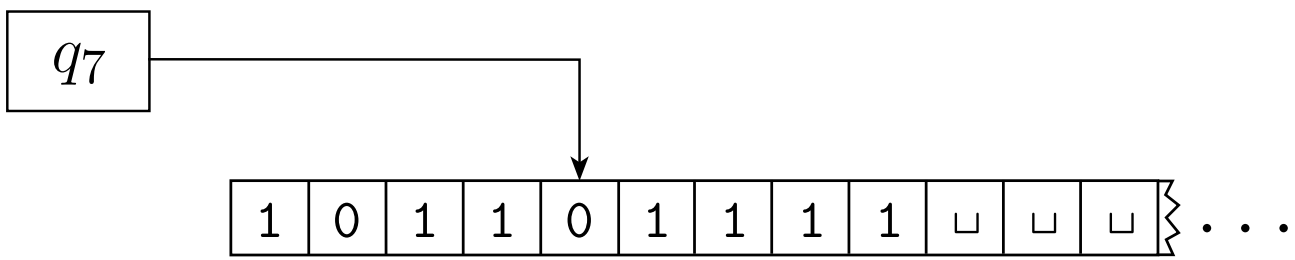
As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location. A setting of these three items is called a *configuration* of the Turing machine.

Representation: We write uqv where $u, v \in \Gamma^*$ and $q \in Q$ for the configuration where

- current state is q
- current tape content is uv
- the current head location is the first symbol of v

(Cells whose contents are unspecified are blank. If $u = \varepsilon$ then Head at leftmost cell.)

Example: A TM with configuration $1011q_701111$



Transitions: Given configuration uaq_ibv where $a, b \in \Gamma$, $u, v \in \Gamma^*$ and state $q_i \in Q$, we move to

- uq_jacv if $\delta(q_i, b) = (q_j, c, L)$
- $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$

Computation:

- Starting configuration is $C_1 = q_0w$ on input $w \in \Sigma^*$
- Obtain new configurations $C_2, C_3, \ldots$ by valid moves/transitions
- Accept and halt if a configuration with the state q_{accept} is reached
- Reject and halt if a configuration with the state q_{reject} is reached

What if the computation doesn't halt (i.e., loops)?
Does it mean some configuration is repeated?

Turing-Recognizable/Decidable Languages

A TM machine M recognizes a language $L \subseteq \Sigma^*$ iff for all inputs $w \in \Sigma^*$:

1. If $w \in L$ then M accepts w and
2. If $w \notin L$ then M doesn't halt (or it rejects w)

Such languages are called (Turing)-Recognizable

A TM machine M decides a language $L \subseteq \Sigma^*$ iff for all inputs $w \in \Sigma^*$:

1. M halts on w, and
2. M accepts w iff $w \in L$

Such languages are called (Turing)-Decidable

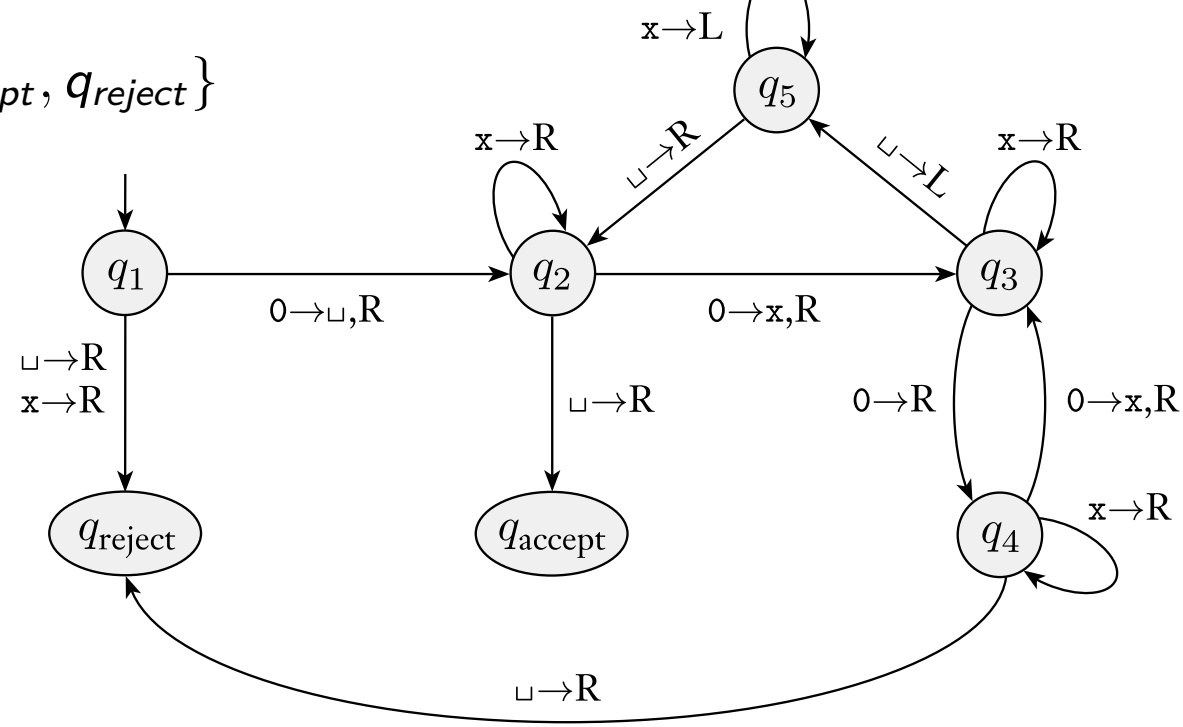
$$\{0^{2^n} : n \geq 0\}$$

TM that decides this language: On input string w

1. Remove first 0
2. Sweep left to right across the tape, crossing off every other 0
3. If in stage 1 the tape contained a single 0, *accept*
4. If in stage 1, the tape contained more than a single 0 and the number of 0s that we crossed out was odd, *reject*
5. Retun the head to the left-hand end of the tape
6. Go to stage 1

Each iteration of stage 1 cuts the number of 0s in half.

$Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$
$\Sigma = \{0\}$
$\Gamma = \{0, x, \sqcup\}$

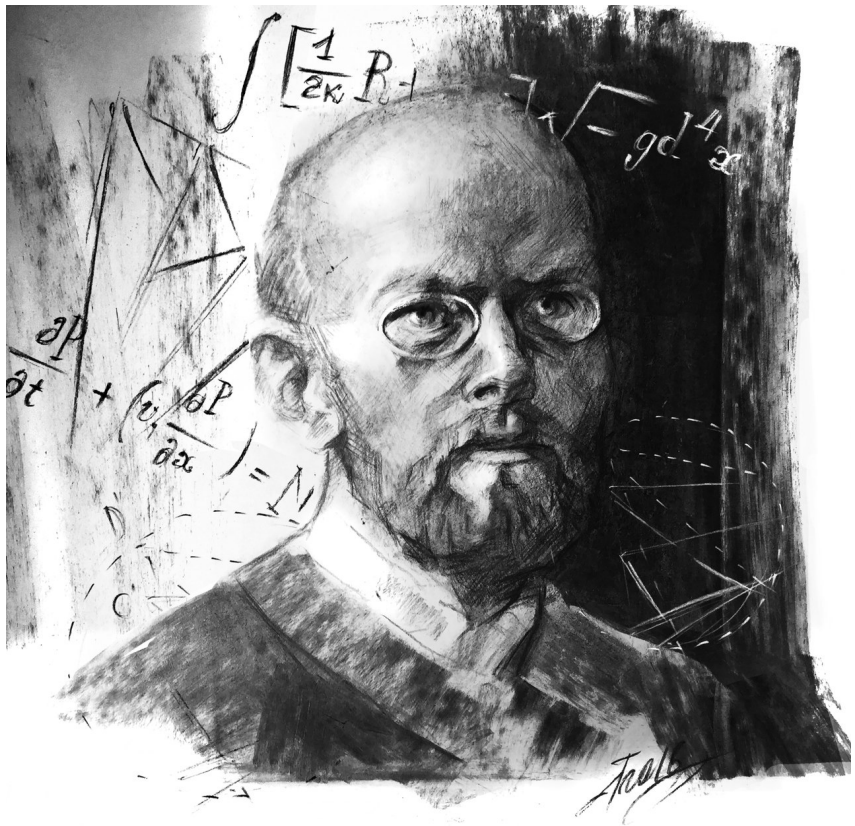


Example on input 0000:

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$\sqcup q_5 x x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

Equivalence to Other Models and History

In 1900, David Hilbert raised 23 mathematical problems



Hilbert's tenth problem

Given a Diophantine equation, **devise** a process according to which it can be determined in a finite number of operations whether the equation is soluble in integers.

- A Diophantine equation is a polynomial equation with integer coefficients and finite number of unknowns.
- E.g. $3x^2 - 2xy - y^2z - 7 = 0$ has solution $x = 1, y = 2, z = -2$
- E.g. $x^{2025} + y^{2025} = z^{2025}$ with $x, y, z \geq 1$? **No solution!**
  (I have a marvelous proof, which this slide is too narrow to contain)

Problem: in 1900 there was not a definition of what an algorithm is!

Definition of Algorithm

- The definition came in 1936 papers of Alonzo Church and Alan Turing
- Church used a notational system called λ-calculus to define algorithms
- Turing did it with his "machines"
- The two definitions were shown to be equivalent

Church-Turing thesis:

Intuitive notion of algorithms	equals	*Turing machine algorithms*

The precise definition of algorithms allowed Yuri Matijasevic in 1970, building on the work of Martin Davis, Hilary Putnam, and Julia Robinson, to resolve Hilbert's tenth problem. No such algorithm can exist!

Church-Turing Thesis

Intuitive notion of algorithms	equals	*Turing machine algorithms*

- All algorithms we know of can be executed on TMs
- Anything you write in C, Java, Scala, Python and so on
- Anything you would possibly do with a Quantum Computer
- The definition is also robust to variations: if we allow for many tapes instead of one, then nothing changes
- Any computational process in nature
- And so on

Next week: **Undecidability**