91.304 Foundations of (Theoretical) Computer Science

Chapter 3 Lecture Notes (Section 3.1: Turing Machines)

David Martin
dm@cs.uml.edu

With some modifications by Prof. Karen Daniels, Fall 2013

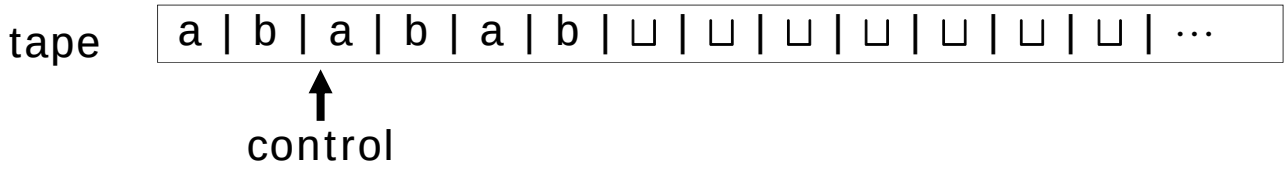
“Manners are not taught in lessons,” said Alice. “Lessons teach you to do sums, and things of that sort.”

“And you do Addition?” the White Queen asked. “What's one and one and one and one and one and one and one and one and one and one?”

“I don't know,” said Alice. “I lost count.”

“She can't do Addition,” the Red Queen interrupted.

Excerpt: *Through the Looking Glass,* Lewis Carroll



Turing machine syntax

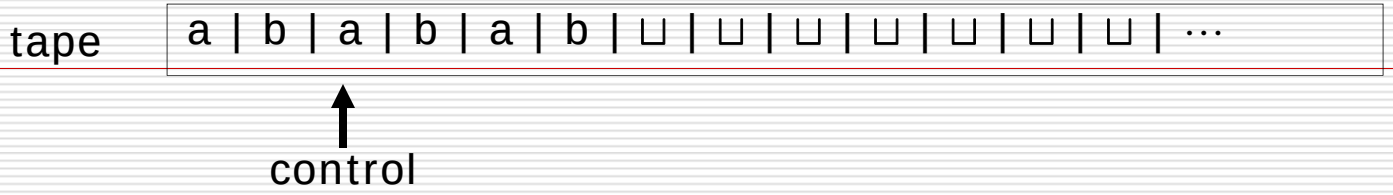
- **Definition** A Turing Machine is an automaton $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{acc},q_{rej})$ where

1. Q is a finite set of states
2. Σ is an input alphabet that does **not** include " ⊔", the special blank character
3. Γ is a tape alphabet satisfying
   1. $\sqcup \in \Gamma$
   2. $\Sigma \subset \Gamma$
4. $\delta: Q\times\Gamma \rightarrow Q\times\Gamma\times\{L,R\}$ is the transition function
   1. "staying put" is not an option, except at left end of tape
5. q_0 is the initial state
6. q_{acc} is the single accepting state
7. q_{rej} is the single rejecting state

Alan Turing proposed the Turing Machine in 1936!

Differences from Finite Automata

- Turing machine
  - Can both read from and write onto tape.
    - No LIFO access restriction as in PDA's stack
  - Read/write head (control) can move both left and right.
  - Tape is infinite.
  - Special states for rejecting and accepting take effect immediately.
  - In some cases machine can fail to halt...



Differences in input mechanism

- A TM has a "tape head" that points to exactly one cell on its tape, which extends infinitely to the right
  - At each transition, the TM looks at the current state and the current cell, and decides what new state to move to, what to write on the current cell, and whether to move one cell to the left or one cell to the right (or stay put at left end of tape)
  - Hence the transition function $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$
- Each tape cell initially contains the blank character ␣
- Our previous automata (DFAs, NFAs, PDAs) all had a separate read-only input stream
- But in a TM, the input is given all at once and just written onto the left end of the tape — overwriting the blanks there

Turing machine computation

- We define a set of *instantaneous descriptions* (IDs or *configurations*) and then show what memory-state snapshots may follow each other, according to the program M.
- First, the snapshots: in a TM, ID(M) = Γ^* Q Γ^*
  - Each element of this set represents the entire tape contents, the current state, and the location of the tape head
  - In example below, the ID is ab q_7a bab␣␣···
  - So the character to the *right* of the state name is the "current" character
  - The tape always has infinitely many blanks on the right; we can write them or omit them as we please

Turing machine computation

- Two IDs are related to each other (by $\vdash$) if one can lead to the other (via 1 transition) according to the δ function
- So we look at all of the things that δ can say, starting with right moves:
  - Suppose $\delta(q,b) = (t,c,\mathbf{R})$ where
    - $q \in Q - \{q_{acc}, q_{rej}\}$ and $b \in \Gamma$ (states in green)
    - $t \in Q$ and $c \in \Gamma$
    - **R** means "right move" (after reading/writing)
  - Then u **q**b v $\vdash$ u **c**t v

    where $u,v \in \Gamma^*$ are undisturbed, the state has changed from q to t, the tape cell has changed from b to c, and the head has moved one character to the right (over the now-changed character)

Turing machine computation

- Left moves
  - Suppose $\delta(q,b) = (t,c,\mathbf{L})$ where
    - $q \in Q - \{q_{acc}, q_{rej}\}$ and $b \in \Gamma$ (states in green)
    - $t \in Q$ and $c \in \Gamma$
  - Then ua **q**b v $\vdash$ u **t**a**c** v
    where $u,v \in \Gamma^*$ and $a \in \Gamma$ are undisturbed, the state has changed from q to t, the tape cell has changed from b to c, and the head has moved one character to the left
  - This says that one ID can lead to another ID when δ says to move left *and there is a character* $a \in \Gamma$ to the left. What if there is no such character?

Turing machine computation

- Left moves at left edge of tape
  - Suppose $\delta(q,b) = (t,c,\mathbf{L})$ where
    - $q \in Q - \{q_{acc}, q_{rej}\}$ and $b \in \Gamma$ (states in green)
    - $t \in Q$ and $c \in \Gamma$
  - Then **q**b v $\vdash$ **t**c v

    where $v \in \Gamma^*$ is undisturbed, the state has changed from q to t, the tape cell has changed from **b** to **c**
  - Where does this put the tape head in this case?
- Note we have **not** explicitly covered the case where $\delta(q,b) = (t,c,L)$ and $q \in \{q_{acc}, q_{rej}\}$
  - Or when we move R instead of L
  - Conclusion: well, if the current ID is u **q**b v and $q \in \{q_{acc}, q_{rej}\}$, then no "next ID" is possible. We say that the **TM halts**

Some Ways to Describe Turing Machine Computation

1. Implementation-level description (high-level)
2. *Instantaneous descriptions* (IDs or configurations) specifying snapshots of tape and read-write head position as computation progresses.
3. Formal description (7-tuple)
4. Detailed state diagram.

- We'll discuss all 4 ways using Turing machine M_1 in textbook (p. 138, 139, 145) for language:

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

- We'll also discuss Turing machine M_2 in textbook (p. 143, 144) for language: $A = \{0^{2^n} \mid n \geq 0\}$

Implementation-Level Description

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

M_1 = "On input string w:

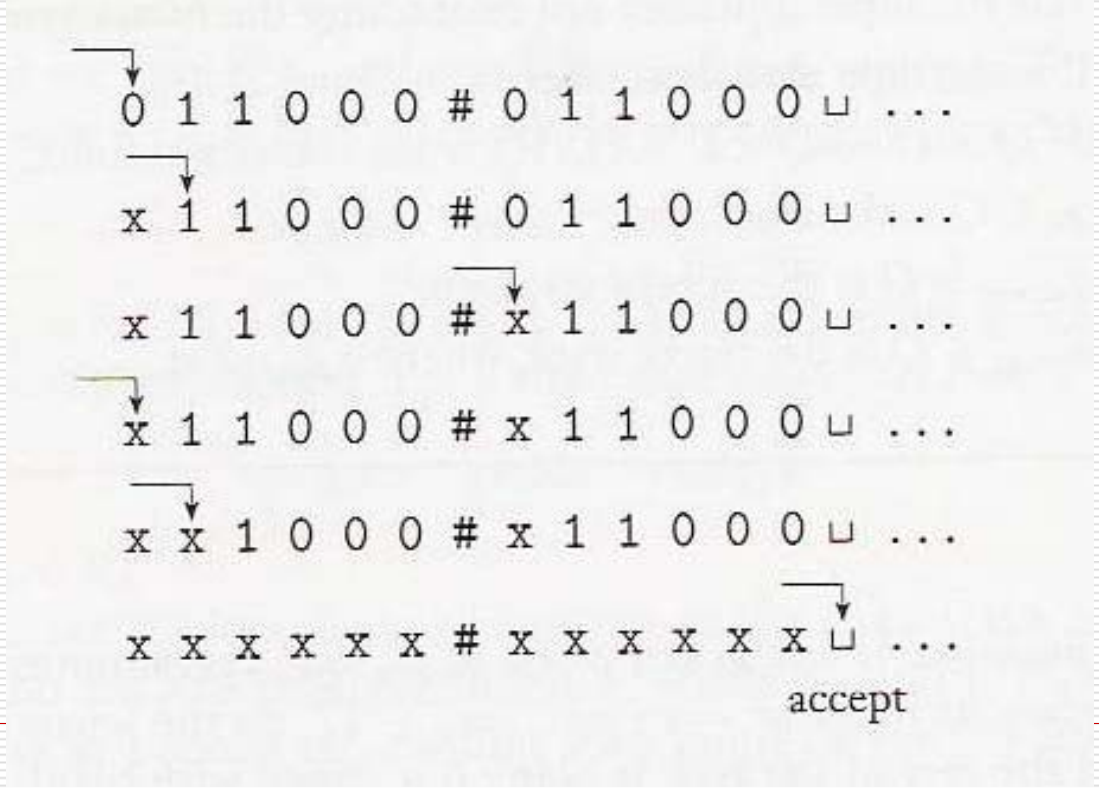
1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."

Small Examples:

- Accepting input: 101#101␣␣…
- Rejecting input: 0101#1000␣␣…

Instantaneous Descriptions (Snapshots) $B = \{w\#w \mid w \in \{0,1\}^*\}$

Sample Input: 011000#011000



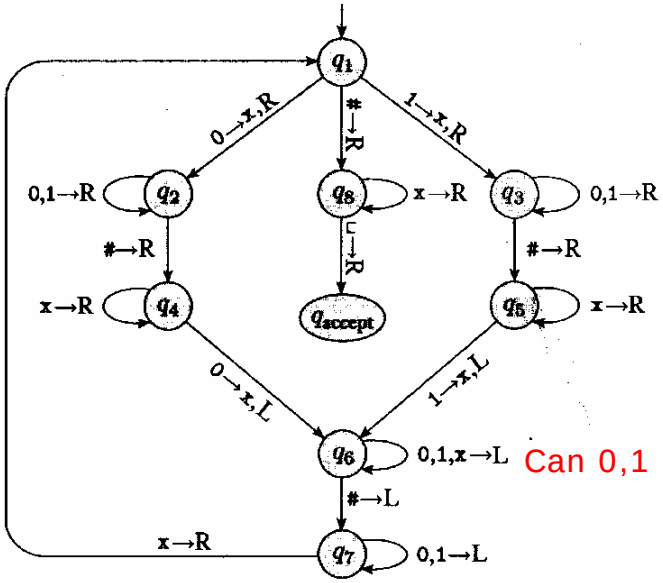
Formal Description and Detailed State Diagram $B = \{w\#w \mid w \in \{0,1\}^*\}$

EXAMPLE 3.9

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described (page 139) for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_{14}, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,\text{x},\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.

8



Can 0,1 happen here?

FIGURE 3.10
State diagram for Turing machine M_1

Detailed State Diagram

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

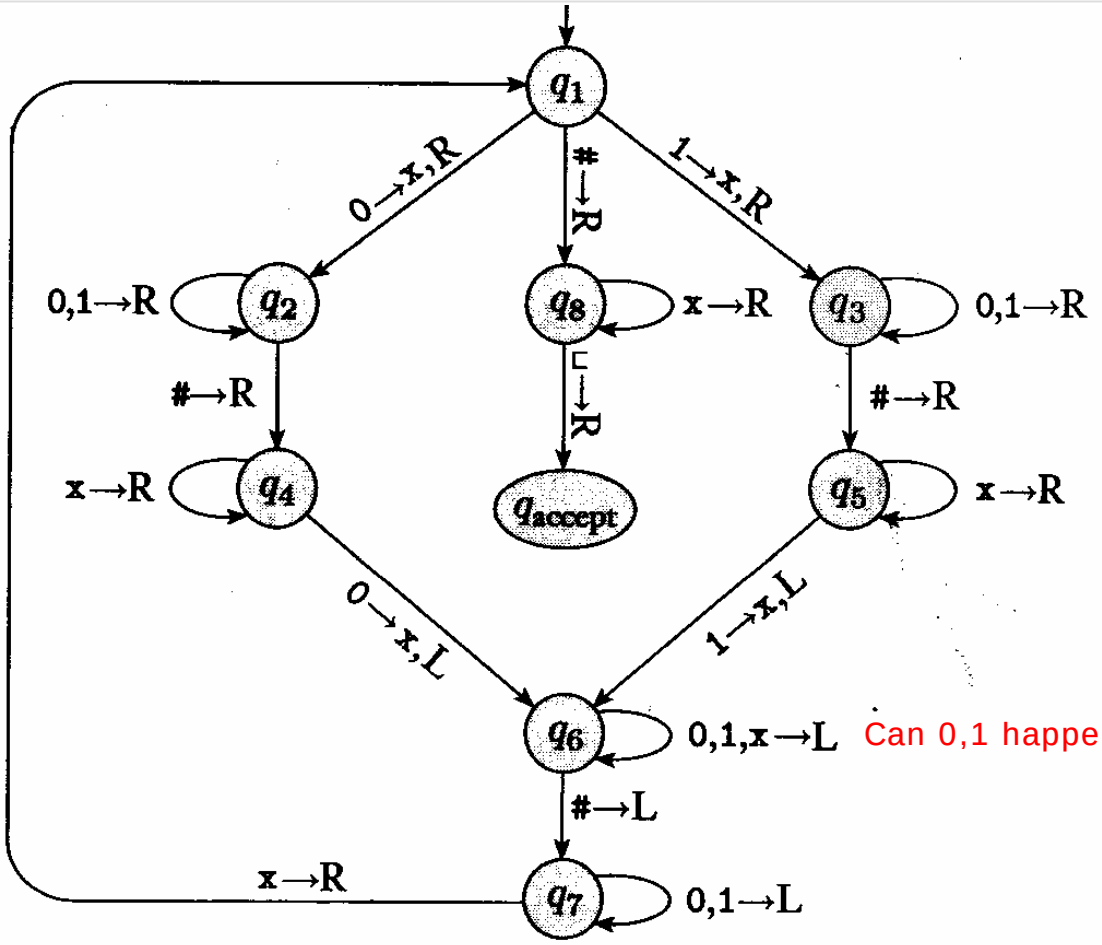


FIGURE 3.10
State diagram for Turing machine M_1

Can 0,1 happen here?

Small Examples:

- Accepting input: 101#101␣␣…
- Rejecting input: 0101#1000␣␣…

Implementation-Level Description

$$A = \{0^{2^n} \mid n \geq 0\}$$

EXAMPLE 3.7

Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

M_2 = "On input string w:

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

Each iteration of stage 1 cuts the number of 0s in half. As the machine sweeps across the tape in stage 1, it keeps track of whether the number of 0s seen is even or odd. If that number is odd and greater than 1, the original number of 0s in the input could not have been a power of 2. Therefore the machine rejects in this instance. However, if the number of 0s seen is 1, the original number must have been a power of 2. So in this case the machine accepts.

Formal Description (7-tuple)

$$A = \{0^{2^n} \mid n \geq 0\}$$

Now we give the formal description of $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0\}$, and
- $\Gamma = \{0, x, \sqcup\}$.
- We describe δ with a state diagram (see Figure 3.8).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.

Detailed State Diagram

$$A = \{0^{2^n} \mid n \geq 0\}$$

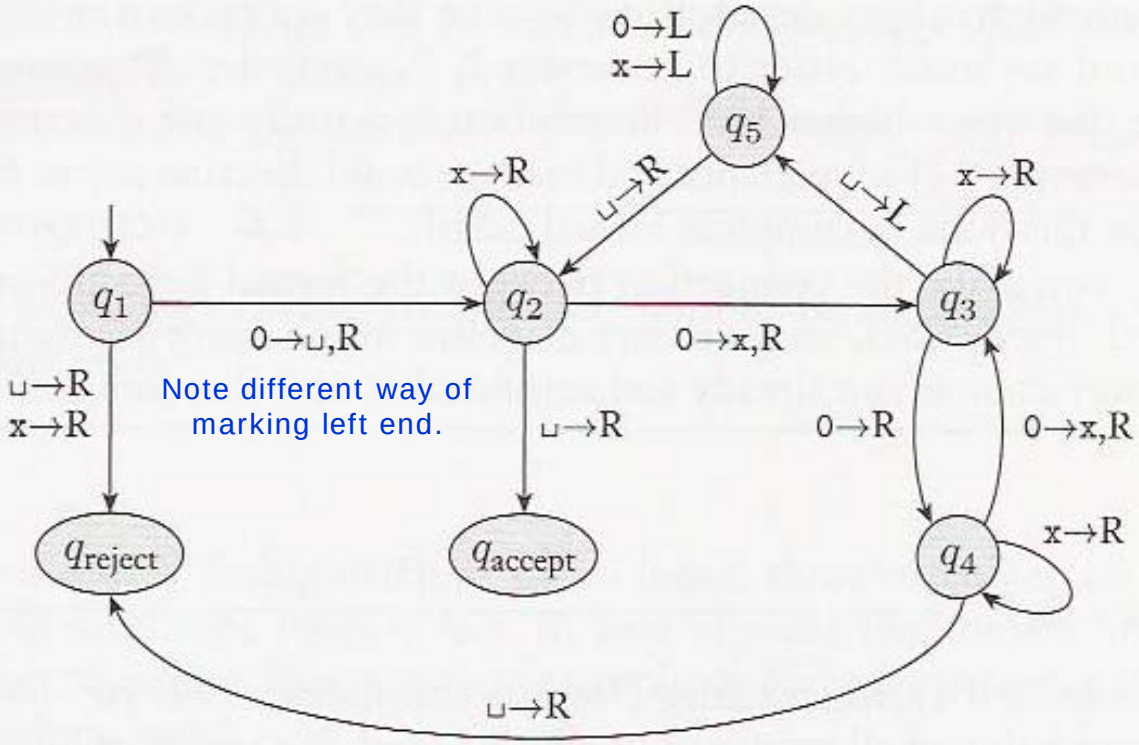


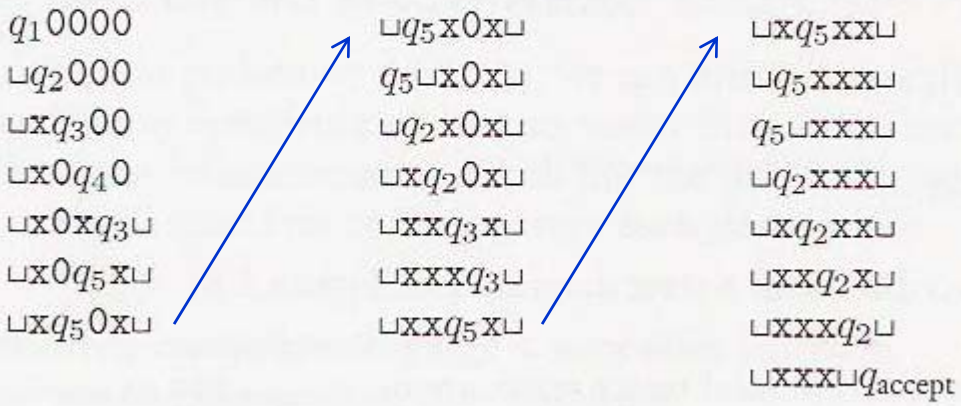
FIGURE 3.8
State diagram for Turing machine M_2

Instantaneous Descriptions (IDs or configurations)

$A = \{0^{2^n} \mid n \geq 0\}$

Sample Input: 0000

Next we give a sample run of this machine on input 0000. The starting configuration is $q_1 0000$. The sequence of configurations the machine enters appears as follows; read down the columns and left to right.



Detailed State Diagram

$$A = \{0^{2^n} \mid n \geq 0\}$$

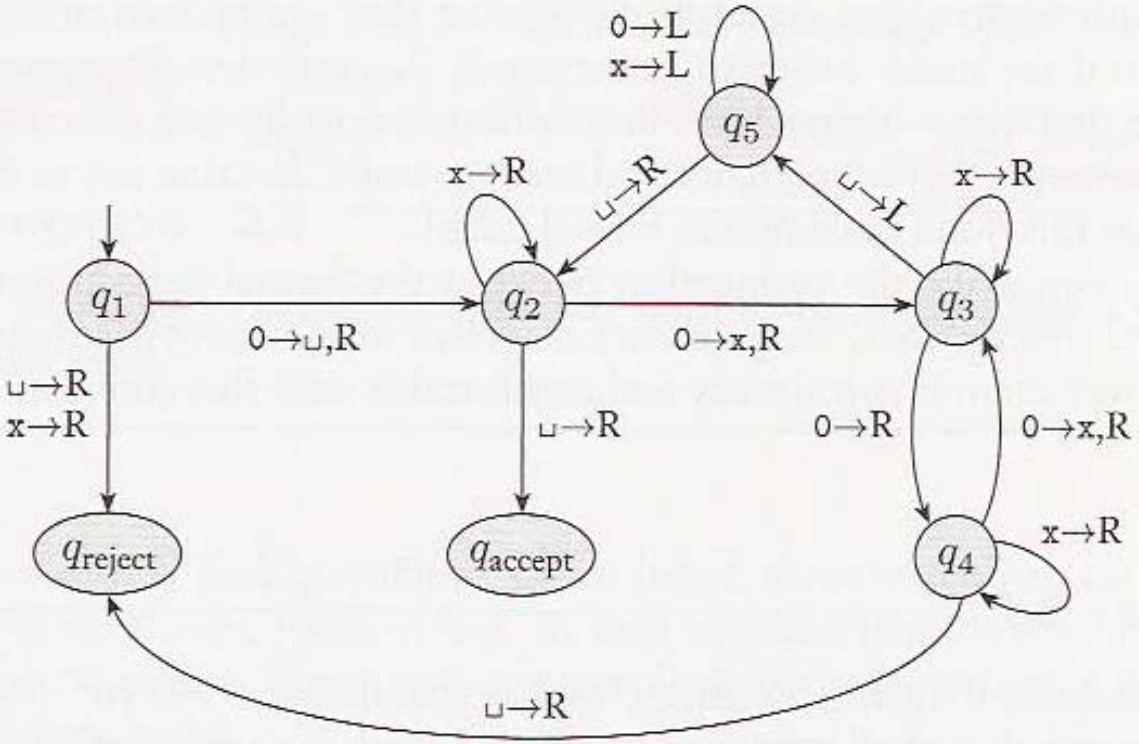


FIGURE 3.8
State diagram for Turing machine M_2

$q_1 0000$
␣$q_2 000$
␣x$q_3 00$
␣x0$q_4 0$
␣x0xq_3␣
␣x0q_5x␣
␣x$q_5$0x␣
␣q_5x0x␣
q_5␣x0x␣
␣q_2x0x␣
␣x$q_2$0x␣
␣xxq_3x␣
␣xxxq_3␣
␣xxq_5x␣
␣x$q_5$xx␣
␣$q_5$xxx␣
q_5␣xxx␣
␣$q_2$xxx␣
␣x$q_2$xx␣
␣xxq_2x␣
␣xxxq_2␣
␣xxx␣q_{accept}

More Examples...

- See Textbook Examples:
  - Example 3.11

$$C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$$

    - Subtlety on detecting left end of tape.
  - Example 3.12 (element distinctness)

$$E = \{\# x_1 \# x_2 \# \cdots \# x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

Language recognized by TM

- Finally, we let $\vdash^*$ be the transitive, reflexive closure of $\vdash$. So if α and β are IDs, the statement $\alpha \vdash^* \beta$ means "the TM can go from α to β in 0 or more steps"
- The **language recognized** by M is $L(M) = \{ x \in \Sigma^* \mid q_0 x \vdash^* u\, q_{acc}\, v$ for some $u,v \in \Gamma^* \}$ **(strings that are accepted by M)**
- Translation?
- Note $x \in \Sigma^*$, not Γ^*

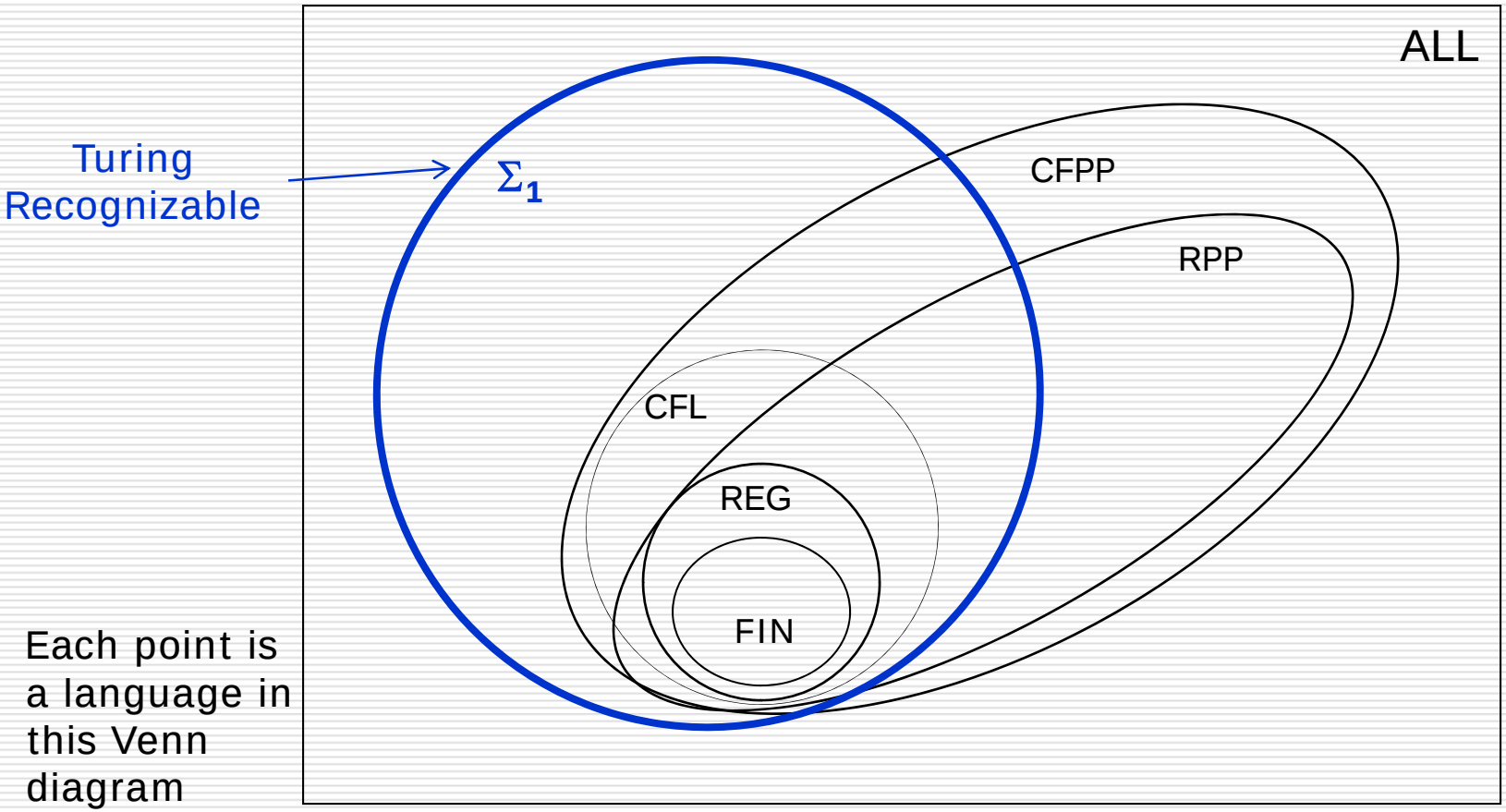
TM language classes

- **Definition** A language L is **Turing-recognizable** if there exists a TM M such that L = L(M).
  - Synonym: L is recursively enumerable, abbreviated "r.e." (see Section 3.2)
- **Definition** The class of all Turing-recognizable languages is

  $\Sigma_1 = \{ L \subseteq \Sigma^* \mid L \text{ is Turing-recognizable} \}$

  - The textbook does not assign a name like this; it just says "class of TM-recognizable langs"
  - **Beware:** The class Σ_1 is **not** an alphabet like Σ
  - The naming is unfortunate but better than some of the alternatives

Turing-Recognizable Languages



Deciders

- We've seen that when you start a TM with an input x, it can do three distinct things:
  - Accept x
  - Reject x
  - Run forever without accepting or rejecting x
    - We call this "looping" -- meaning that the TM runs *forever.* (The "loop" might not be so simple, the point is it runs forever.)
- Some TMs always accept or reject and never loop on any input whatsoever. You could easily write an example of one. A TM with this property is called a **decider**.
  - A decider **always halts** on every input

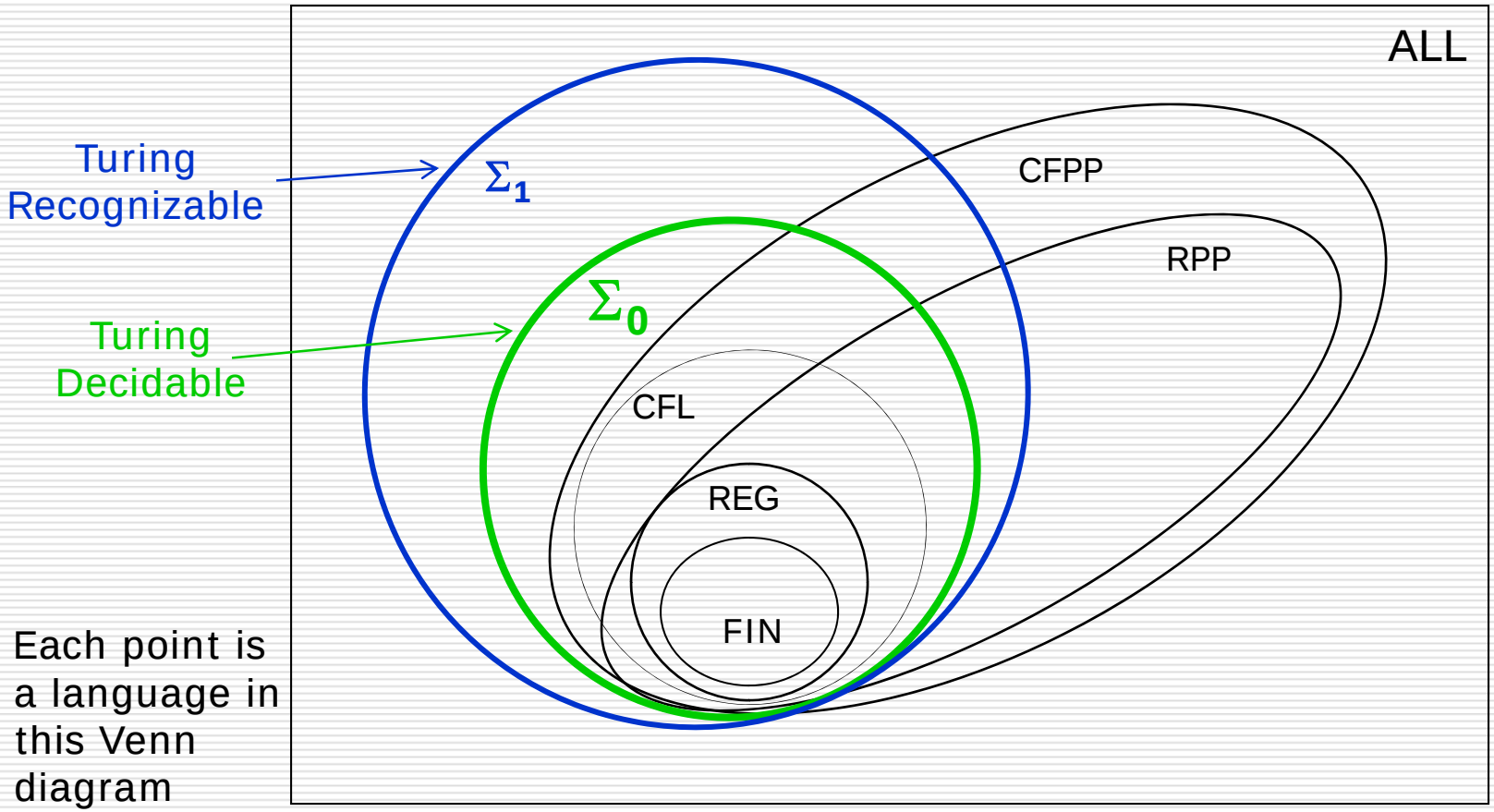
Decidable languages

- **Definition** A language is **decidable** if there exists a **decider** TM M such that L = L(M)
  - Synonyms: L is "computable" and "recursive"
  - It is in general **not easy** to tell if a language is decidable or not
- **Definition** The class of all Turing-decidable languages is

$$\Sigma_0 = \{ L \subseteq \Sigma^* \mid L \text{ is Turing-decidable} \}$$

  - Note Σ_0 (decidable) versus Σ_1 (recognizable) versus Σ (alphabet)

Turing-Decidable Languages

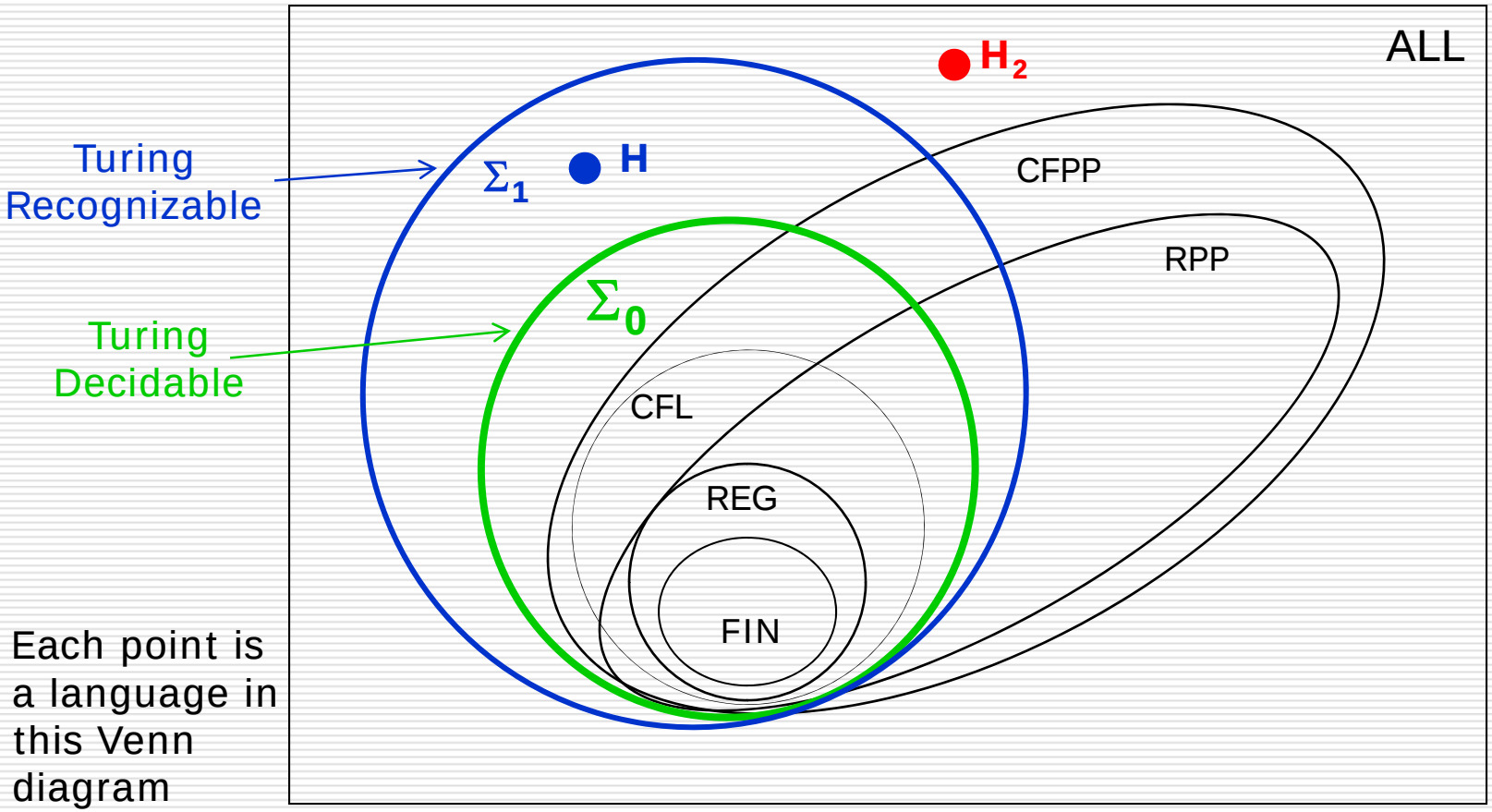


Each point is a language in this Venn diagram

Decidable versus recognizable

- **Fact** (obvious) $\Sigma_0 \subseteq \Sigma_1$
  - Every decider is automatically a recognizer too
- **Fact** (not at all obvious) $\Sigma_0 \neq \Sigma_1$
  - This means that there exists some language $H \in \Sigma_1 - \Sigma_0$
    - H is a language that can be recognized by some TM, but can't be recognized by any TM that always halts!
- **Fact** (not at all obvious) $\Sigma_1 \neq$ ALL
  - This means that there exists some language $H_2 \in$ ALL $- \Sigma_1$
    - H_2 is a language that can't even be recognized by any TM

Ultimately…


Turing Recognizable
Turing Decidable
Each point is a language in this Venn diagram
ALL
H_2
H
Σ_1
Σ_0
CFPP
RPP
CFL
REG
FIN

Reminder

- The *decidable* languages: Σ_0
- The *recognizable* languages: Σ_1

Closure properties of Σ_0 and Σ_1

- Σ_1 is closed under $\cup, \cap, \cdot, *$, reversal
  - Proofs for $\cup$ and $\cap$ are similar to the NFA constructions we used, if you use a 2-tape TM (section 3.2)
  - Proof for reversal is also easy with a 2-tape TM (section 3.2)
  - $\cdot$ and $*$ are somewhat harder
  - **Not** closed under complement
- Σ_0 is closed under all of these operations and **complement** as well

Preview: a non-recognizable L

- This all means that some L exists that is not recognized by any TM
  - What does it look like?
  - Is it important?

  YES, because of Church-Turing Thesis
  - Intuitive notion of *algorithms* = Turing machine *algorithms*
  - To be defined and discussed in Section 3.3