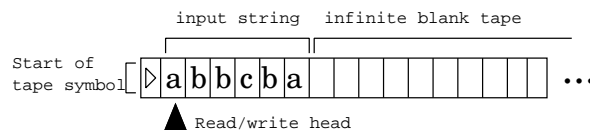


11-0: Turing Machines

- Machines so far (DFA, PDA) read input only once
- Next: Turing Machines
  - Can back up over the input
  - Can overwrite the input

11-1: Turing Machines



- Input string is written on a tape:
- At each step, machine reads a symbol, and then either
  - Writes a new symbol
  - Moves read/write head to right
  - Moves read/write head to left

11-2: Turing Machines

- A Turing Machine  $M = (K, \Sigma, \delta, s, H)$ 
  - $K$  is a set of states
  - $\Sigma$  is the tape alphabet
  - $s \in K$  is the start state
  - $H \subset K$  are “Halting states” – y for accept, and n for reject
  - $\delta : (K - H) \times \Sigma \mapsto K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$

11-3: Turing Machines

- TM for  $L = \{0^n 1^n : n > 0\}$
- Idea:
  - Mark off first 0, go to end of string and mark off last one (ensuring that the string starts with 0 and ends with 1)
  - Repeat until all characters have been morked off
  - Make sure # of 0s match the # if 1s.

11-4: Turing Machines

TM for  $L = \{0^n 1^n : n > 0\}$

	0	1	$\sqcup$	$X$	$Z$
$q_0$	$(q_1, X)$	$(n, 1)$	$(n, \sqcup)$	$(n, X)$	$(y, Z)$
$q_1$	$(q_1, \rightarrow)$	$(q_1, \rightarrow)$	$(q_2, \leftarrow)$	$(q_1, \rightarrow)$	$(q_2, \leftarrow)$
$q_2$	$(n, 0)$	$(q_3, Z)$	$(n, \sqcup)$	$(n, X)$	$(n, Z)$
$q_3$	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$	$(n, \sqcup)$	$(q_0, \rightarrow)$	$(q_3, \leftarrow)$

TM for  $L = \{0^n 1^n : n > 0\}$

11-5: Turing Machines

	0	1	□	X	Z
$q_0$	$(q_1, X)$	no	no	no	yes
$q_1$	$(q_1, \rightarrow)$	$(q_1, \rightarrow)$	$(q_2, \leftarrow)$	$(q_1, \rightarrow)$	$(q_2, \leftarrow)$
$q_2$	no	$(q_3, Z)$	no	no	no
$q_3$	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$	no	$(q_0, \rightarrow)$	$(q_3, \leftarrow)$

11-6: Turing Machines

TM for  $L = \{0^n 1^n 2^n : n > 0\}$

11-7: Turing Machines

TM for  $L = \{0^n 1^n 2^n : n > 0\}$

	0	1	2	□	X	Y	Z
$q_0$	$(q_1, X)$	no	no	no	no	$(q_4, \rightarrow)$	no
$q_1$	$(q_1, \rightarrow)$	$(q_2, Y)$	no	no	$(q_1, \rightarrow)$	$(q_1, \rightarrow)$	no
$q_2$	no	$(q_2, \rightarrow)$	$(q_3, Z)$	no	no	$(q_2, \rightarrow)$	$(q_2, \rightarrow)$
$q_3$	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$	no	$(q_0, \rightarrow)$	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$
$q_4$	no	no	no	yes	$(q_4, \rightarrow)$	$(q_4, \rightarrow)$	$(q_4, \rightarrow)$

11-8: Turing Machines

TM for  $L = \{ww^R : w \in \{a, b\}^*\}$

11-9: Turing Machines

TM for  $L = \{ww^R : w \in \{a, b\}^*\}$

	a	b	□	X	Z
$q_0$	$(q_1, X)$	$(q_4, X)$	yes	no	yes
$q_1$	$(q_1, \rightarrow)$	$(q_1, \rightarrow)$	$(q_2, \leftarrow)$	$(q_1, \rightarrow)$	$(q_2, \leftarrow)$
$q_2$	$(q_3, Z)$	no	no	no	no
$q_3$	$(q_3, \leftarrow)$	$(q_3, \leftarrow)$	no	$(q_0, \rightarrow)$	$(q_3, \leftarrow)$
$q_4$	$(q_4, \rightarrow)$	$(q_4, \rightarrow)$	$(q_5, \leftarrow)$	$(q_4, \rightarrow)$	$(q_5, \leftarrow)$
$q_2$	no	$(q_3, Z)$	no	no	no

11-10: Turing Machine Diagrams

- “Table Notation” for Turing Machines can be difficult to read
- Define small machines, use them to build larger machines
  - *Not* changing the definition of a TM, just using a more convenient notation

11-11: Turing Machine Diagrams

- Writing machines:  $M_a = (K, \Sigma, \delta, s, H)$ 
  - $K = \{s, h\}$
  - $s = s$
  - $H = \{h\}$
  - $\delta = \{((s, b), (h, a)) : \forall b \in \Sigma\}$
- Writes an 'a' on the tape, and then halts.

11-12: Turing Machine Diagrams

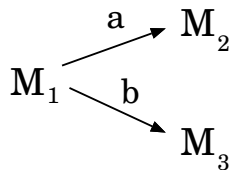
- Moving Machines:  $M_{\leftarrow} = (K, \Sigma, \delta, s, H)$ 
  - $K = \{s, h\}$
  - $s = s$
  - $H = \{h\}$

- $\delta = \{((s, b), (h, \leftarrow)) : \forall b \in \Sigma\}$

- Moves the head to the left, and then halts.

11-13: Turing Machine Diagrams

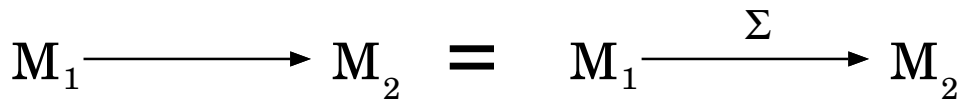
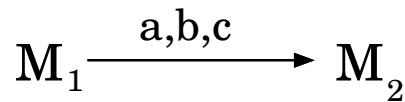
- Connecting Diagrams:



- Run  $M_1$  until it halts, and then examine the symbol under the read/write head
  - If the symbol is an 'a', execute  $M_2$  until it halts
  - If the symbol is a 'b', execute  $M_3$  until it halts

11-14: Turing Machine Diagrams

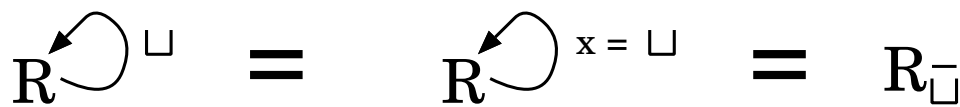
- Connecting Diagrams:



11-15: Turing Machine Diagrams

- Shorthand:

- $M_a = a$
- $M_{\leftarrow} = L$
- $M_{\rightarrow} = R$
- $R \rightarrow R \rightarrow a \rightarrow = R R a = R^2 a$



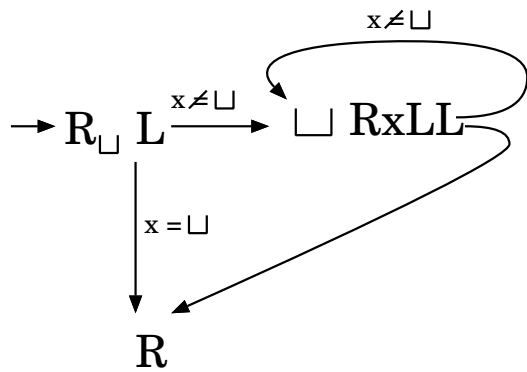
•

11-16: Turing Machine Diagrams

- Shift-right machine
  - Convert  $\triangleright \sqcup w$  to  $\triangleright \sqcup \sqcup w$

11-17: Turing Machine Diagrams

- Shift-right machine
  - Convert  $\triangleright \sqcup w$  to  $\triangleright \sqcup \sqcup w$

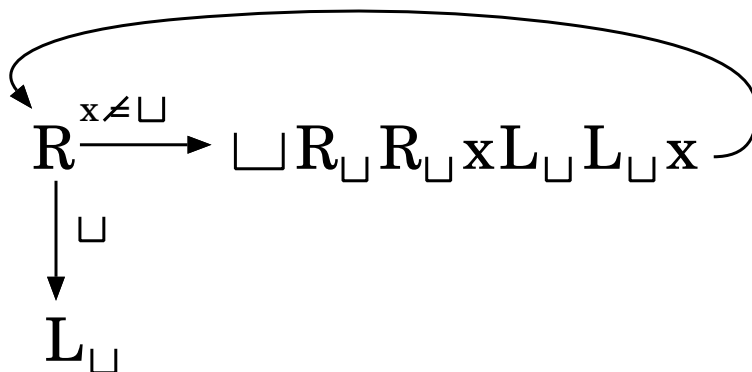


11-18: Turing Machine Diagrams

- Copy machine
  - Convert  $\triangleright \sqcup w$  to  $\triangleright \sqcup w \sqcup w$

11-19: Turing Machine Diagrams

- Copy machine
  - Convert  $\triangleright \sqcup w$  to  $\triangleright \sqcup w \sqcup w$

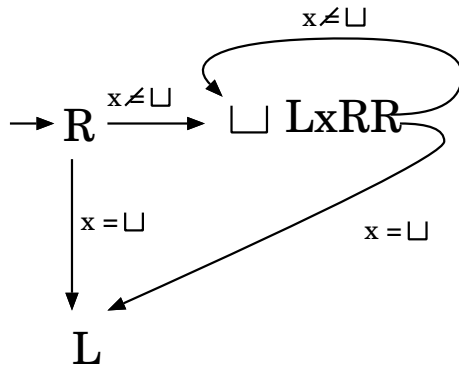


11-20: Turing Machine Diagrams

- Shift-left machine
  - Convert  $\sqcup w$  to  $w \sqcup$

11-21: Turing Machine Diagrams

- Shift-left machine
  - Convert  $\sqcup w$  to  $w\sqcup$



11-22: Turing Machine Diagrams

- Copy machine (part II)
  - Convert  $\triangleright\sqcup w$  to  $\triangleright\sqcup ww$
  - (Using other machines .... )

11-23: Turing Machine Diagrams

- Copy machine (part II)
  - Convert  $\triangleright\sqcup w$  to  $\triangleright\sqcup ww$

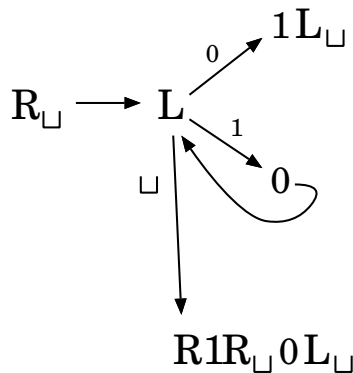
$$M_{\text{COPY}} \longrightarrow R_{\sqcup} \longrightarrow M_{\text{LEFT-SHIFT}} \longrightarrow L_{\sqcup}$$

11-24: Turing Machine Diagrams

- Successor
  - Convert  $\triangleright\sqcup w$  (where  $w$  is the binary representation of an integer) to  $\triangleright\sqcup v$  (where  $v$  is the binary representation of  $(w + 1)$ )
    - $\triangleright\sqcup 11011 \Rightarrow \triangleright\sqcup 11100$
    - $\triangleright\sqcup 11111 \Rightarrow \triangleright\sqcup 10000$

11-25: Turing Machine Diagrams

- Successor
  - Convert  $\triangleright\sqcup w$  (where  $w$  is the binary representation of an integer) to  $\triangleright\sqcup v$  (where  $v$  is the binary representation of  $(w + 1)$ )

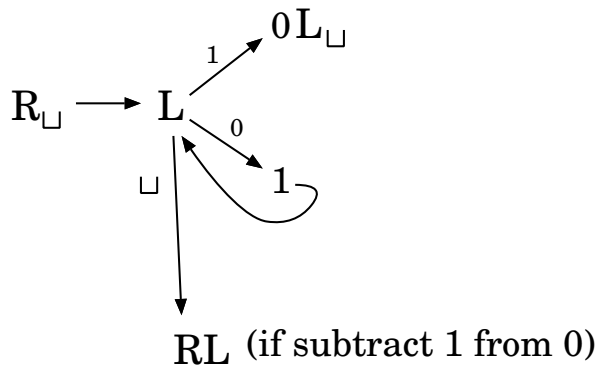


11-26: Turing Machine Diagrams

- Predecessor
  - Convert  $\triangleright \square w$  (where  $w$  is the binary representation of an integer) to  $\triangleright \square v$  (where  $v$  is the binary representation of  $(w - 1)$ )
    - $\triangleright \square 11100 \Rightarrow \triangleright \square 11011$
    - $\triangleright \square 11111 \Rightarrow \triangleright \square 10000$

11-27: Turing Machine Diagrams

- Predecessor
  - Convert  $\triangleright \square w$  (where  $w$  is the binary representation of an integer) to  $\triangleright \square v$  (where  $v$  is the binary representation of  $(w - 1)$ )



11-28: Turing Machine Diagrams

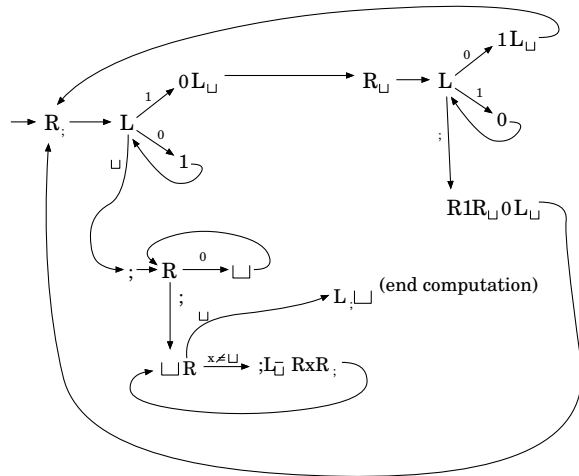
- Add
  - Convert  $\triangleright \square w; v$  to  $\triangleright \square w + v$
  - (first, convert to  $\triangleright \square 0 \dots 0; w + v$ )

11-29: Turing Machine Diagrams

- Add
  - Convert  $\triangleright \square w; v$  to  $\triangleright \square w + v$



- Add: Convert  $\triangleright \sqcup w; v$  to  $\triangleright \sqcup w + v$

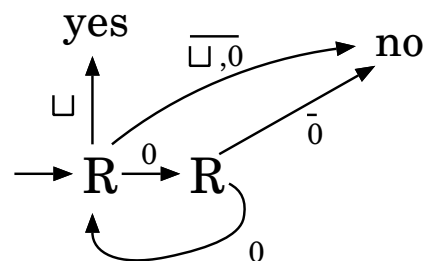


11-34: Turing Machine Diagrams

- We can add “yes” and “no” machines to our diagrams for machines that accept and reject strings
- Diagram for a Turing Machine that accepts the language  $L = \{0^{2^n} : n \geq 0\}$

11-35: Turing Machine Diagrams

- We can add “yes” and “no” machines to our diagrams for machines that accept and reject strings
- Diagram for a Turing Machine that accepts the language  $L = \{0^{2^n} : n \geq 0\}$



11-36: Turing Machine Diagrams

- Diagram for a Turing Machine that accepts the language  $L = \{a^{2^n} : n \geq 0\}$

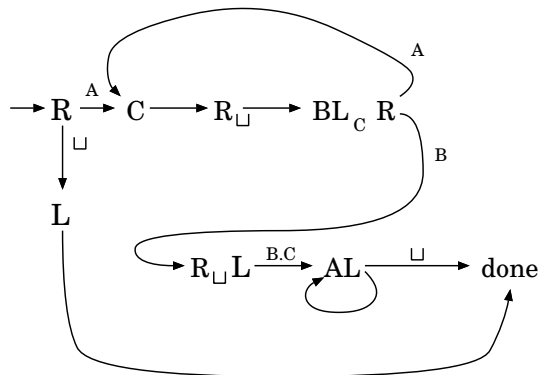
11-37: Turing Machine Diagrams

- Diagram for a Turing Machine that accepts the language  $L = \{a^{2^n} : n \geq 0\}$
- First: Write a TM that doubles a string of A's (converts  $w = A^n$  to  $w' = A^{2n}$ )
  - (Can use other tape symbols if desired)

11-38: Turing Machine Diagrams

- Write a TM that doubles a string of A's (converts  $w = A^n$  to  $W = A^{2n}$ )





11-39: Turing Machine Diagrams

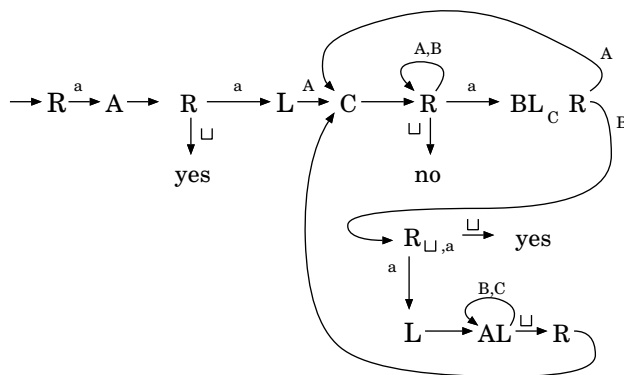
- Given a machine that converts  $w = A^n$  to  $w' = A^{2n}$ , how can we accept the language  $L = \{a^{2^n} : n \geq 0\}$ ?

11-40: Turing Machine Diagrams

- Given a machine that converts  $w = A^n$  to  $w' = A^{2n}$ , how can we accept the language  $L = \{a^{2^n} : n \geq 0\}$ ?
  - Check if the string is  $a$ . If so, accept.
  - Otherwise, overwrite the first  $a$  with an  $A$ .
  - Repeat:
    - Double the # of  $A$ 's – if a  $\square$  is overwritten in this process, halt and reject
    - After doubling, check to see if the next symbol is a  $\square$ . If so, halt and accept

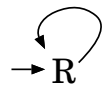
11-41: Turing Machine Diagrams

- Diagram for a Turing Machine that accepts the language  $L = \{a^{2^n} : n \geq 0\}$



11-42: Non-Halting TMs

- It is possible to create a Turing Machine that does not halt



- It is possible to create a Turing Machine that only halts on some inputs

$\rightarrow R_a \rightarrow \text{yes}$

#### 11-43: Deciding TM

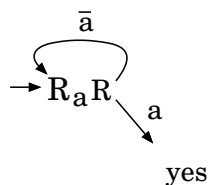
- A Turing Machine  $M$  *Decides* a language  $L$  if:
  - $M$  halts on all inputs
  - $M$  accepts all strings  $w \in L$
  - $M$  rejects all strings  $w \notin L$
- We've already seen an example of Turing machines that decide languages  $(a^{2^n}, a^{2^n})$

#### 11-44: Semi-Deciding TM

- A Turing Machine  $M$  *Semi-Decides* a language  $L$  if:
  - $M$  halts on all strings  $w \in L$
  - $M$  accepts all strings  $w \in L$
  - $M$  runs forever on all strings  $w \notin L$
- TM that semi-decides  $L =$  all strings over  $\{a, b\}$  that contain the substring aa

#### 11-45: Semi-Deciding TM

- A Turing Machine  $M$  *Semi-Decides* a language  $L$  if:
  - $M$  halts on all strings  $w \in L$
  - $M$  accepts all strings  $w \in L$
  - $M$  runs forever on all strings  $w \notin L$
- TM that semi-decides  $L =$  all strings over  $\{a, b\}$  that contain the substring aa



#### 11-46: Recursive Languages

- The *Recursive Languages* is the set of all languages that are decided by some Turing Machine
  - $L_{REC} = \{L : \exists \text{ Turing machine } M, M \text{ decides } L\}$
- Is  $L_{REC}$  closed under complementation?
  - That is, if  $L \in L_{REC}$ , must  $\bar{L} \in L_{REC}$ ?

#### 11-47: Recursive Languages

- The *Recursive Languages* is the set of all languages that are decided by some Turing Machine
  - $L_{REC} = \{L : \exists \text{ Turing machine } M, M \text{ decides } L\}$
- $L_{REC}$  is closed under complementation?
  - $L \in L_{REC} \implies \bar{L} \in L_{REC}$  (flip yes/no states)

**11-48: r.e. Languages**

- The *Recursively Enumerable (r.e.) Languages* is the set of all languages that are semi-decided by some Turing Machine
  - $L_{re} = \{L : \exists \text{ Turing machine } M, M \text{ semi-decides } L\}$
- Is  $L_{REC} \subseteq L_{re}$ ?

**11-49: r.e. Languages**

- The *Recursively Enumerable (r.e.) Languages* is the set of all languages that are semi-decided by some Turing Machine
  - $L_{re} = \{L : \exists \text{ Turing machine } M, M \text{ semi-decides } L\}$
- Is  $L_{REC} \subseteq L_{re}$ 
  - Replace “no” states with non-halting machine

**11-50: r.e. Languages**

- The *Recursively Enumerable (r.e.) Languages* is the set of all languages that are semi-decided by some Turing Machine
  - $L_{re} = \{L : \exists \text{ Turing machine } M, M \text{ semi-decides } L\}$
- Is  $L_{re} \not\subseteq L_{REC}$ 
  - More on this later ...