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_i a stop state, other info in a tuple (q_i, a, b, j, q_k) irrelevant... we just stop)
- may never hit a stop state.

Prop'n Every "program Turing machine" can be simulated by a "table Turing machine" and vice versa.

Pf: - Suppose first we are given a program P for a TM with instructions $I_0, I_1, \ldots, I_n = \text{STOP}$
- we introduce states $q_0, q_1, \ldots, q_n$ (only stop state) corresponding to each instruction
- we construct a table for these states that simulates P.

- for a fixed $K \leq n$:

(i) if I_K = PRINT, add $(q_K, a, \sigma_j, 0, q_{K+1})$ to table for all $a \in A$

"if in state q_K, overwrite any a w/ σ_j, stay at current square, go to state q_{K+1}"

↳ for table TM being in state q_K corresponds to being at Kth instruction I_K in program P.

(ii) if I_K is DEL, add $(q_K, a, *, 0, q_{K+1})$ for all $a \in A$

(iii) " " RIGHT add $(q_K, a, a, 1, q_{K+1})$ $\forall a \in A$

(iv) " " LEFT " $(q_K, a, a, -1, q_{K+1})$ $\forall a \in A$

(v) " " GOTO$_\ell$ " $(q_K, a, a, 0, q_\ell)$ $\forall a \in A$

(vi) " IF$_j$ GOTO$_\ell$ add $(q_K, \sigma_j, \sigma_j, 0, q_\ell)$

and for $a \neq \sigma_j$, add $(q_K, a, a, 0, q_{K+1})$

(vii) if $I_K = I_n$ = STOP add $(q_n, a, a, 0, q_n)$ $\forall a \in A$

- Convince yrself: this works! i.e. if we run P on a program TM and the table constructed from P on a table TM (on same input) get same output.

- actual proof: induct on length n of P

(q, a, b, s, q')

- Idea is: table instructions are conditioned on both current square and current state, whereas program instructions I_k aren't (except IF, GOTO.. but only on square)
- so we translate an I_k to a table instr. by thinking of I_k as beginning "if I'm at kth instruction (yes) and any of the following letters are printed (all of them) then do I_k" ✓

Conversely: - given a table of instructions can we write a program P that simulates it?

- Spr we have a machine with states $q_0, q_1, \ldots, q_p$ and a corresponding table
- Spr $S \subseteq \{0, 1, \ldots, p\}$ is the set of stop state indices.
- we may assume q_0 is not a stop state: otherwise the table TM is simulated by the following program:

1. STOP

- we label the instructions in our table as $(q_i, \sigma_j, \sigma_{p(i,j)}, t_{i,j}, q_{m(i,j)})$

Recall: $\sigma_0 = *$, $\sigma_1 = B$, $\sigma_2 = A$

where the function $p(i,j)$ is into the set of alphabet indices $\{0, 1, \ldots, k\}$

and $m(i,j)$ is into the state indices $\{0, 1, \ldots, p\}$

- For each non-stop index $i \notin S$, introduce the following block of instructions:
(- the actual numerical value of l_i below is determined after we stack these blocks together to form our program)

$(q_i$

$(q_i, \sigma_j$ $\left\{\begin{array}{l} l_i.\ IF_0\ GOTO\ l_{i,0} \\ l_i+1.\ IF_1\ GOTO\ l_{i,1} \\ \\ l_i+K.\ IF_K\ GOTO\ l_{i,K} \end{array}\right.$ } imagine we're in state q_i, this sorts according to what is printed on current square.

$\left\{\begin{array}{l} l_{i,0}.\ PRINT\ \rho(i,0) \\ l_{i,0}+1.\ (LEFT/RIGHT)\ (\text{depending on } t_{i,0} = -1, \text{ or } 1)\ (\text{if } t_{i,0} = 0, \text{ no instruction}) \\ l_{i,0}+2.\ GOTO\ l_{m(i,0)} \\ \quad (\text{i.e. goto block of instructions corresponding to } q_{m(i,0)}) \end{array}\right.$

$(q_i, \sigma_0, \sigma_{\rho(i,0)}, t_{i,0}, q_{m(i,0)}$

Same thing for $(q_i, \sigma_1$

$\left\{\begin{array}{l} l_{i,1}. \\ \vdots \end{array}\right.$

σ_2 $l_{i,2}$.

$\vdots$

- Finally, for every step index $i \in J$ let ℓ_i-th instruction be GOTO n where I_n = STOP is final instruction.
- Concatenate these ℓ_i blocks together in order (for each q_w)
- resulting program simulates the table TM.

Markov algorithms

- we introduce yet another model of computation!
- still working w/ finite alphabet A.

Def'n a Markov algorithm (MA) on A is a finite sequence $\mathcal{A}$ of productions

$$p_1 \to q_1$$
$$p_2 \to q_2$$
$$\vdots$$
$$p_n \to q_n$$

where each $p_i, q_i \in A^*$ are words in A.

- in some of these productions the arrow $\to$ may have a subscript $\to_t$

Def'n $p \to q$ is a simple production
$p \to_t q$ is a terminal production.

- on an input word $w \in A^*$, algorithm goes through the productions in order until it finds first one $p_i \to q_i$ s.t. p_i is a subword of w
  i.e. $w = x p_i y$ for some $x, y \in A^*$
- replaces p_i in w with q_i to get w'
- repeats procedure on w'
- stops when no productions apply to current word or when first production that does apply is terminal.

Def'n Sps $x, y \in A^*$ are words.
Write $A : x \to y$ ("the algorithm transforms x directly to y") to mean:
the first production $p_i \to q_i$ that applies to x is simple and applying it to x yields y

- e.g. Sps $A = \{ab \to cba,\ b \to \emptyset\}$ (written in order)
- if $x = cdbba$ then $A : x \to y$ where $y = cdba$

Convention: if $p_i = \emptyset = 0$ is empty word and $p_i \to q_i$ is a production, when applied it transforms x to $q_i x$ (replaces empty space @ beginning with q_i)

Def'n Write $A: x \to_t y$ ("A transforms x directly to y terminally") if first instruction that applies to x $P_i \to_t q_i$ is terminal and transforms x to y.

- we inductively define $A: x \Rightarrow_m y$ ("A transforms x to y in m steps")
  - $A: x \Rightarrow_0 y$ iff $x = y$
  - $A: x \Rightarrow_{m+1} y$ iff $\exists z \in A^*$ $A: x \Rightarrow_m z$ and $A: z \to y$

- if $A: x \Rightarrow_m y$ and last instruction is terminal (no prev. instructions terminal) we say A transforms x to y terminally in m steps
- we write $A(x) = y$ if A produces y and stops when applied to x, i.e. $\exists m$ s.t. $A: x \Rightarrow_m y$ terminally or $A: x \Rightarrow_m y$ and no productions apply to y.

Ex if $A = \{ab \to cba,\ b \to \varepsilon\}$ as above and $x = cdbba$ then algorithm goes $x \to cdba \to cda$ and stops
- so $A: x \Rightarrow_1 cdba$
  $A: x \Rightarrow_2 cda$ terminally
- so $A(x) = cda$

Ex: can have non-terminating loops, e.g.
if $A = \{a \to b,\ b \to a\}$ and $x = a$
then $x \to b \to a \to b \to a \to \cdots$
so: $A: x \Rightarrow_m b$ $\forall$ odd m
$A: x \Rightarrow_m a$ $\forall$ even m

- here $A(x)$ undefined.

- Given an MA A on the alphabet $A \cup \{,\}$ (with comma), for a fixed n can think of A as defining a (partial) function $e^n_A : (A^*)^n \to A^*$ where:
  - $e^n_A(x_1, \ldots, x_n)\downarrow$ iff A terminates on input $x_1, \ldots, x_n$ (commas included)
  - and in this case we have $e^n_A(\vec{x}) = y$, where y is $A(\vec{x})$ w/ all commas deleted.

Def'n $f: (A^*)^n \to A^*$ is Markov computable if for some finite alphabet $B \supseteq A$ and MA A on $B \cup \{,\}$ we have $f = e^n_A \restriction A^*$.

Ex Consider the reversal function $f(w) = w^*$ (i.e. if $a_1 a_2 \cdots a_n$ is word in A, $f(a_1 a_2 \cdots a_n) = a_n \cdots a_2 a_1$)

- We show f is Markov computable
- Let $B = A \cup \{\alpha, \beta\}$ (so: we're actually using an expanded alphabet!).

- Let A be following MA on B:

$\{\alpha\alpha \to \beta,\ s\beta \to s\beta,\ \beta\alpha \to \beta,\ \beta \to_t 0,$
$\alpha st \to t\alpha s,\ 0 \to \alpha\}$

↳ here s, t are letters in A and we've written all rules w/ s, t as actually a sequence of rules all next to eachother in A

- Consider A applied to a word $abc \in A^*$

$abc \to \alpha abc \to b\alpha ac \to bc\alpha a$
$\to \alpha bc\alpha a \to c\alpha b\alpha a \to \alpha c\alpha b\alpha a$
$\to \alpha\alpha c\alpha b\alpha a \to \beta c\alpha b\alpha a \to c\beta\alpha b\alpha a$
$\to c\beta b\alpha a \to cb\beta\alpha a \to cb\beta a \to cba\beta$
$\to_t cba$ (stop!)

- works in general! so $\mathcal{C}_A = f$ and f is Markov computable!

Recall: if f is a (partial) function on a given alphabet, we've shown:

f is recursive
$\Rightarrow f$ is RM computable
$\Rightarrow f$ is TM computable
$\Rightarrow f$ is MA computable
$\Rightarrow f$ is recursive

now want to show

(i.e. close the implication (ecp.)