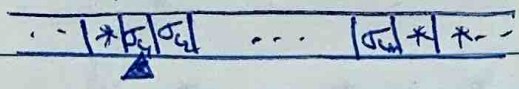
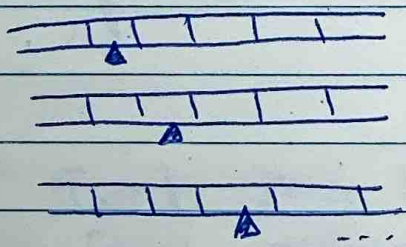
- a program has exactly one STOP instruction (at the end)
- input for a program is a word $w = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_m} \in A^*$ placed in consecutive squares w/ head scanning first symbol:



- if program halts STOP, output is word v appearing on tape (ignoring empty squares)
  - ↳ output can be empty.

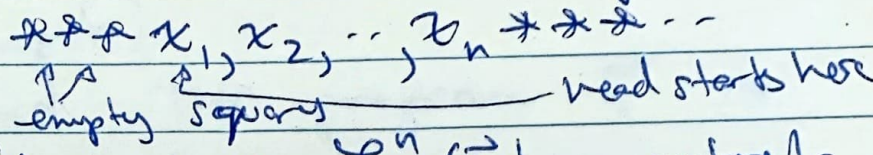
Ex a TM program that never halts

1. RIGHT
2. GOTO 1
3. STOP



- there is a natural notion of a TM-computable function
- to separate inputs, we introduce "," (comma) as a separator symbol
- Sps P is a TM program on $A \cup \{,\}$
- for $n \geq 1$, say P computes $\varepsilon^n_P : (A^*)^n \to A^*$ if ε^n_P is the function s.t. for input $\vec{x} = (x_1, \ldots, x_n)$

we have
$\varphi_P^n(\vec{x})\downarrow$ iff P halts on input



and in this case $\varphi_P^n(\vec{x})$ = output of computation. (with commas deleted)

note: if $x_1 = \varepsilon = \emptyset$ then head starts over first comma.

Defn $F: (A^*)^n \to A^*$ is TM-computable if there is $B \supseteq A$ and a TM program P on $B \cup \{,\}$ s.t. $F = \varphi_P^n \restriction A^*$

Theorem every RM-computable function is TM-computable.

Pf:- before functions, first say how to translate between RM programs and TM progs
- Sps P is an RM program (on A)
- Sps m is largest register mentioned by P (or any fixed number larger than this)
- can think of P as determining a function w/ m inputs (or "up to" m inputs): in sense that only stuff printed up to R_m affects output of P

only important thing is we're fixing m

- we construct an "equivalent" TM program P' (equivalent on some m inputs)
- precisely: find TM program P' on $A \cup \{,\}$ s.t.
  (i) if we put $w_1, .., w_m$ in P (words on separate registers – no commas) and $w_1, .., w_m$ in P' (symbols on squares, words separated by commas) then P halts iff P' halts
  (ii) if P halts on input $w_1, .., w_m$ w/ output $v_1, v_2, ..$ then P' halts w/ output $v_1 = a_1 a_2 .. a_t$ (i.e. only first word, symbols printed in consec squares) w/ head over v_1.

note: what P' does on inputs not of the form $w_1, ..., w_m$ is irrelevant.

↪ sps we've done this (i.e. for every RM P can find TM P' s.t. (i), (ii)) and sps $F: (A^*)^n \to A^*$ is RM-comp'ble.
- i.e. there is RM prog P on $B \supseteq A$ that computes F
- consider the TM prog P'
- P' only guaranteed to return correct outputs on m-ary inputs

- So we first need to run a prep TM program R that turns

$$\cdots * * x_1, x_2, \ldots, x_n * * * \cdots$$

into $\cdots * * x_1, x_2, \ldots, x_n \underbrace{,,,,\ldots,}_{m-n} * * \cdots$

($m-1$ total commas)

- fact (take for granted / exercise): there is such a prog R

- then: the TM program $P'' = R$ followed by P computes f (and we're done).

→ (we're being picky about commas since if programming a TM helps to know exact # of commas in expected inputs)

- So: how do we find P' satisfying (i) + (ii)?
- By HW3: may assume P only has ADD$_j$, DEL$_j$, IF FIRST$_j$ GOTO$_\ell$, and STOP instructions.

- Idea is: if I_k is an (RM) instruction in P then in P' we'll substitute I_k w/ a block of TM instructions B that simulate I_k

- i.e. if I_k changes $(x_1, .., x_m)$ to $(x_1', .., x_m')$ on the RM
then B will change $**x_1, .., x_m**$ to $**x_1', .., x_m'**$ on the TM
and move head back to first symbol in x_1'.

- ... unless I_k is STOP
then B will change $**x_1, .., x_m**$ to $**x_1**$ (and move head to front)

- we describe B's for each type of ADD, DEL, IFFIRST GOTO, and STOP instruction
- really: mostly wave hands and assert TMs can make certain moves.

---

1. ADD j Rn
   - want TM instructions that change $**x_1, .., x_n, x_{n+1}, .., x_m**$ to $**x_1, .., x_n \sigma_j, x_{n+1}, .., x_m**$
   - B will be concatenation of following subroutines:
     "move head to nth comma"
     (i.e. comma between x_n, x_{n+1})
     "print σ_j"
     (i.e. change $**x_1, .., x_n, x_{n+1}, .., x_m**$ to $**x_1, .., x_n \sigma_j, x_{n+1}, .., x_m**$)

"shift right"

(change $\cdots * x_1, \ldots, x_n \sigma_j, x_{n+1}, \ldots, x_m * \cdots$

to $\cdots * x_1, \ldots, x_n \sigma_j, * x_{n+1}, \ldots, x_m * \cdots$)

"go back to empty square and print σ_j"

$\rightarrow \cdots * x_1, \ldots, x_n \sigma_j, x_{n+1}, \ldots, x_m * \cdots$

"go back to first symbol in x_1"

- let's explicitly describe a "move to nth comma" routine
- assume $A = \{\sigma_1, \ldots, \sigma_k\}$ has k letters
- head assumed to be @ first symbol of x_1 in $\cdots * x_1, \ldots, x_m * * \cdots$

1.	LEFT	} moves head back n forth, just so we have a leading RIGHT to loop.
2.	RIGHT	
3	IF$_1$ GOTO 2	} if first symbol not a comma ($i.e.$ $x_1 \neq 0$) move right once → (repeats till comma)
4	IF$_2$ GOTO 2	
	⋮	
k+2.	IF$_k$ GOTO 2	
k+3	IF$_0$ GOTO k+4	} if first symbol, move right once (now @ first symbol in x_2)
k+4	RIGHT	
k+5	IF$_1$ GOTO k+4	} cycle thru x_2 till comma
	⋮	
k+k+4.	IF$_k$ GOTO k+4	
k+k+5	IF$_0$ GOTO k+k+6	} at comma go to x_3
k+k+6	RIGHT	

... (do this n times)
(once we get to nth comma
go to next block).

l. IF, GOTO l+1 | "print σ_j"
l+1 (commands for [redacted])

↳ other routines: you try. ✓

2 DEL R_n
- similar, you try (need a "shiftLeft" routine) ✓

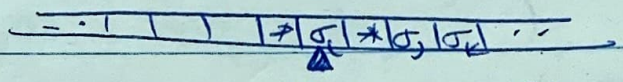
3 IF FIRST; [redacted] R_n GOTO l
- you try ✓

4 STOP
- here need TM routine to first change $**x_1,..,x_m*...$ to $*x_1*...$ then go to first symbol, then stop.
- you try ✓

TMs revamped

- we present an alternative form of TM
- $A = \{\sigma_1, ..., \sigma_k\}$ is our alphabet introduce $\sigma_0 = *$ = empty square symbol

- Still have a bi-infinite tape and read/write head
- but now head can be in one of finitely many states $\{q_0, \ldots, q_p\}$



- q_0 is the start state
- several of the q_i's may be stop states (can have more than one)

- instead of a program (i.e. list of instructions), we have a table that consists of $(p+1)(k+1)$-many 5-tuples of the form

$$(q_i, a, b, j, q_m)$$

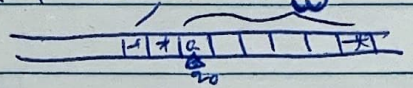
where q_i, q_m are states
$a, b \in A$ are symbols
$j \in \{-1, 0, 1\}$

- have one such tuple for every state/symbol combo (q_i, a)
- read tuple as:

"if in state q_i and square contains a, write b, move j, and change to state q_m"

where $j = -1, 0, 1$ corresponds to "left" "stay" "right" resp.

- on input word w we run TM according to these tuples, starting with tuple $(q_0, a, \ldots)$, where a is first symbol in w, w/ head @ a.



- we output once we hit a stop state

(- for q_j a stop state, other info in a tuple (q_j, a, b, j, q_i) irrelevant... we just stop)

- may never hit a stop state.