Fun with Turing Machines

Mark Greenstreet, CpSc 421, Term 1, 2008/09

- Primes
- Simple Operations
- A Programmable Turing Machine

Prime Sieve Algorithms

```
boolean[] primes(int n) {
    boolean[] b = new boolean[n];
    int p = 2; // current prime
    for(int i = 0; i < n; i++) b[i] = true;
    b[0] = false; b[1] = false;
    while(p < n) {
        for(int i = 2*p; i < n; i += p)
            b[i] = false; // a multiple of p
        for(p++; (p < n) && !b[p]; p++); // find next prime
    }
    return(b);
}
```

A TM for $\mathbf{1}^p$, where p is prime

Strategy: use tape as a sieve.

- For smallest prime not yet considered, cross-off all multiples of that prime.
- If we cross of the last 1 of the input string, then reject.
- Otherwise, if the last 1 of the input string is the next prime to consider, then accept.
- Example:

Input String	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
1 not prime	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
p = 2	0 1 1 ⊗ 1 ⊗ 1 ⊗ 1 ⊗ 1 ⊗ 1 ⊗ 1 ⊗ 1	eliminate multiples of 2
p = 3	0 1 1 0 1 ⊗ 1 0 ⊗ 0 1 ⊗ 1 0 ⊗ 0 1	eliminate multiples of 3
p = 5	0 1 1 0 1 0 1 0 0 ⊗ 1 0 1 0 ⊗ 0 1	eliminate multiples of 5
p = 7	0 1 1 0 1 0 1 0 0 0 1 0 1 ⊗ 0 0 1	...
p = 11	0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1	
p = 13	0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1	
p = 17	0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1	
accept	0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1	

- But, the tape head can only move one square at a time.

Using Markers

$\vdash$	0	p	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, $\vdash$ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	p	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	⋯
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	p	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	(p)	1	(0)	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	p	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	⋯
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	p	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	p	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	(p)	1	0	1	(1)	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	(p)	1	0	1	(0)	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	(0)	p	1	0	1	(0)	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

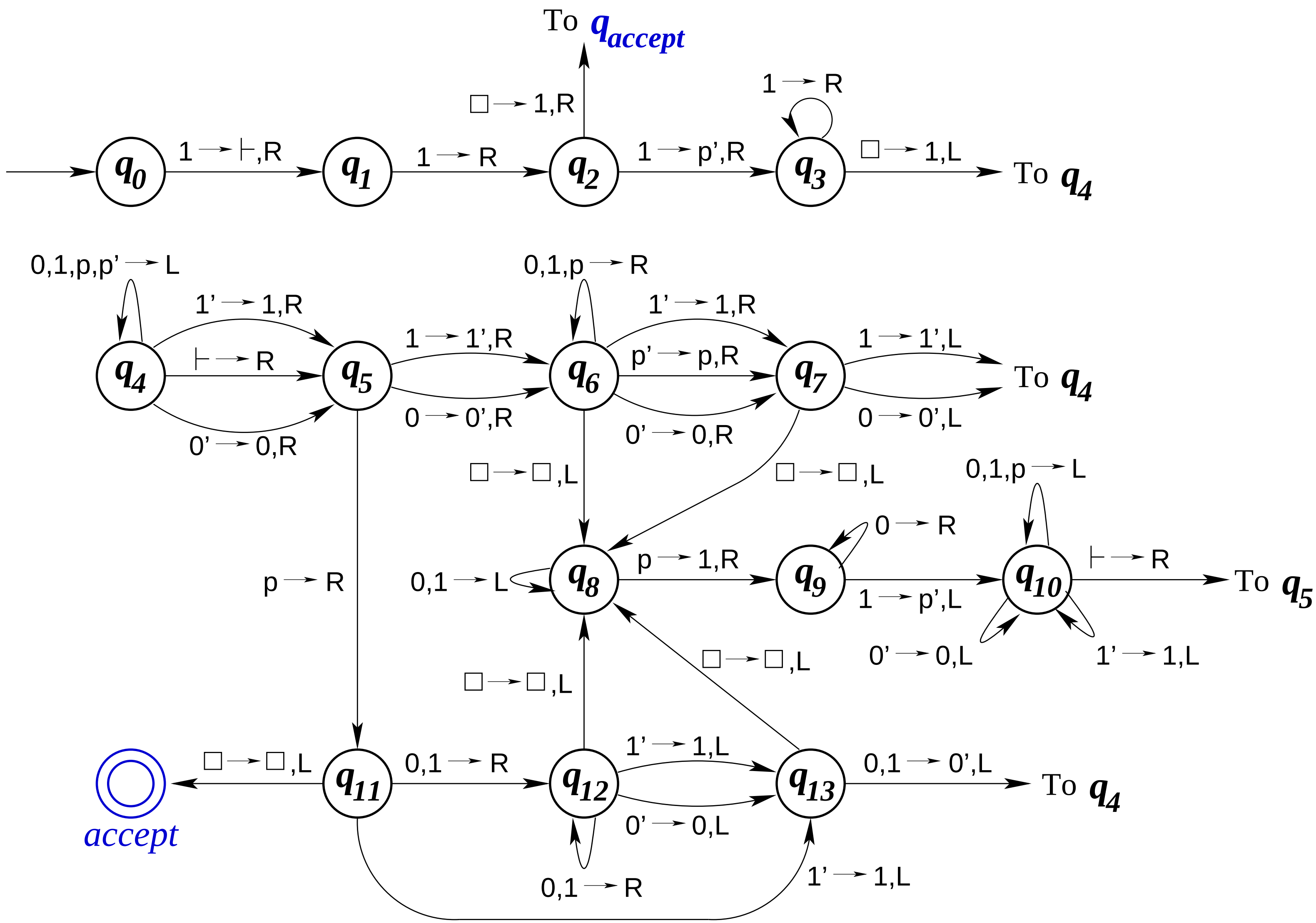
- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

Using Markers

⊢	0	p	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	□	□	···
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			

- We'll prepend a left end-marker, ⊢ to the tape.
  Note that this is at the "zero" position for the string.
- Imagine that we have two markers, a blue one and a red one.
  - We'll initially place the blue marker at the zero position of the tape,
  - and we'll initially place the red marker on the square for the current prime.
- Now, we'll repeatedly move both markers to the right one square at a time.
- When the blue marker reaches the square for the current prime
  - We'll write a `0` on the square for the red marker,
  - and we'll return the blue marker to the zero position.
- We repeat this until the red marker reaches a □, the end of the string.

A TM for 1^p



To q_accept
□ → 1,R
1 → R
q0
1 → ⊢,R
q1
1 → R
q2
1 → p',R
q3
□ → 1,L
To q4
0,1,p,p' → L
1' → 1,R
⊢ → R
0' → 0,R
q4
q5
1 → 1',R
0 → 0',R
0,1,p → R
q6
1' → 1,R
p' → p,R
0' → 0,R
q7
1 → 1',L
0 → 0',L
To q4
□ → □,L
□ → □,L
0,1,p → L
p → R
0,1 → L
q8
p → 1,R
q9
0 → R
1 → p',L
q10
⊢ → R
To q5
0' → 0,L
1' → 1,L
□ → □,L
□ → □,L
□ → □,L
accept
q11
0,1 → R
q12
1' → 1,L
0' → 0,L
q13
0,1 → 0',L
To q4
0,1 → R
1' → 1,L

How it works (states $q_0 \ldots q_3$)

- Omitted edges are to the reject state:
  - Most such edges can never be taken.
  - Real rejects occur from states q_0, q_1 and q_9 when reading a $\square$.
- States $q_0 \ldots q_3$ initialize the computation:
  - $q_0 \longrightarrow q_1$ writes the left endmarker on the tape.
  - $q_1 \longrightarrow q_2$ makes sure that there are at least two inputs in the input. If the machine encounters a $\square$ on either of the first two squares, it rejects.
  - $q_2 \longrightarrow q_3$ marks 2 as the first prime.
  - q_3 reads to the end of the tape, and then
  - $q_3 \longrightarrow q_4$ appends a 1 to make up for the leftmost 1 that was overwritten with the $\vdash$ symbol.

How it works (states $q_4 \ldots q_7$)

- State q_4 moves the head to the left to the square with the "blue" marker. That is either a $0'$, $1'$ or $\vdash$.
- States $q_4 \ldots q_7$ move the markers to the right:
  - $q_4 \longrightarrow q_5$ removes the left marker from the previous square.
  - $q_5 \longrightarrow q_6$ places the left marker on the next square. If that square held the p symbol, that means we've moved p positions and the machine transitions to state q_{11} to set the corresponding square at the right marker to 0 (described below).
  - q_6 moves to the right until the right marker is found.
    If them machine encounters a $\square$ first, that means we're done scanning for the multiples of the current prime. The machine transitions to state q_8 to determine the next prime to check.
  - $q_6 \longrightarrow q_7$ and $q_7 \longrightarrow q_8$ move the right marker one square to the right. Then the machine goes back to state q_4 to return the head to the left marker and start the next round.

How it works (states $q_8 \ldots q_{10}$)

- States $q_8 \ldots q_{10}$ look for the next prime. a multiple of the current prime.
  - q_8 moves to the left to find the current prime.
  - $q_8 \rightarrow q_9$ changes the p symbol to a 1.
  - q_9 moves to the right to find a square marked with a 1 (indicating a prime).
  - $q_9 \rightarrow q_{10}$ marks that prime with p′.
    If no such prime is found, then the last square on the tape must be marked with a zero (i.e. it is not a prime). The machine encounters a $\square$ and rejects.
  - q_{10} moves to the left, clearing the left marker on the way. This means that the left-marker is on the $\vdash$ square, leaving the machine ready to eliminate multiples of the new prime.

How it works (states $q_{11} \ldots q_{13}$)

- States $q_{11} \ldots q_{13}$ write a 0 on a square that is a multiple of the current prime.
  - $q_{11} \longrightarrow q_{accept}$:
    - If the symbol following the square for the prime is a $\square$, then the input string was $\mathtt{1}^{\mathtt{p}}$ where p is the current prime. The machine accepts.
    - Otherwise, the machine moves to the right, $q_{11} \longrightarrow q_{12}$, to start looking for the right marker.
    - If the right marker is immediately after the prime, the machine move directly from state q_{11} to q_{13}. This happens when $p = 2$ and the right marker is on the square for 3.
  - q_{12} the machine moves to the right looking for the right marker.
  - $q_{12} \longrightarrow q_{13}$ the machine moves the right mareer one square to the right.
  - $q_{13} \longrightarrow q_4$ if the next square is either a 0 or a 1, the machine writes a 0 (to indicate that the square is in a non-prime position) and marks it for the next round of the scan.

A TM that acts like a “real” computer

The tape

Data manipulation

Making the TM programmable

The Tape

The tape is of the form

$$\vdash \Psi x_0 \Psi x_1 \Psi \cdots \Psi x_n \Psi \square^*$$

where

- Each x_i is in $L(\mathtt{1}^*)$. If $x_i = 1^j$, then x_i represents the integer j.
  - This unary encoding is inefficient (uses lots of tape), but tape is free ☺.
  - We could describe a machine that used binary (or decimal, etc.) for its number representations, but that would add extra details to the description that aren't critical for our point that we can make a programmable computer.
- Each Ψ is a $\#$ symbol followed by a string in $\{A, B, P\}^*$.
  - The tape has exactly one A, exactly one B and exactly one P.
  - The symbols A, B and P mark words that the "program" is currently manipulating.

Operation: insert a 1

- Add states q_{11}, $q_{1\#}$, q_{1A}, q_{1B}, q_{1P}, and $q_{1\Box}$ with the following transistions:

	1	$\#$	A	B	P	$\Box$
q_{11}	$(1, q_{11})$	$(1, q_{1\#})$	$(1, q_{1A})$	$(1, q_{1B})$	$(1, q_{1P})$	$(1, q_{1\Box})$
$q_{1\#}$	$(\#, q_{11})$	$(\#, q_{1\#})$	$(\#, q_{1A})$	$(\#, q_{1P})$	$(\#, q_{1P})$	$(\#, q_{1\Box})$
q_{1A}	(A, q_{11})	$(A, q_{1\#})$	(A, q_{1A})	(A, q_{1B})	(A, q_{1P})	$(A, q_{1\Box})$
q_{1B}	(B, q_{11})	$(B, q_{1\#})$	(B, q_{1A})	(B, q_{1B})	(B, q_{1P})	$(B, q_{1\Box})$
q_{1P}	(P, q_{11})	$(P, q_{1\#})$	(P, q_{1A})	(P, q_{1B})	(P, q_{1P})	$(P, q_{1\Box})$

- The entry in row q column c is a tuple of the form (c', q'). When the machine is in state q and there is a c on the current tape square, the machine writes a c' on the tape, and transitions to state q' and moves to the rights.

Inserting a 1: explanation

- This machine-fragment starts in state q_{11} at the position where a 1 should be inserted and ends in state $q_{1\square}$ having inserted the one.
- Initially,
  - The machine writes a 1,
  - Uses its finite state to store the value of the tape symbol that it overwrote, and
  - moves one square to the right.
- At each subsequent step
  - The machine writes the symbol from the previous square,
  - Uses its finite state to store the value of the symbol that was at this square, and
  - moves one square to the right.
- When it reaches the end of the tape string (i.e. a $\square$)
  - The machine writes the symbol from the previous square and
  - moves one square to the right, entering state $q_{1\square}$.
  - The rest of the TM can “connect” with state $q_{1\square}$ to continue the computation.

Deleting a symbol

We add states to:

- Write a □ at the current tape position and move to the right.
- Continue moving to the right until we reach another □ (the end of the tape string).
- Use a variation of the “insert a 1” procedure to “insert” another blank on the last non-blank square of the tape, and go to the left, copying the overwritten symbols until we reach the □ at we wrote at the beginning.
- Now, the symbol that we had wanted to delete is gone, and the string to its right has been shifted over one tape square.

The resetA "instruction"

- Move the A marker to the first $\#$ (i.e. have it mark x_0).

loop { Move left to the endmarker, $\vdash$.
Move right two squares (one after the first $\#$).
Insert a A (like inserting a `1` as described above).
Move to the left (from the right end of the tape)
until reaching the previous A.
Delete the previous A (as described above).
}

The clrA "instruction"

- Set the word marked by A to 1^0 (a.k.a. ϵ).

loop { Move left to the endmarker, $\vdash$.
Move to the right until reaching the A.
Move to the right past the A and any other markers (i.e. B or P).
if the current symbol is a 1
delete it (as described above).
else exit-loop.
}

The incrA “instruction”

- Add one to the word marked by A.

Move left to the endmarker, $\vdash$.
Move to the right until reaching the A.
Move to the right past the A and any other markers
(i.e. B or P).
Insert a `1` as described above.

The addAB "instruction"

- Replace the word marked by A with the sum of the word marked by A and the word marked by B.

  Move left to the endmarker, $\vdash$.
  Move to the right until reaching the B.
  Move to the right past the B and any other markers (i.e. A or P).
  while the current symbol is a 1 {
      Mark the current symbol (i.e. change it to $1'$).
      Increment the word marked by A (see the incrA instruction).
      Move to the $1'$.
      Unmark it and move one square to the right.
      if the current symbol is a $\#$, exit-loop.
  }

- Other "ALU instructions" can be implemented in a similar manner.

The moveAB "instruction"

- Let x_B be the value of the word marked by the B marker. Move A to mark x_{x_B}.
- For example, if B marks word 5, and A marks word 17, and $x_5 = 42$ and $x_17 = 2$, then executing moveAB will
  - Set A to mark word 42.
  - Leave B marking word 5.
  - The rest of the values on the tape are unchanged.

Implementing moveAB

- Move left to the endmarker, $\vdash$.
  Move to the right until reaching the A.
  Delete the A.
  Move left to the endmarker, $\vdash$.
  Write an A after the first $\#$.
  for each 1 in the word marked by B {
      Move the A marker one $\#$ to the right.
          (If there is not such $\#$, append $\#$'s to the end of
          the tape string as needed.)
  }
- This lets us move the markers to arbitrary locations on the tape – in other words, it provides memory access.
- Note that by appending $\#'s$ onto the tape as needed, our TM computer never runs out of memory.

Instruction summary

- We now have basic instructions for data manipulation:
  - resetA: Move the A marker to word x_0.
  - clrA: Set the word marked by A to 0.
  - incrA: Add one to the word marked by A.
  - addAB: Replace the word marked by A with the sum of the words marked by A and B.
  - moveAB: Move the A marker to the word indicated by the word marked by the B marker.
- We could make similar "instructions" manipulating the word marked by the B (or P) markers.
  - For example, moveAA sets the A marker to the word indicated by the word at the current position of the A marker.
  - If A marks x_{17}, and x_{17} holds the value 1^{42}, then moveAA will set the A marker to mark word x_{42}.

An example

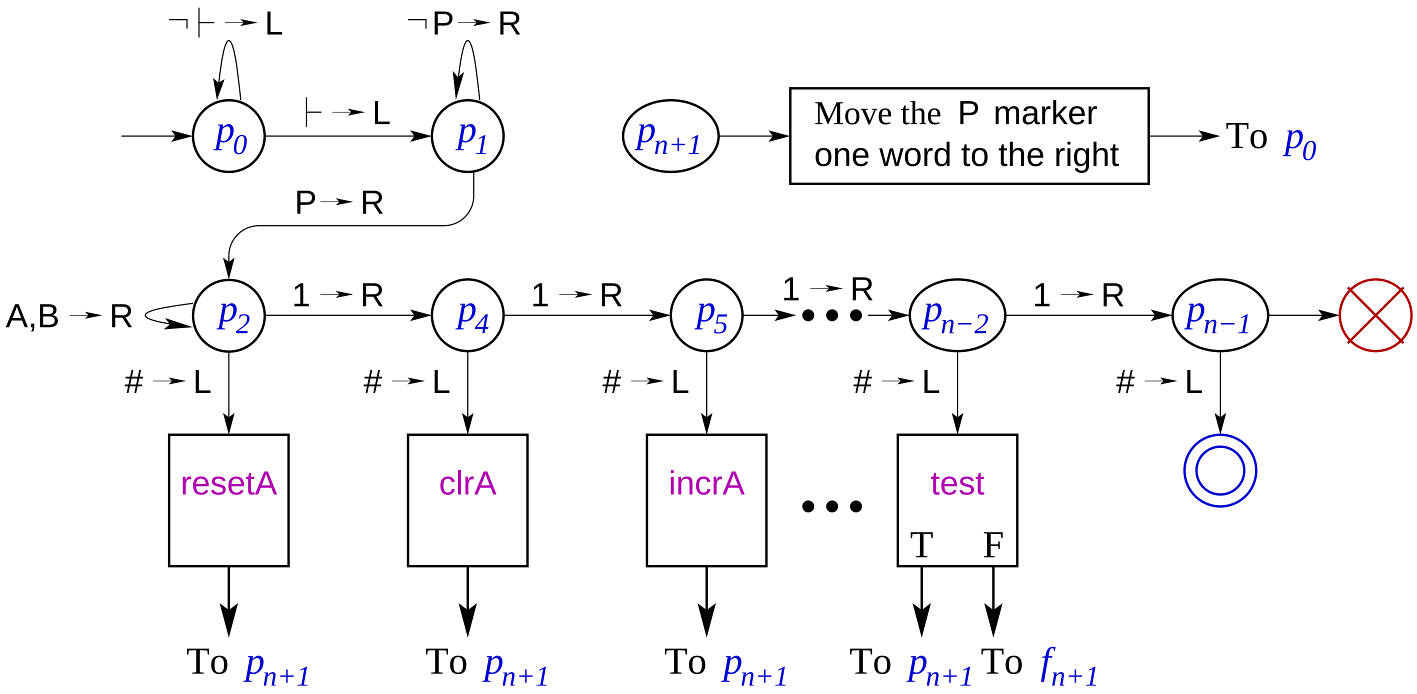
- Copy the word marked by B to x_5:

resetA	A now marks x_0
clrA	$x_0 \leftarrow 0$
incrA	$x_0 \leftarrow 1$
incrA	$x_0 \leftarrow 2$
incrA	$x_0 \leftarrow 3$
incrA	$x_0 \leftarrow 4$
incrA	$x_0 \leftarrow 5$
moveAA	A now marks x_5
clrA	$x_5 \leftarrow 0$
addAB	$x_5 \leftarrow x_B$

  where x_B is the value of the word marked by B.

- But how do we store and execute instructions?

A Stored Program TM



- the P marker is the "program counter."
- I added instructions for accept and reject.
- The testA instruction implements a branch:
  - if the word marked by A is non-zero, then next instruction is executed normally.
  - otherwise, the f states provide alternative implementations of each instruction.

This week

- Reading
  - October 20 (Today): *Sipser* 3.2.
  - October 22 (Wednesday): *Sipser* 3.3.
  - October 24 (Friday): *Sipser* 4.1.
- Homework
  - October 20 (Today): Homework 5 due.
  - October 24 (a week from today): Homework 6 due; homework 7 goes out.