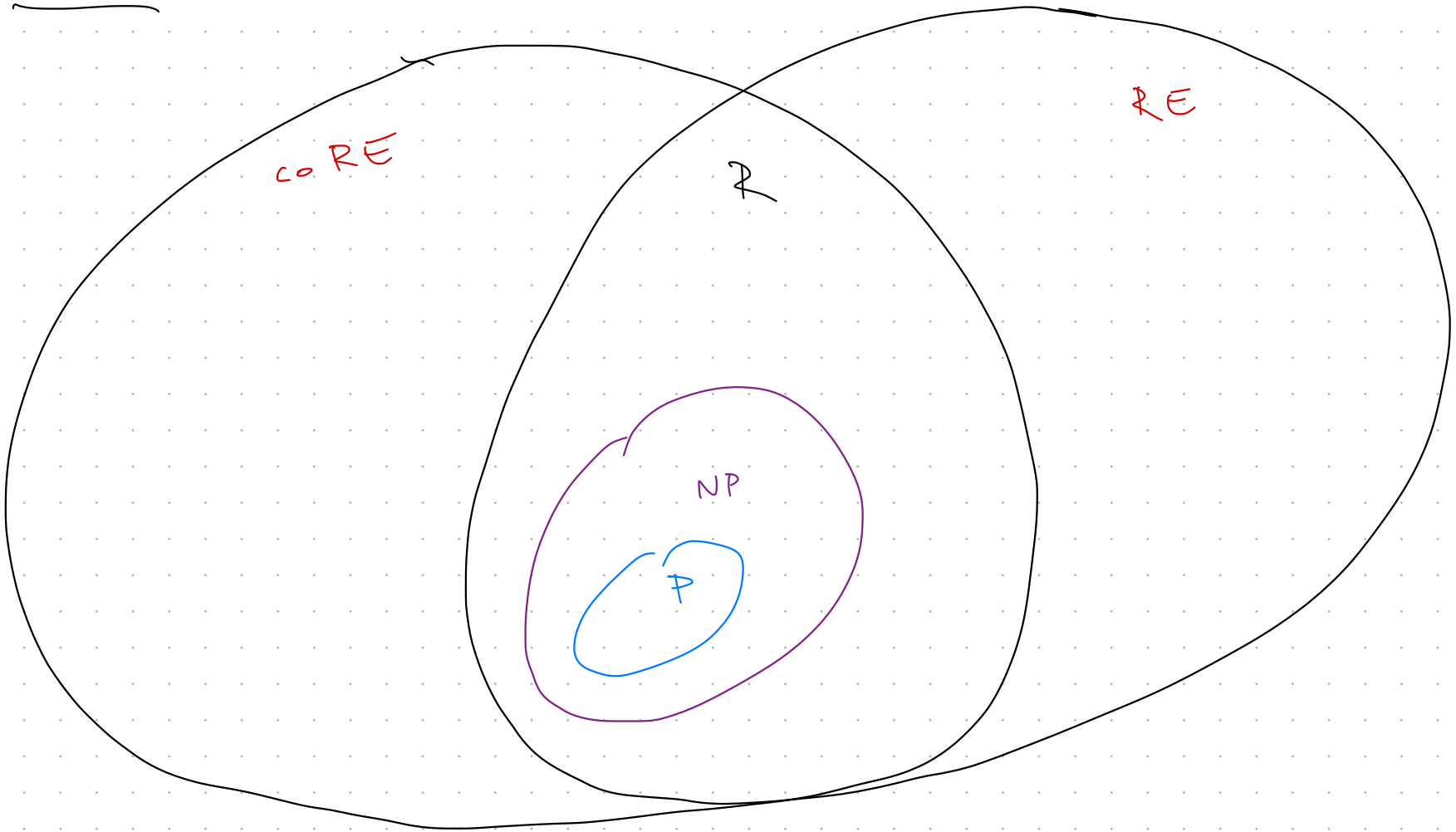
21 April 2025

Turing Machines & The Church-Turing Thesis

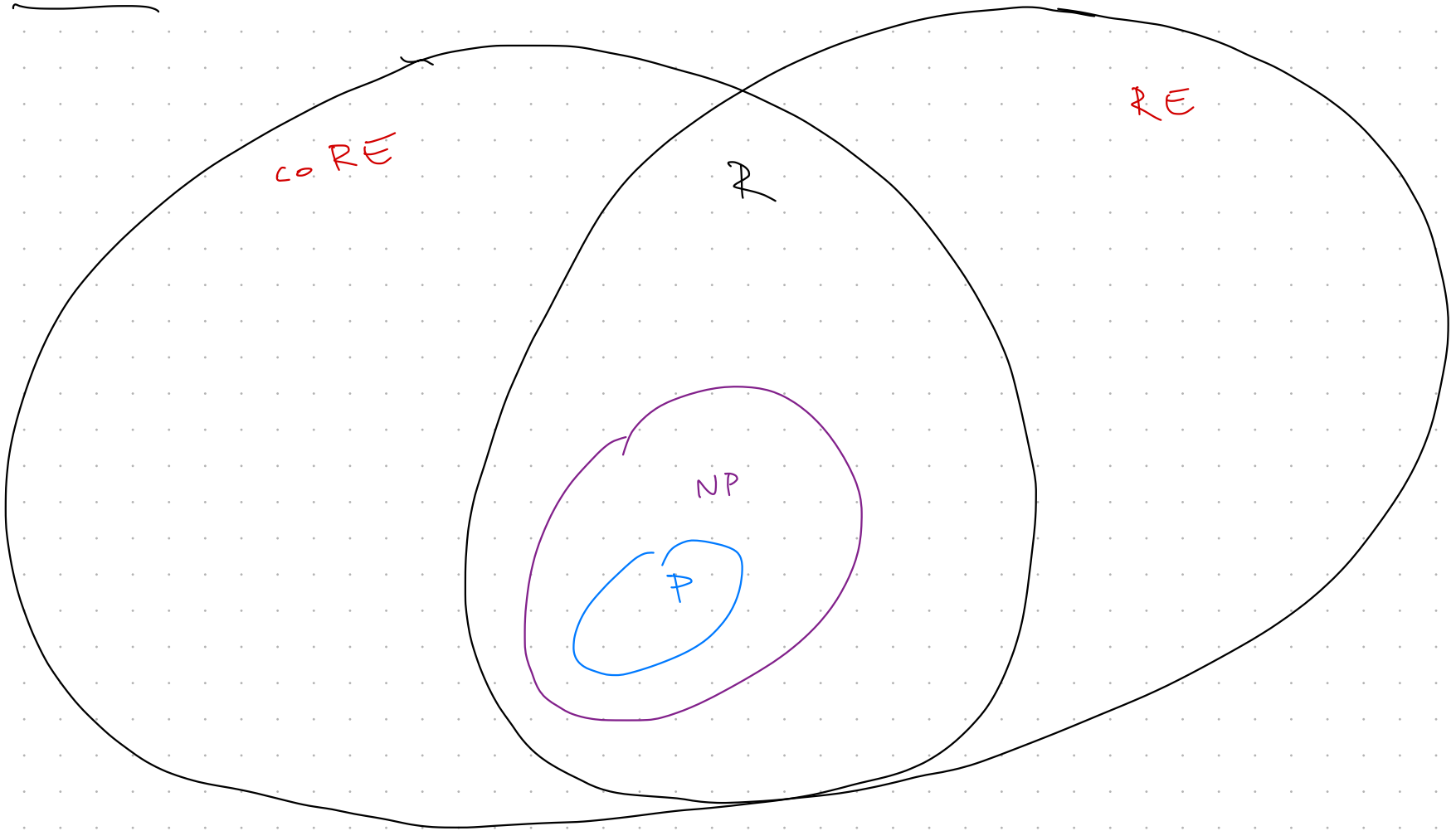
Plan

* Defining "Algorithms"
* Announcements
* TM Examples

So Far



So Far



A problem L is <u>decidable</u> (R) if there exists a <u>program</u> D such that

$x \in L \Rightarrow$ D accepts x

$x \notin L \Rightarrow$ D does not accept x

Theorem. There are undecidable problems.

↳ No program solves the problem.

Theorem. There are undecidable problems.

↳ No program solves the problem.

But what do we mean by "program"?

* Python program? C program?
* Physical device (transistors, abacus, quantum system)
* Human cognition?

Major Endeavor of 1920s – 1940s

↳ Define "Effective Computation"

Proposed Model (from Alan Turing)

* Turing Machine
  ↳ Obviously programmable
  ↳ Powerful enough to capture all other proposals (e.g. λ-calculus, μ-recursive fns)

Major Endeavor of 1920s – 1940s

↳ Define "Effective Computation"

Proposed Model (from Alan Turing)

* Turing Machine
  - ↳ Obviously programmable
  - ↳ powerful enough to capture all other proposals (e.g. λ-calculus, μ-recursive fns)

Church-Turing Thesis

Every physically-realizable computational model can be simulated by a Turing Machine.

Church-Turing Thesis

Everything computable, computable by a TM.

* Not a theorem. More like a physical law of nature.
  - very robust thesis
  - widely accepted

Church-Turing Thesis

Everything computable, computable by a TM.

* Not a theorem. More like a physical law of nature.

  - very robust thesis
  - widely accepted

Extended Church-Turing Thesis

Everything efficiently-computable is efficiently-computable by a TM.

- More controversial --- e.g. Quantum Computers

  ↳ But still very robust.

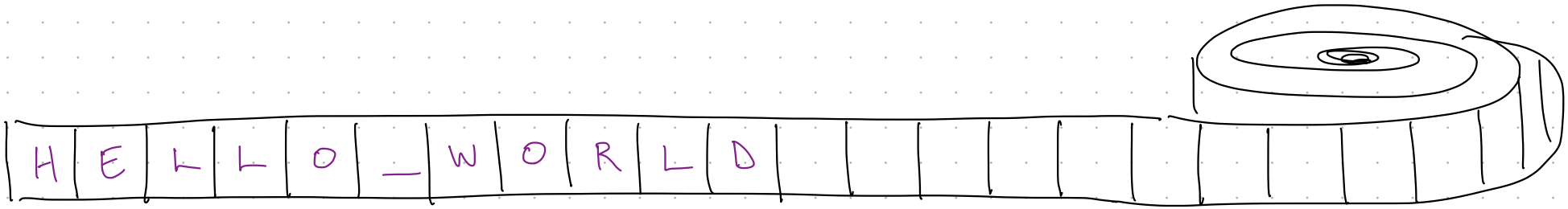
  (e.g. efficient simulation of python by TMs)

Announcements

* None ---

Turing Machines

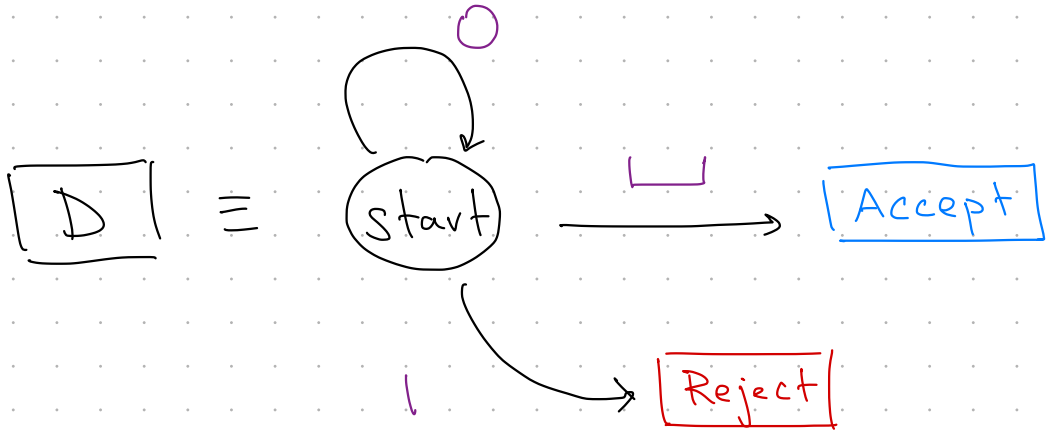
* Simple (finite) logical controller
* Infinite Read/Write memory tape.



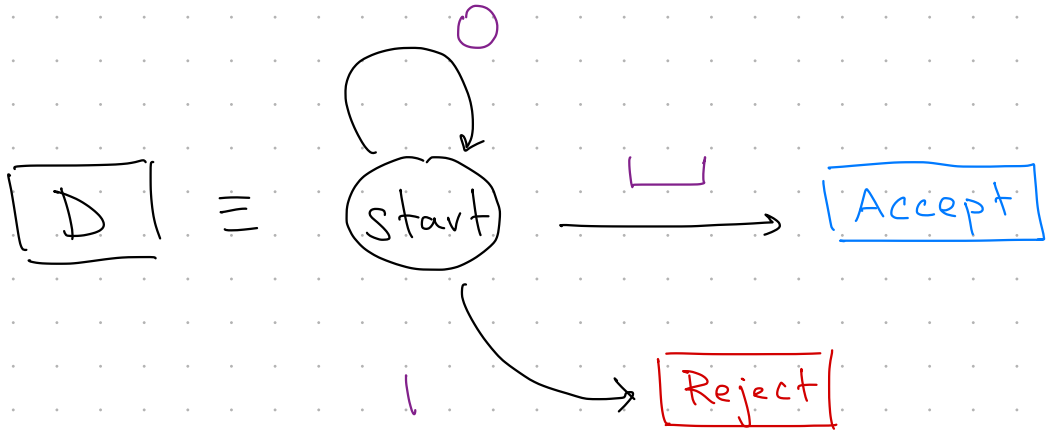
Computation

* Finite controller Reads cell under tape "head"
* Then,
  - Writes new symbol to cell
  - Moves tape head Left or Right
  - Updates internal (finite) state

Warmup: Deterministic Finite Automata

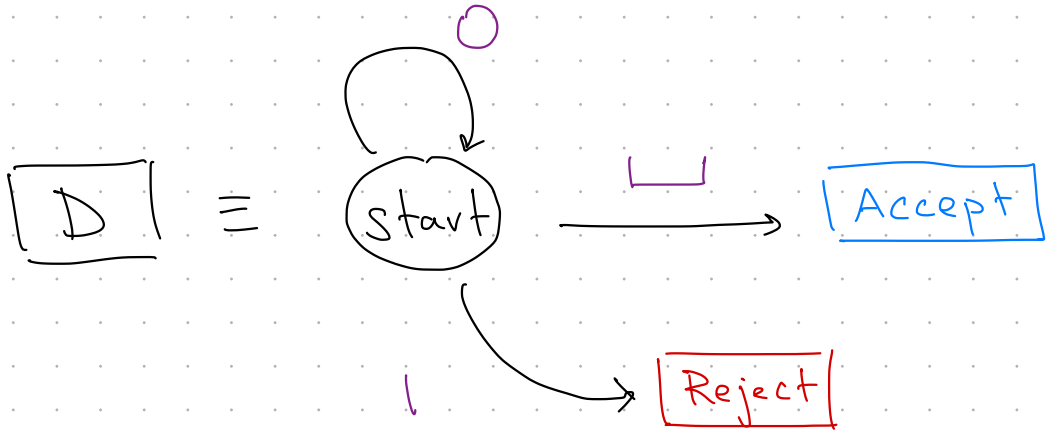


Warmup: Deterministic Finite Automata



$$\mathcal{L}(D) = \{ 0^n : n \in \mathbb{N} \}$$

Warmup: Deterministic Finite Automata

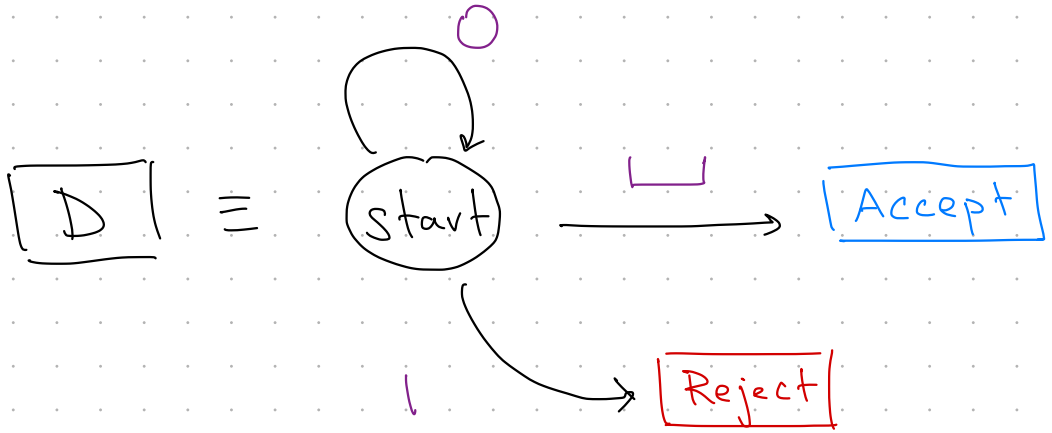


DFA defined by

$\mathcal{L}(D) = \{0^n : n \in \mathbb{N}\}$

* $\Sigma \equiv$ Input Alphabet
* $Q \equiv$ Finite set of internal states
* $q_0 \in Q \equiv$ start state
* $a \in Q \equiv$ Accept state
* $r \in Q \equiv$ Reject state

Warmup : Deterministic Finite Automata



DFA defined by

$\mathcal{L}(D) = \{ 0^n : n \in \mathbb{N} \}$

* Σ ≡ Input Alphabet
* Q ≡ Finite set of internal states
* $q_0 \in Q$ ≡ start state
* $a \in Q$ ≡ Accept state
* $r \in Q$ ≡ Reject state
* δ ≡ transition rule

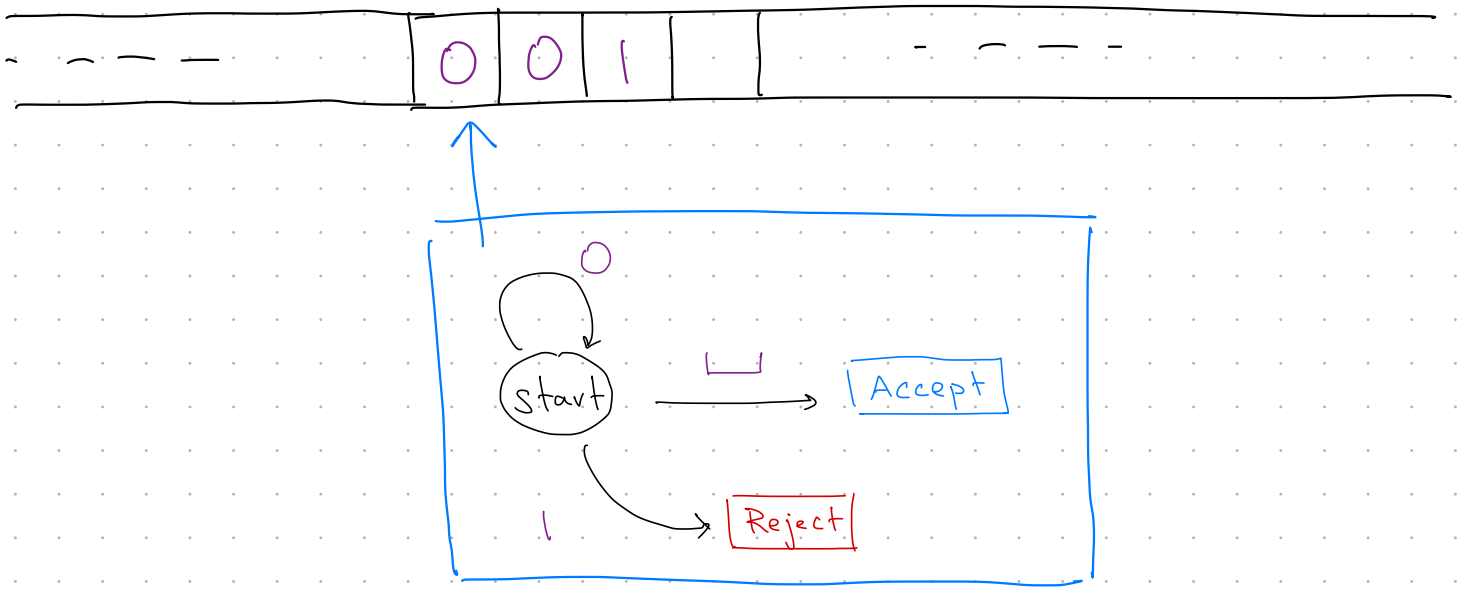
$$\delta(q, \sigma) \longrightarrow q' \in Q$$

new state

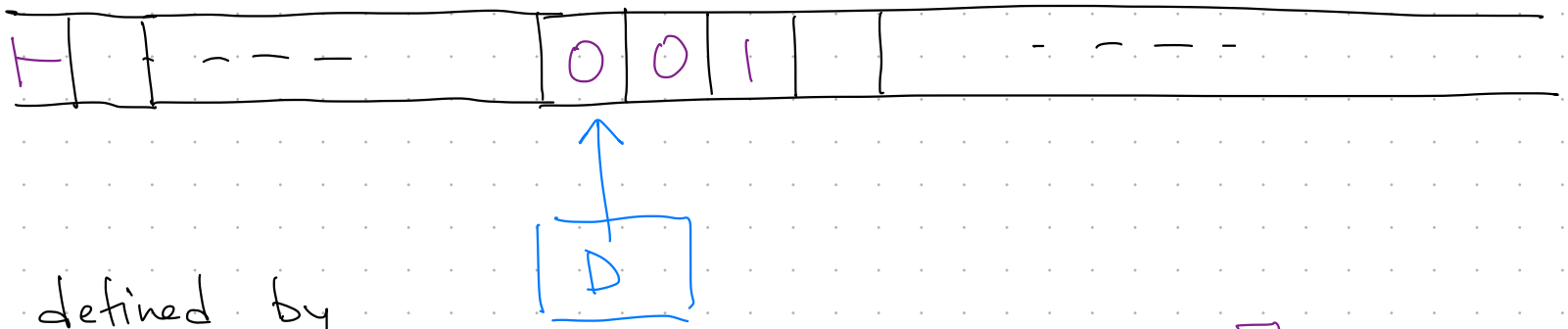
current state

symbol read

Turing Machines



Turing Machines



TM defined by

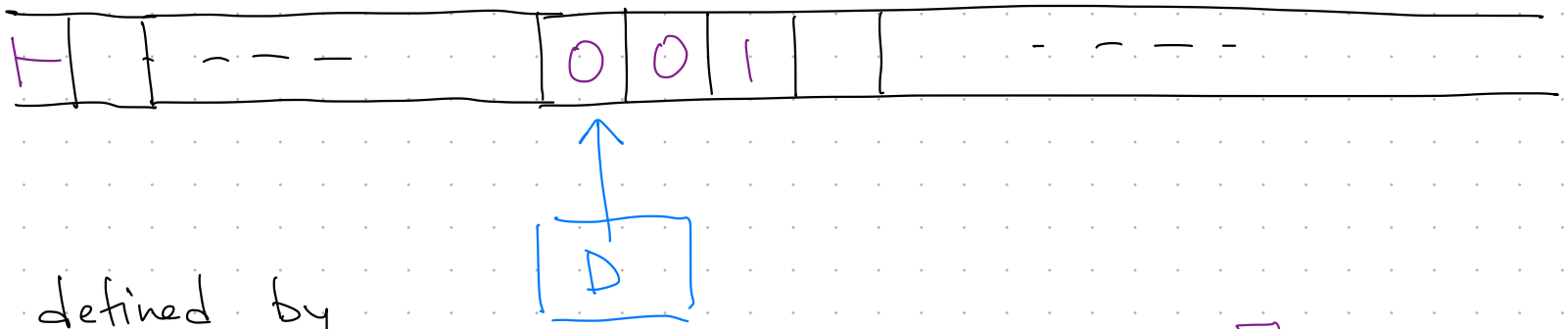
* Σ ≡ Input Alphabet
  Γ ≡ Tape Alphabet

$\Sigma \subseteq \Gamma$

$\sqcup \in \Gamma$ ≡ blank

$\vdash \in \Gamma$ ≡ Left end of tape

Turing Machines



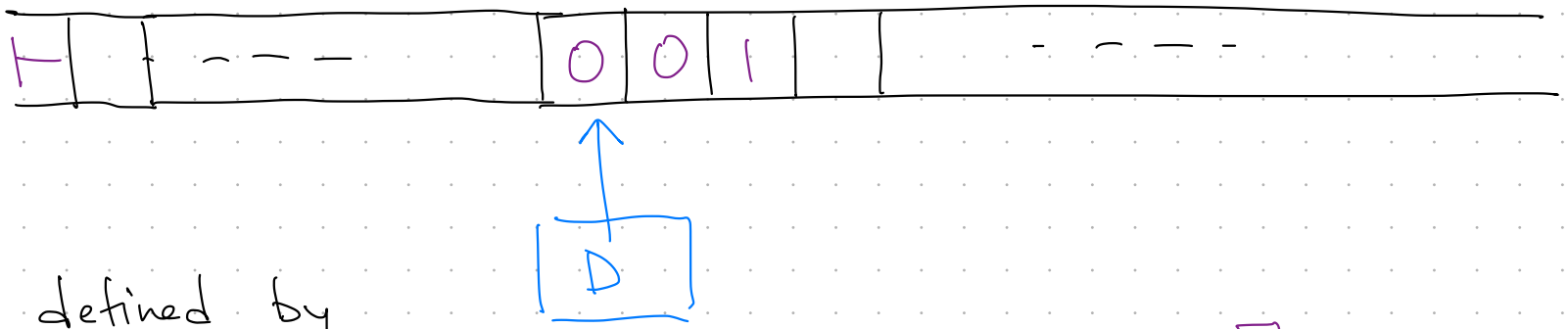
TM defined by

* Σ ≡ Input Alphabet
  Γ ≡ Tape Alphabet

  $\Sigma \subseteq \Gamma$
  $\sqcup \in \Gamma$ ≡ blank
  $\vdash \in \Gamma$ ≡ Left end of tape

* Q ≡ Finite set of internal states
  including **start** Accept Reject

Turing Machines



TM defined by

* $\Sigma \equiv$ Input Alphabet
* $\Gamma \equiv$ Tape Alphabet

$\Sigma \subseteq \Gamma$

$\sqcup \in \Gamma \equiv$ blank

$\vdash \in \Gamma \equiv$ Left end of tape

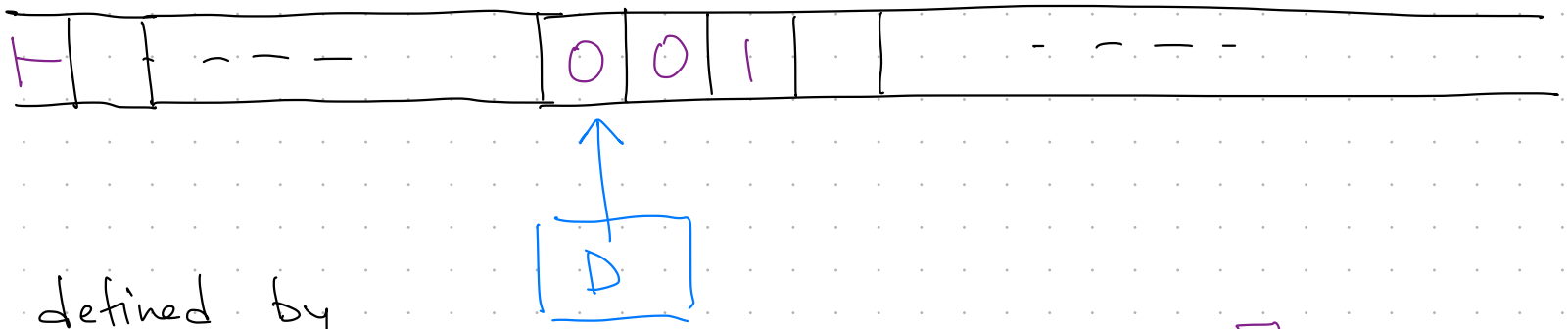
* $Q \equiv$ Finite set of internal states including start, Accept, Reject
* $\delta \equiv$ transition rule

$$\delta(q, \gamma) \longrightarrow q', \gamma', M$$

current state

symbol read

Turing Machines



TM defined by

* $\Sigma \equiv$ Input Alphabet
* $\Gamma \equiv$ Tape Alphabet

$\Sigma \subseteq \Gamma$

$\sqcup \in \Gamma \equiv$ blank

$\vdash \in \Gamma \equiv$ Left end of tape

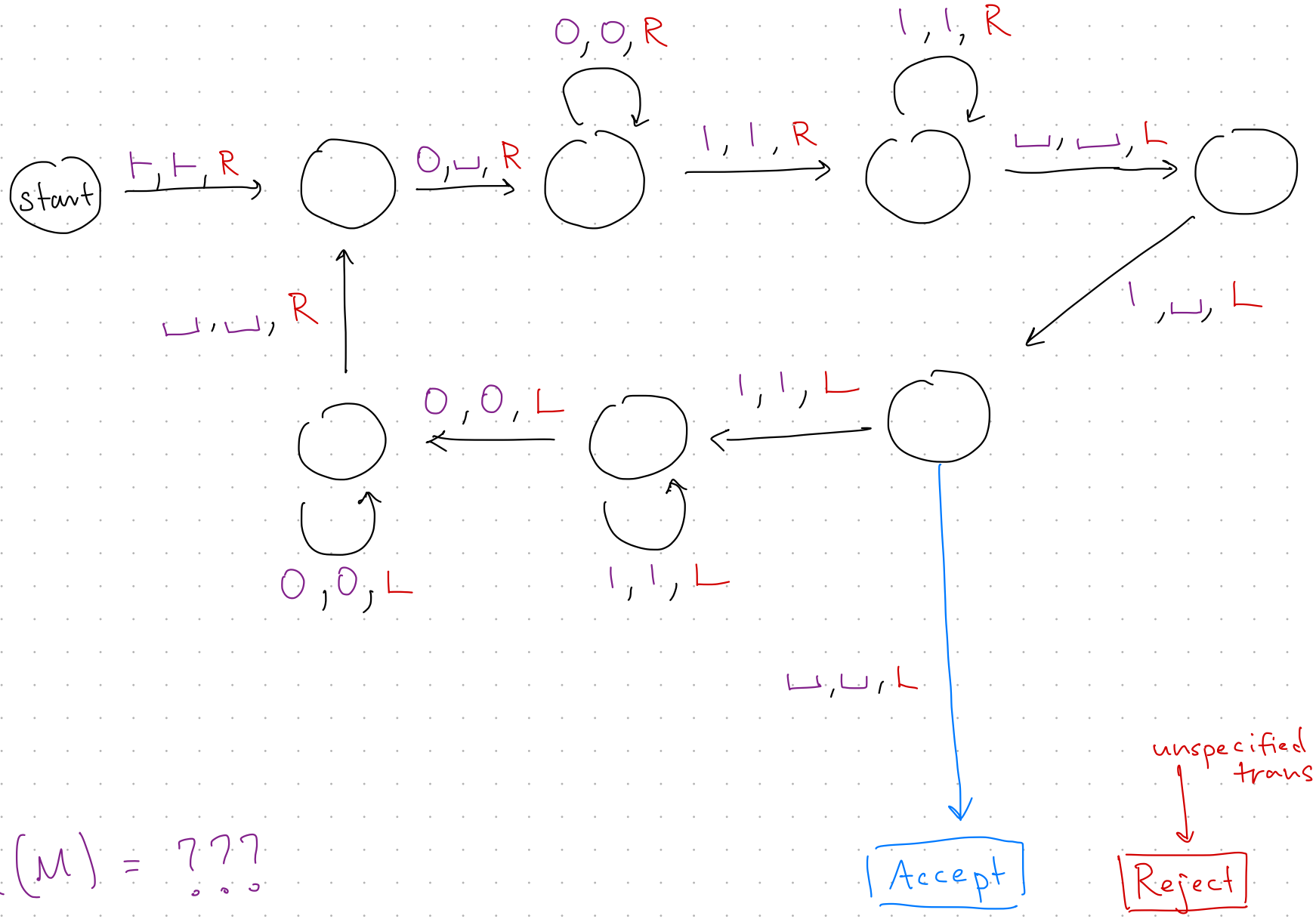
* $Q \equiv$ Finite set of internal states including start, Accept, Reject
* $\delta \equiv$ transition rule

$$\delta(q, \gamma) \longrightarrow q', \gamma', M$$

- q: current state
- γ: symbol read
- q': new state
- γ': symbol to be written
- M: move direction $\in \{L, R\}$

Example TMs

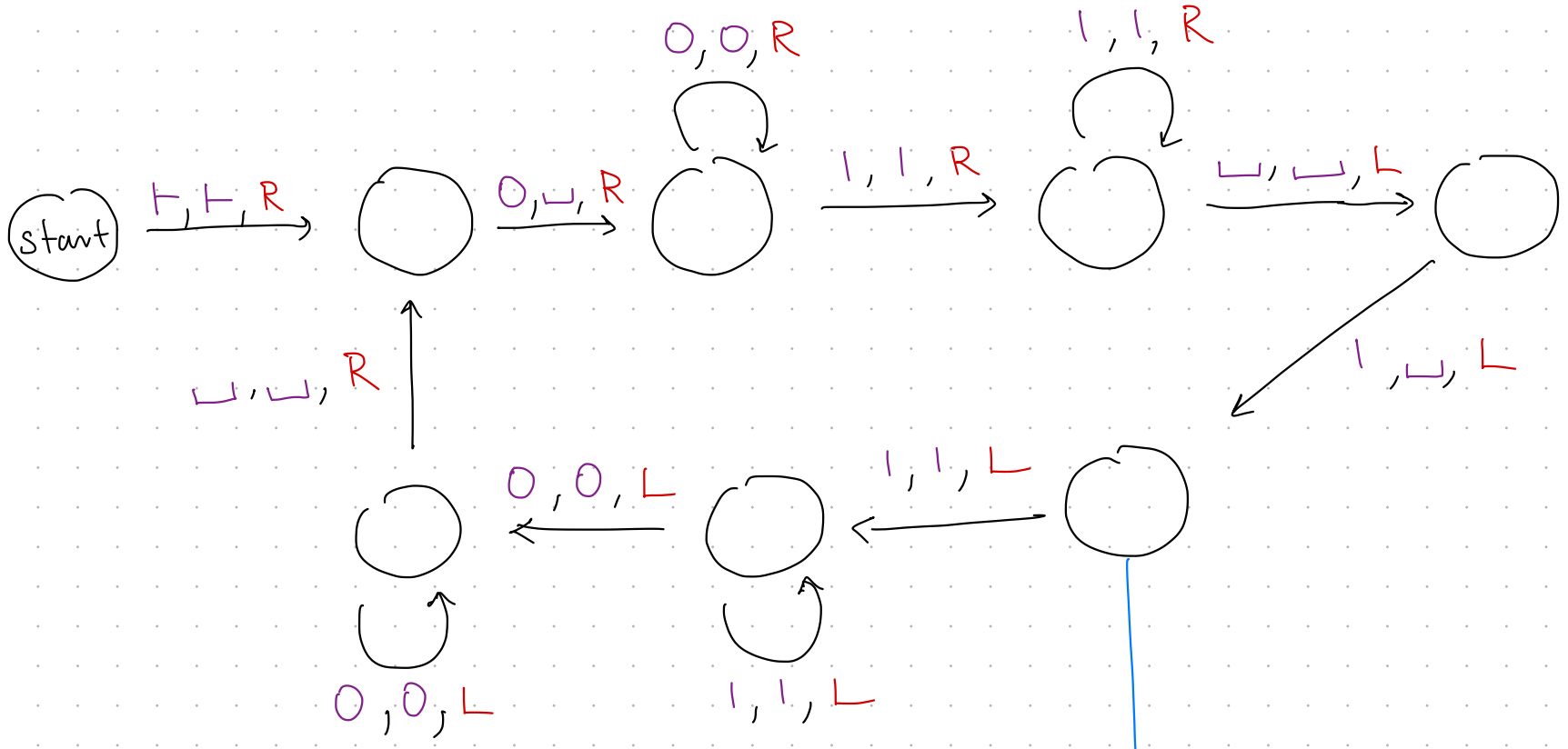
$\Sigma = \{0, 1\}$ $\Gamma = \{0, 1, \sqcup, \vdash\}$



$L(M) = ???$

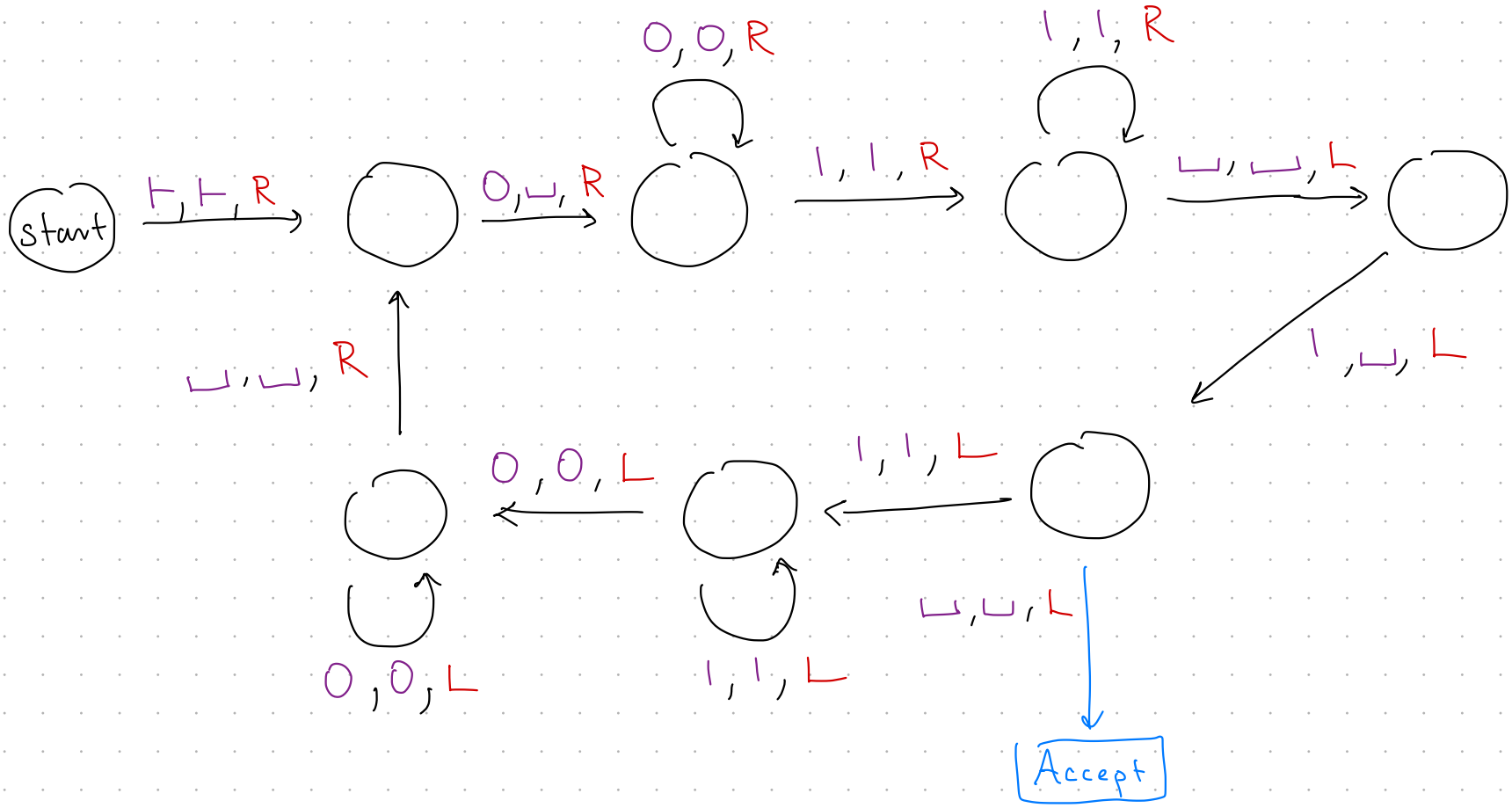
Example TMs

$\Sigma = \{0, 1\}$ $\quad \Gamma = \{0, 1, \sqcup, \vdash\}$

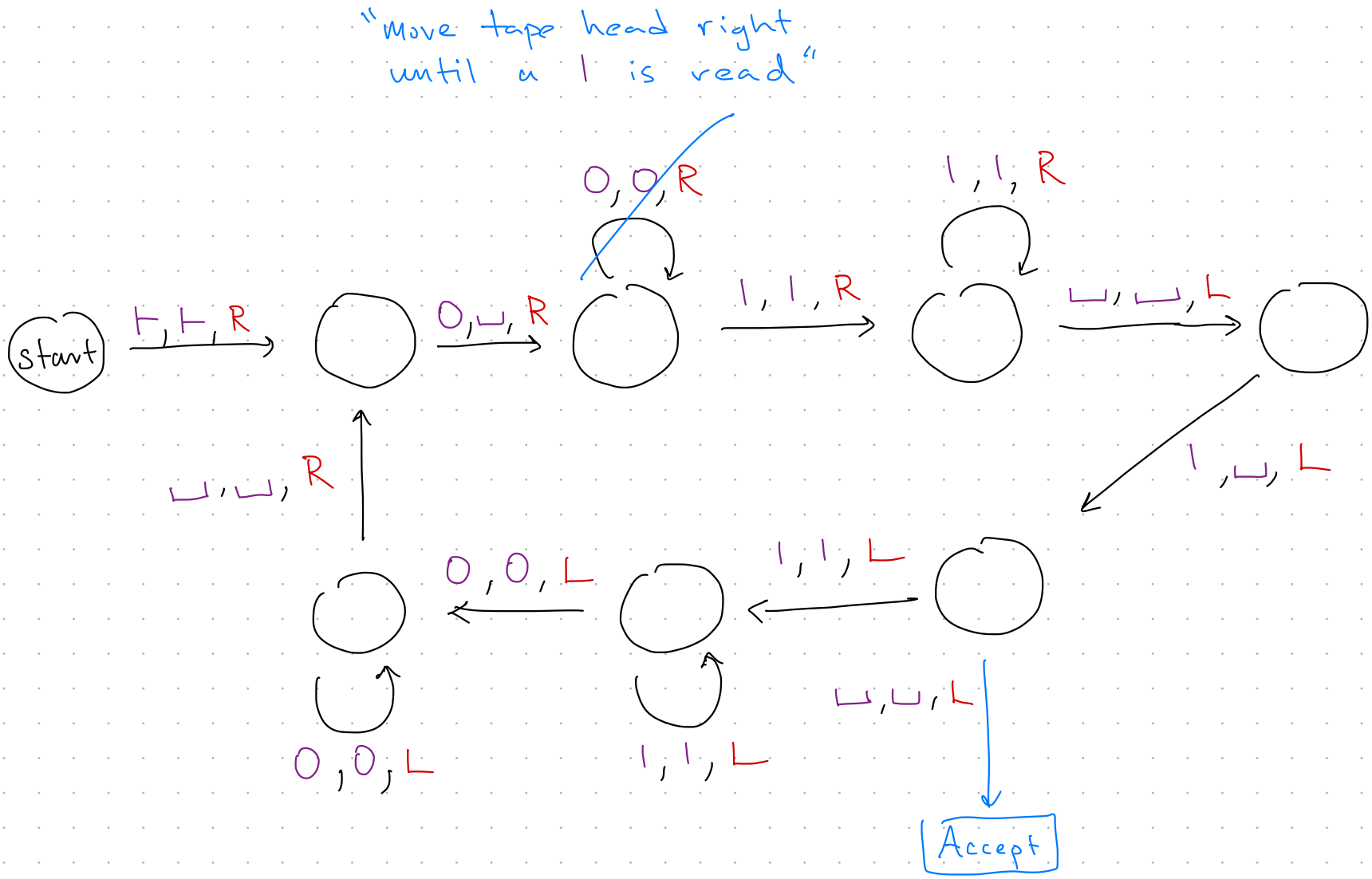


$$L(M) = \{ 0^n 1^n : n \in \mathbb{N} \}$$

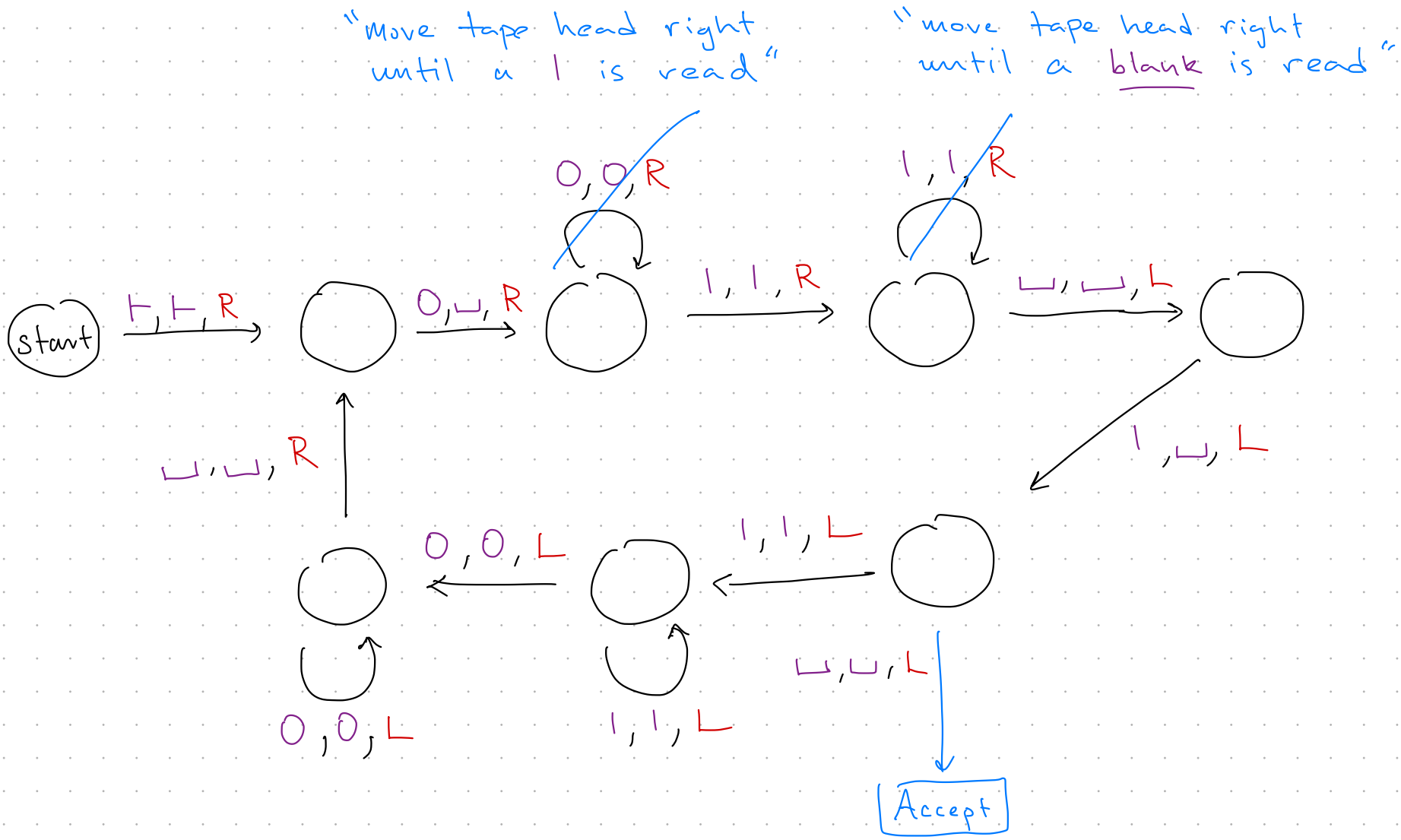
Translating TMs to high-level programs



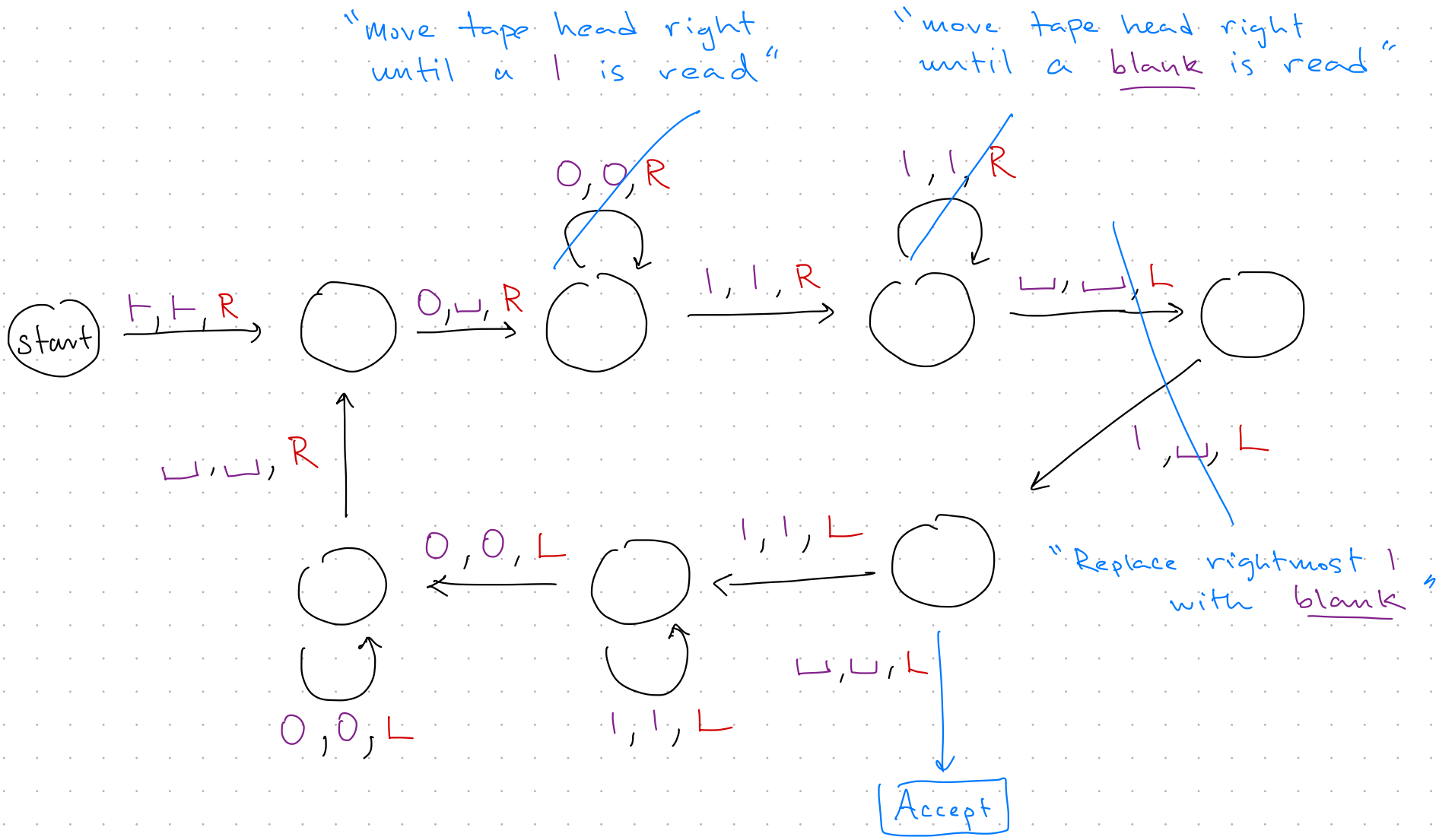
Translating TMs to high-level programs



Translating TMs to high-level programs



Translating TMs to high-level programs



TM "Program"

* Repeat

  - Move R

    | if ¬ Read 0 : Reject
    | Write blank , Move R

    | While Read 0 : Move R
    | While Read 1 : Move R

    } move to end of $0^n 1^m$

    | if ¬Read blank : Reject
    | Move L , Write blank , Move L

  - if Read blank : Accept

    } Equal # of 0 and 1 erased

    | While Read 1 : Move L
    | While Read 0 : Move L

Rules of Thumb for TM Code

* Variables
  - Finite number of variables
  - Taking finite number of values
* Conditionals
  - Finite number of nested conditionals
* Function calls / subroutines
  - Functions must be computable
  - Finite depth call stack.

(otherwise, requires argument that state can be maintained by finite state machine using tape.)