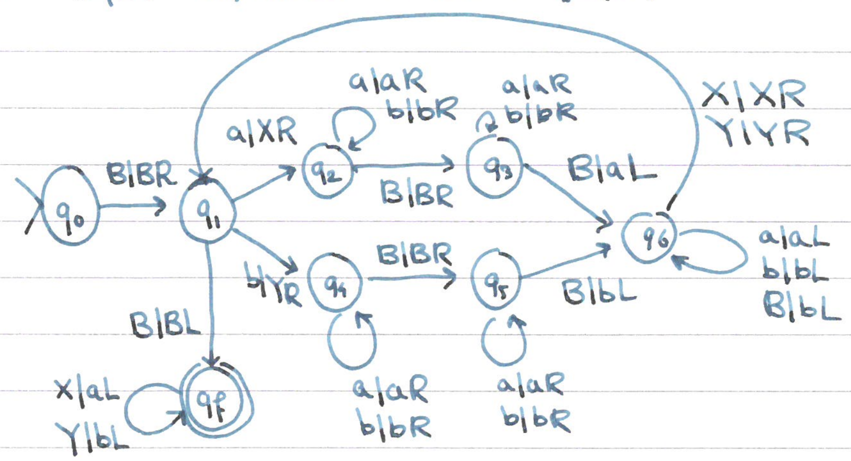
Computability

8.1.1 a Turing Machine (T.M.) is a quintuple
$M = (Q, \Sigma, \Gamma, \delta, q_0)$ where
- Q is state set [redacted]
- Σ is input alphabet
- $\Gamma \supseteq \Sigma$ is tape alphabet, which at least has a blank symbol $B \in \Gamma \setminus \Sigma$
- $\delta : Q \times \Gamma \nrightarrow Q \times \{L, R\}$ transition function (partial function)
- q_0 is initial state.

note: partial function $A \nrightarrow B$ is a relation $f \subseteq A \times B$ with
$\forall a \in A \, (\exists! \, b \in B \, (a f b) \vee \nexists b \in B \, (a f b))$

ex. COPY with input alphabet $\Sigma = \{a, b\}$:

BuB
↕ ↕
BuBuB



and for general Σ, there is a COPY machine with $4 + 2|\Sigma|$ states.

8.1.2 tracing a computation: we denote a TM in state q_i with head on tape at pos. n and a tape in state $\gamma_1 \cdots \gamma_m \in \Gamma^{<\omega}$ as

$$\boxed{\gamma_1 \cdots \gamma_{n-1} \; q_i \; \gamma_n \cdots \gamma_m}$$

In order not to halt abnormally, it is conventional to have a B at pos. 1 to indicate "end of tape" example: running COPY on BabB:

$$
\begin{aligned}
q_0 BabB &\vdash B q_1 abB \vdash BX q_2 bB \\
&\vdash BXb q_2 B \vdash BXbB q_3 B \\
&\vdash BXb q_6 Ba \vdash BX q_6 bBa \\
&\vdash B q_6 XbBa \vdash BX q_1 bBa \\
&\vdash BXY q_4 Ba \vdash BXYB q_5 a \\
&\vdash BXYBa q_5 B \vdash BXYB q_{16} ab \\
&\vdash BXY q_6 Bab \vdash BX q_{16} YBab \\
&\vdash BXY q_1 Bab \vdash BX q_f YBab \\
&\vdash B q_f XbBab \vdash q_f BabBab \downarrow
\end{aligned}
$$

8.2.1 A T.M with final state in addition to $Q, \Gamma, \Sigma, \delta, q_0$ has a subset $F \subseteq Q$. If a computation on an input $s \in \Sigma^{<\omega}$ halts in a $q_i \in F$ then M accepts s. $\mathcal{L}(M)$ is the language of all such s.

- M <u>recognises</u> $\mathcal{L}(M)$.
- if halts on all inputs, it <u>decides</u> $\mathcal{L}(M)$.

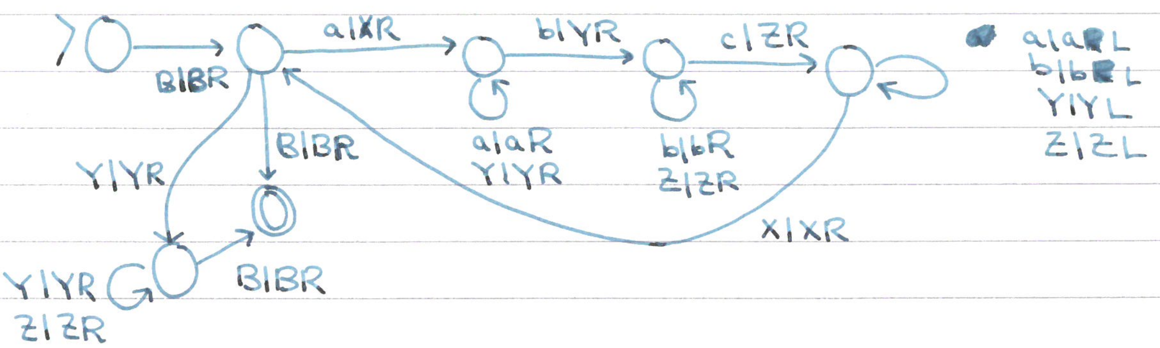
- a language $\mathcal{L}(M)$ is <u>rec. enum.</u> if it is recognised by some T.M.
- it is <u>recursive</u> if it is decided by some T.M.

note you can always reduce F to a single final state q_f by adding transitions and states as follows:

- $Q' = Q \cup \{q_f\}$
- if $\delta(q_i, \gamma)\uparrow$ for $q_i \in F$, $\gamma \in \Gamma$, then let $\delta'(q_i, \gamma) = [q_f, \gamma, R]$. Otherwise, $\delta = \delta'$

we see M' halts in q_f $\Leftrightarrow$ M would have halted in F.

ex. T.m. that accepts $\{a^i b^i c^i \mid i \in \mathbb{N}_{\geq 0}\}$:



8.3.1 acceptance criterium: "by halting". This means M acepts an $s \in \Sigma^{<\omega}$ if its computation halts on input s.

thrm • equally powerful in the sense that

$\mathcal{L}$ is acepted by some M' by halting

$\Leftrightarrow$

$\mathcal{L}$ is acepted by some M by final state.

namely, if M' exists, let M be the machine M' with final states Q.

conversely, if M exists, add a "looping state"

$q_\varepsilon \cup Q =: Q'$, with

$\delta'(q_\varepsilon, \gamma) = [q_\varepsilon, \gamma, R] \quad \forall \gamma \in \Gamma$, and

$\delta'(q_i, \gamma) = [q_\varepsilon, \gamma, R]$ for all $q_i \in F^c$, $\gamma \in \Gamma$ such that $\delta(q_i, \gamma)\uparrow$

then M' loops on s $\Leftrightarrow$ M loops or M halts in non-F state.

$\Leftrightarrow s \notin \mathcal{L}(M)$

— so $\mathcal{L}(M') = \mathcal{L}(M)$.

$\square$

8.4.1 a multitrack T.M. is a T.M. with its tape divided into multiple tracks. (say, $k \geq 1$)

So its tape is now an element of $(\Gamma^k)^{<\omega}$
and $\delta : Q \times \Gamma^k \nrightarrow Q \times \Gamma^k \times \{L, R\}$

a language L is accepted by a single-track T.M M
$\Leftrightarrow$ L is accepted by a multitrack T.M M'.

$\Rightarrow$ because $k = 1$ makes single-track multitrack
(or we can just ignore the upper track).
$\Leftarrow$ because we can set $\Gamma = (\Gamma')^k$,
$\Sigma = \Sigma' \times \{B\}^{k-1}$ and $\delta = \delta'$ in this way

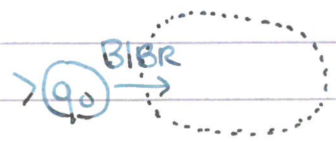
it does require us to work over different alphabets.
But this is not a weakness as any alphabet can be encoded in 0-1 strings, and we can adopt any TM to work with those.

8.5.1 Two-way Turing m. has its tape extend in 2 directions, its positions resembling $\mathbb{Z}$ rather than $\mathbb{N}_{\geq 0}$. the word is still placed on position $1, 2, \ldots$

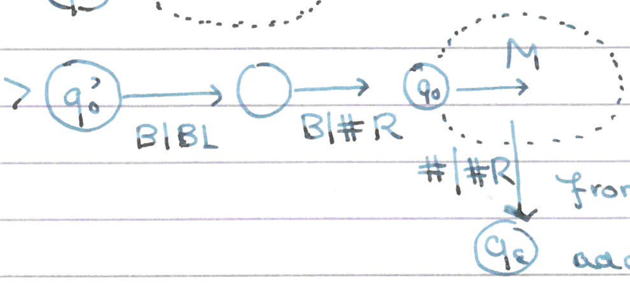
This means abnormal termination cannot happen.

any T.M M on $\mathbb{N}_{\geq 0}$ can be turned into an equivalent T.M M' on $\mathbb{Z}$ by letting it go into an additional error state if it "crosses 0".

let M :



let M' :



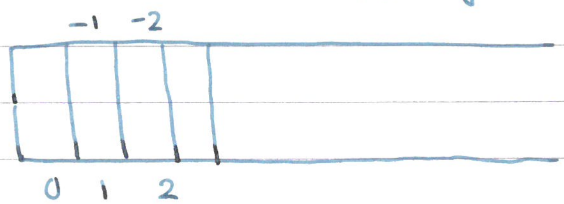
from every $q \in Q_M$, add $\delta(q, \#) = (q_E, \#, R)$

then M' goes into $q_E \notin F$ $\Leftrightarrow$ M halts abnormally.

Conversely, if $M^\bullet$ acepts a language $\mathcal{L}(M^\bullet)$ and is 2-way, we can construct a 2-track TM M'

namely $Q' = (Q \cup \{q_s, q_t\}) \times \{U, D\}$, $q_0' = [q_s, D]$
$\Gamma' = \Gamma \cup \{\#\}$

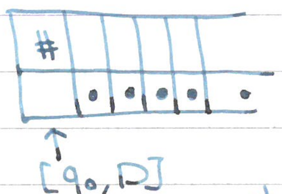
- the negative part of $\mathbb{Z}$ extends to the right on the upper tape



- q_s is used to place # on the upper tape at pos. 0
- q_t is used to go to state $[q_0, D]$
- # is used to indicate when the head crosses 0 and should go from U to D or D to U.

1 $\delta'([q_s, D], [B,B]) = [[q_t, D], [B, \#], R]$
2 $\delta'([q_t, D], [\gamma, B]) = [[q_0, D], [\gamma, B], L]$

$\Rightarrow$ we have



3 whenever $\delta(q_i, \gamma) = [q_j, \tilde{\gamma}, d]$, $\gamma \in \Gamma$, $x \neq \#$
$\delta'([q_i, D], [\gamma, x]) = [[q_j, D], (\tilde{\gamma}, x], d]$
$\delta'([q_i, U], [x, \gamma]) = [[q_j, U], [x, \tilde{\gamma}], d']$
where d' is the opposite direction of d.
(there are the "ordinary" translations of trans. in not-zero positions).

4 $\delta'([q_i, D], [x, \#]) = [[q_j, D], [y, \#], R]$, if $\delta(q_i, x) = [q_j, y, R]$
5 $\delta'([q_i, D], [x, \#]) = [[q_j, U], [y, \#], R]$, if $\delta(q_i, x) = [q_j, y, L]$
6 $\delta'([q_i, U], [x, \#]) = [[q_j, U], [y, \#], R]$, if $\delta(q_i, x) = [q_j, y, L]$
7 $\delta'([q_i, U], [x, \#]) = [[q_j, D], [y, \#], L]$, if $\delta(q_i, x) = [q_j, y, R]$

8.6.2 Diagrammatic notation for 2-track turing machines : since $\delta(q_i,[x,y]) = [q_j,[x',y'],d]$ we denote transitions as

$$q_i \xrightarrow[{[x|x', y|y', d]}]{} q_j$$

8.6.1 multitape T.M. has k tapes and k tape heads that can be independently moved at transitions, i.e δ is of the form

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L,R,S\}^k$$

for convenience we also allow heads to remain stationary (S), but this is no loss of generality, as we can always add transitions where a head moves R then L to emulate S.

transitions are drawn as $[x|x'd \quad y|y'd \ldots]$

thrm a multitape (precisely, a 2-tape) T.M can be simulated on a 5-track T.M.

proof track 1 & 3 keep the state of tape 1 & 2
track 2 & 4 keep the tape head's position of tape 1 & 2 respectively. They do this with auxiliary symbols $\#, X \in \Gamma' \setminus \Gamma$. There are directions $\{L, R, S, U\}$.
track 5 is used to reposition tape heads

U indicates unknown

- initially, we write # at pos. 0 on track 5 and X at pos. 0 on track 2 and track 4.

states Q' are 8-tuples of the form
$[s, q_i, x_1, x_2, y_1, y_2, d_1, d_2]$

s "status"
$q_i \in Q$
$x_i, y_i \in \Sigma \cup \{U\}$
$d_i \in \{L, R, S, U\}$

" $\delta(q_i, x_1, x_2) = [q_j, y_1, d_1, y_2, d_2]$

actions : M begins the simulation of a transition in state $[f_1, q_i, U, U, U, U, U, U]$ at position 0 on the tracks.

f_1 1) find first symbol : M' moves right until it reads the X on track 2. When this happens, it enters state $[f_1, q_i, x_1, U, U, U, U, U]$ where x_1 is the symbol on track 1 under X then M' returns to initial position. Here, # on track 5 is used to indicate the initial position.

f_2 2) find second symbol : the state $[f_1, q_i, x_1, U, U, U, U, U]$ is different from $[f_1, q_i, U, U, U, U, U, U]$, so M' "knows" it is in a different phase. Now it will find the 2nd tape ~~position~~ symbol by looking for X on track ~~3~~ 4 and reading x_2 from ~~tape~~ track 3 : enters $[f_2, q_i, x_1, x_2, U, U, U, U]$ then moves back to initial position

p_1 3) M' enters state $[p_1, q_j, x_1, x_2, y_1, y_2, d_1, d_2]$ where these are filled in based on our knowledge of $\delta(q_i, x_1, x_2) = [q_j, y_1, d_1, y_2, d_2]$ and the fact that we can uniquely define this transition. based on the state [. . .] and M' being at pos 0.

p_1 4) print symbol 1 : M' searches for X on track 2. it uses 1 transition, uniquely determined by the status p_1 and all info $x_1, y_1, d_1,$ contained in the state, to
1) move X on track 2
2) replace x_1 by y_1 on track 1
then it moves back to pos. 0 and changes status to p_2

p_2 5) print symbol 2 : analogous, for track 4, 3 and symbols x_2, y_2, d M' can "know" what to do because it sees that its status is p_2
6) move back to initial position & change state to $[f_1, q_j, U U U U U U]$.
wake up.

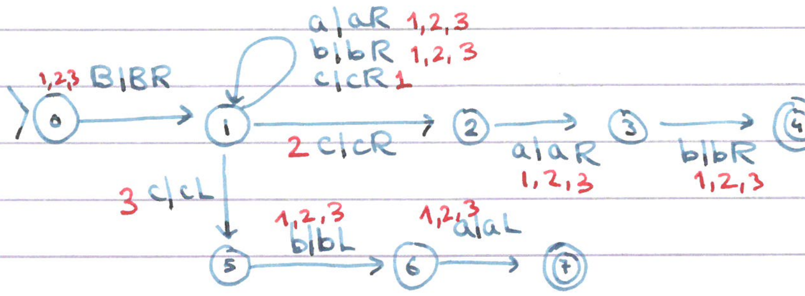
8.7.1 nondeterminism for Turing machines

we formalize "there being no unique outcome of a computation" with "sets of ~~out~~ transitions" rather than transitions.

that is, $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\})$
and δ need not be partial anymore since we can model "no transition" simply by setting $\delta(q_i, x) = \emptyset$

def ! a $s \in \Sigma^{<\omega}$ is accepted by a non-deterministic Turing machine if there is <u>at least one</u> trace of computation that terminates in F

ex. (Sudkamp, 8.7.1):



acepts any word which either contains (abc) or (cab) as substring.

— like with NFA and DFA, nondeterminism does not lead to the class of languages that are recognized being larger:

$\exists M'$ deterministic TM. accepts $\mathcal{L}$ $\iff$
$\exists M$ nondeterministi TM. that accepts $\mathcal{L}$.

where $\Rightarrow$ is obvious since $\delta(q,x) = \begin{cases} \emptyset & \text{if } \delta'(q,x)\uparrow \\ \{\delta(q,x)\} & \text{if } \delta(q,x)\downarrow \end{cases}$
~~the~~ makes every deterministic TM also a nondeterministic TM.

$\Leftarrow$ is more difficult : how do we simulate all the "traces such that an accepting trace is found in finite time ?

systematically ordering & interleaving the different traces of computation such that any finite = halting trace is encountered in finite time (assuming there is no abnormal termination in any of the traces) :

- For each state, symbol pair $(q_i, x) \in Q \times \Gamma$, $\delta(q_i,x) \neq \emptyset$ label order the alternatives from 1 to $n = \max_{(q_i,x) \in Q \times \Gamma} |\delta(q_i,x)|$. If (q_j, x) has fewer than n transitions, one transition is assigned remaining labels to complete the ordering. If $\delta(q_i,x) = \emptyset$, no labels are assigned. See next page for example 8.7.1 from Sudkamp.

see previous page for the same labels in the figure

state	Γ		δ	state	Γ		δ
q_0	B	1	q_1 B R	q_2	a	1	q_3 a R
		2	q_1 B R			2	q_3 a R
		3	q_1 B R			3	q_3 a R
q_1	a	1	q_1 a R	q_3	b	1	q_4 b R
		2	q_1 a R			2	q_4 b R
		3	q_1 a R			3	q_4 b R
q_1	b	1	q_1 b R	q_5	b	1	q_6 b L
		2	q_1 b R			2	q_6 b L
		3	q_1 b R			3	q_6 b L
q_1	c	1	q_1 c R	q_6	a	1	q_7 a L
		2	q_2 c R			2	q_7 a L
		3	q_5 c L			3	q_7 a L

only these are distinct. $\Rightarrow \max |\delta(q_i,x)| = 3$

a computation is then uniquely defined by

- an input word $w \in \Sigma^{<\omega}$
- a sequence $(m_1, m_2, \dots, m_k)$
  where m_i is the ~~number~~ label of
  the transition picked in the i-th step.

such a computation may not yet have halted, mind
or it may already have halted prematurely.

$\Rightarrow$ $(m_1 \dots m_k), w \mapsto$ computation is not an
injection or anything. It is a surjection,
and therefore useful in enumerating all traces

ex take machine 8.7.1 and word abc.
An accepting trace is

$$q_0 Babc \overset{1,2,3}{\vdash} Bq_1abc \overset{1,2,3}{\vdash} Baq_1bc \overset{1,2,3}{\vdash} Babq_1c$$
$$\overset{3}{\vdash} Ba\,q_5bc \overset{1,2,3}{\vdash} Bq_6abc \overset{1,2,3}{\vdash} q_7Babc$$

so any sequence in $\{1,2,3\}^3 \times \{3\} \times \{1,2,3\}^2$
will give this computation that acepts abc

Let M be a nondeterministic T.m that
accepts by halting.

M' has : $\Sigma' = \Sigma$ $\quad \Gamma' = \{x, \#x \mid x \in \Gamma\} \cup \{1, 2, \dots n\}$

M' is a three-tape deterministic T.m.

tape 1 : store input
tape 2 : copy input, simulate M, erase, repeat
tape 3 : hold sequence of form ~~[illegible]~~ $(m_1, \dots m_k)$

~~Computability~~

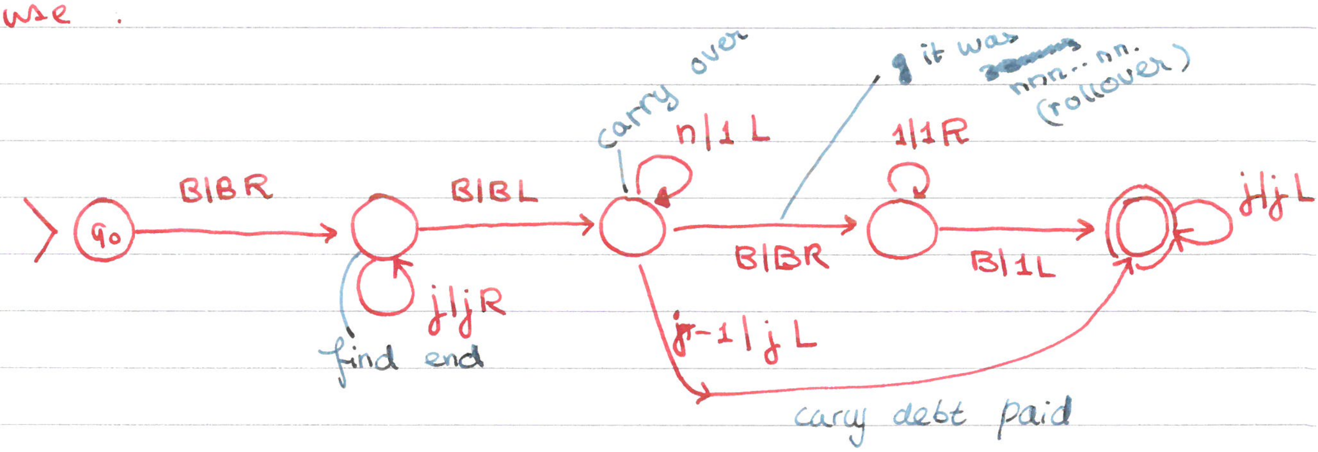
~~Practice exams~~ ~~(22-23)~~

M' does the following :

1) write a sequence $(m_1 .. m_k)$ on tape 3
2) copy input w from tape 1 to tape 2
3) simulate M on w on tape 2, guided by sequence on tape 3
4) if the simulation halts prior to finishing the sequence $(m_1 .. m_k)$, accept
5) if the sequence is finished generate the next sequence in lexicographical order and continue with 1) (after clearing tape 2)

The values on t.3 are generated in lexicographic order. This ensures that if a computation with guiding seq. $(m_1 .. m_k)$ does not halt, then all shorter computations did not halt either. It implies that if there is a halting computation, it will be reached in finite time.

to generate the next $(m_1 .. m_k)$ in lexicographic order, use .



M knows when to stop simulating by reading a blank on tape 3

M knows where the left boundary on tape 2 is by marking the B at pos 0 on tape 2 with #: use $\#B \in \Gamma'$

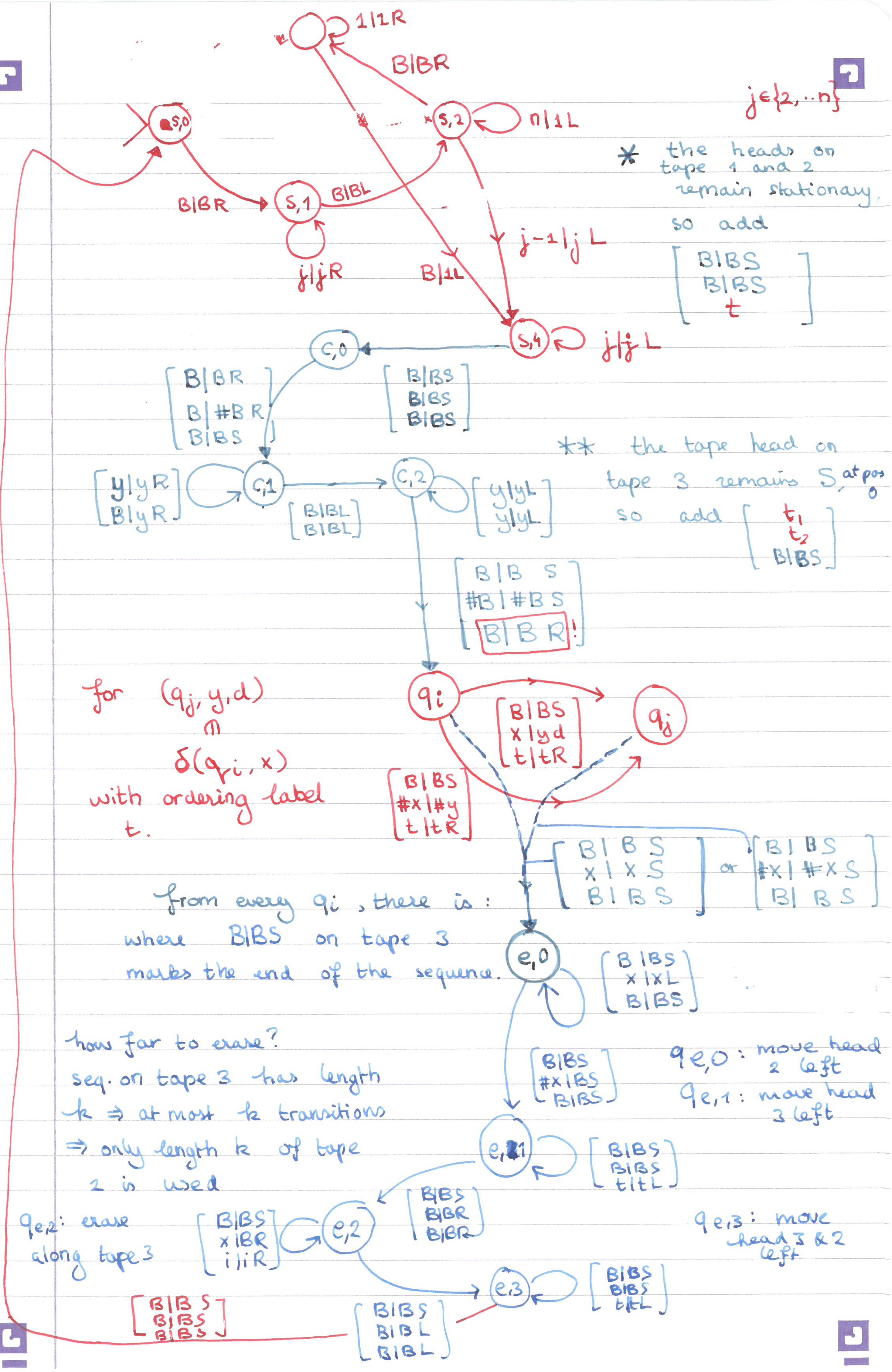
~~M has states~~

machine M', rigorously.

M' consists of phases
- generate $\boxed{s}$equence
- $\boxed{c}$opy
- simulate
- $\boxed{e}$rase

we therefore have states

$$Q_{M'} = Q_M \cup \{ q_{s,j} \mid j = 0,1,2,3,4 \} \cup \{ q_{c,j} \mid j = 0,1,2 \}$$
$$\cup \{ q_{e,j} \mid j = 0,1,2,3 \}$$



$j \in \{2, .. n\}$

* the heads on tape 1 and 2 remain stationary, so add $\begin{bmatrix} B|BS \\ B|BS \\ t \end{bmatrix}$

** the tape head on tape 3 remains S, at pos 0 so add $\begin{bmatrix} t_1 \\ t_2 \\ B|BS \end{bmatrix}$

for $(q_j, y, d) \in \delta(q_i, x)$ with ordering label t.

From every q_i, there is: where B|BS on tape 3 marks the end of the sequence.

how far to erase? seq. on tape 3 has length k ⇒ at most k transitions ⇒ only length k of tape 2 is used

$q_{e,0}$: move head 2 left

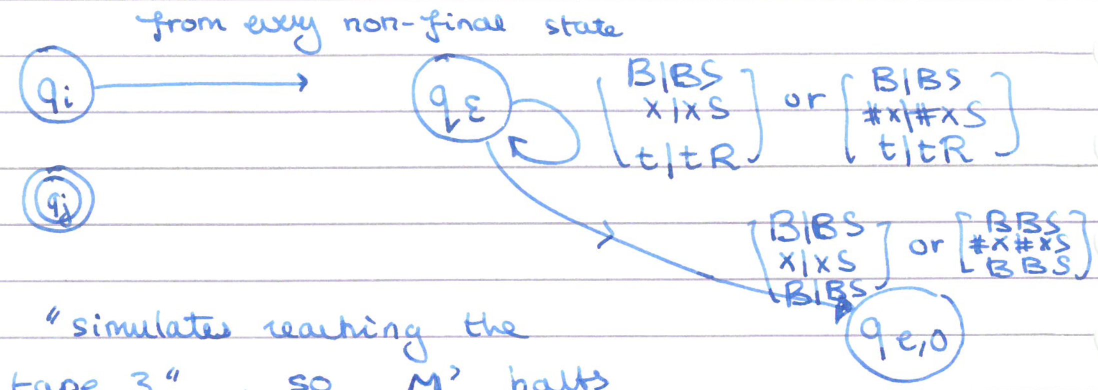
$q_{e,1}$: move head 3 left

$q_{e,2}$: erase along tape 3

$q_{e,3}$: move head 3 & 2 left

for a *NTM that accepts by final state, we add a state

(If every computation in a non-deterministic turing machine halts, so will every computation in the corresponding M' deterministic in this way)



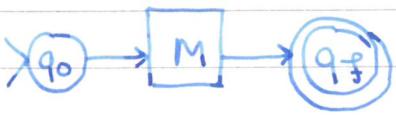
then q_ε "simulates reaching the end of tape 3", so M' halts
$\Leftrightarrow$ M halts with final state for some trace.

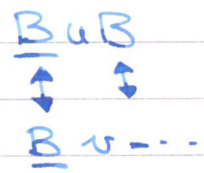
— This does not yet show that M' always halts. However, it is indeed a theorem that such an M' can be constructed (Sudkamp ex. 8.23)

$\Rightarrow$ if there is a halting NTM accepting L, then L is recursive.

9.1.1 a DTM one tape with single final state,
$M = (Q, \Gamma, \Sigma, \delta, q_0, q_f)$, computes a partial function
$\Sigma^{<\omega} \nrightarrow \Sigma^{<\omega}$ if

1) $\delta(q_0, x)\uparrow \iff x \neq B$ and $\delta(q_0, B) = [q_i, B, R]$
2) $\forall i \forall x, y, d\ \delta(q_i, x) \neq [q_0, y, d]$
3) $\delta(q_f, B)\uparrow$
4) $f(u) = v \iff q_0 B u \vdash_M q_f B v$
5) $f(u)\uparrow \iff q_0 B u \uparrow$

M is depicted as macro 

and often explained with diagram 

If f is n-ary $(n \geq 0)$, $f : (\Sigma^{<\omega})^n \nrightarrow \Sigma^{<\omega}$
then the arguments are placed on tape separated by a single B

a total k-ary function $r : (\Sigma^{<\omega})^k \to \{0,1\}$
defines a k-ary relation on $\Sigma^{<\omega}$.

9.2 A number-theoretic function has the form
$\mathbb{N}^k \nrightarrow \mathbb{N}$. We encode $x \in \mathbb{N}$ as the
string 1^{x+1} over the alphabet $\Sigma = \{1\}$

for a $L \subseteq \Sigma^{<\omega}$, the characteristic function
is a total function $\chi_L\ \Sigma^{<\omega} \to \{0,1\}$.

L is recursive $\iff$ there is a t.M. that computes χ_L

L is r.e $\iff$ there is a t.M that computes one of its partial characteristic functions where a partial char. function for $\mathcal{L}$ is a function $\hat{\chi}_{\mathcal{L}} : \Sigma^{<\omega} \nrightarrow \{0,1\}$ such that

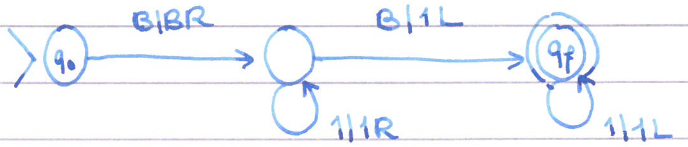
$\hat{\chi}_{\mathcal{L}}(w) = 1$ if $w \in \mathcal{L}$ and

$\hat{\chi}_{\mathcal{L}}(w) = 0$ or $\hat{\chi}_{\mathcal{L}}(w)\uparrow$ if $w \notin \mathcal{L}$.

---

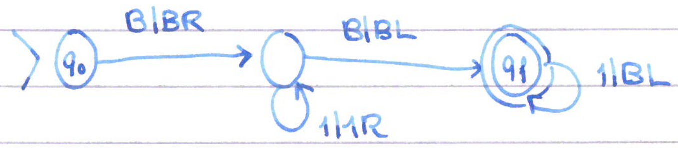
some macros:

S successor



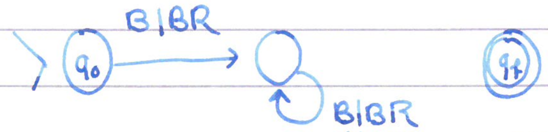
$s(n) = \text{"}n+1\text{"}$

Z zero



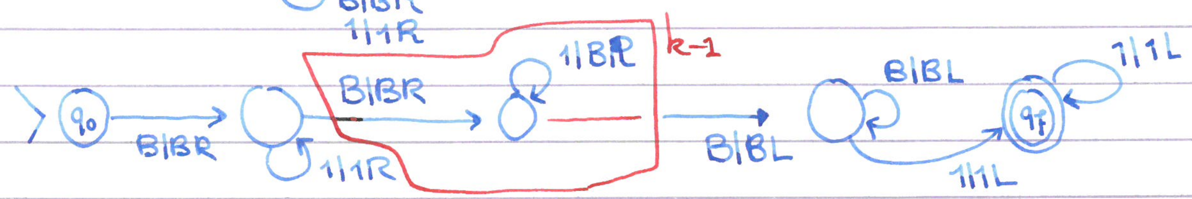
$z(n) = 0$

E empty



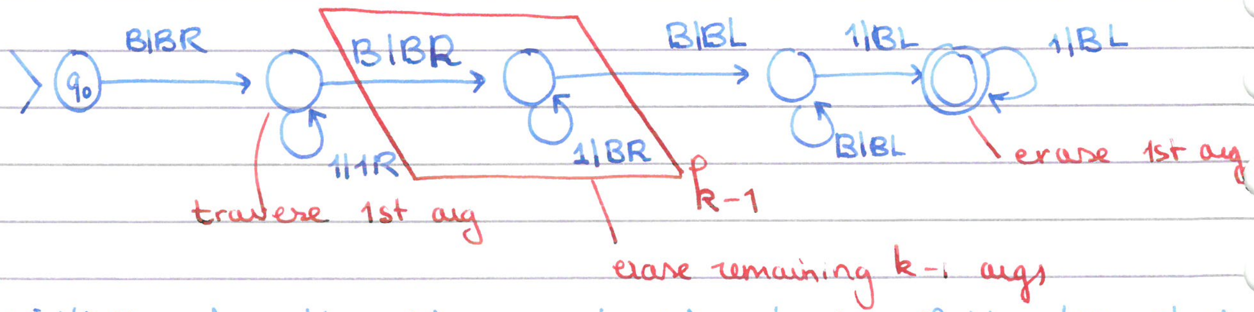
$e(n)\uparrow \quad \forall n$

$P_i^{(k)}$ project k to ith arg.



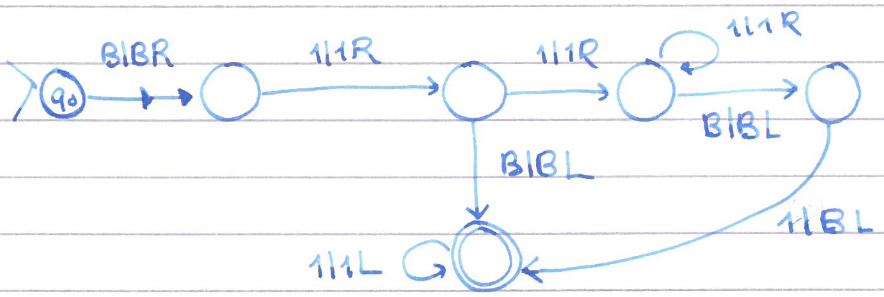
this is $P_1^{(k)}$. General $P_i^{(k)}$ are easiest defined using other macros, see below. $p_i^{(k)}(n_1 \dots n_k) = n_i$

E_k erase k args



we initially keep the 1st arg to keep track of the tape start.

D pred decrease



$\text{pred}(n) = n \dot{-} 1 = \begin{cases} 0 & n = 0 \\ n-1 & n > 0 \end{cases}$

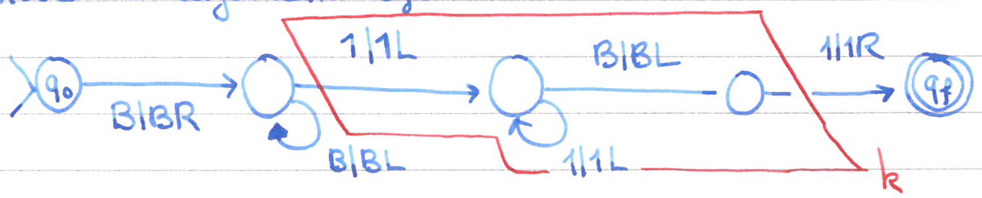
since there is only a transition B|BR out of q_0 and no B|■ - - out of q_f, we can concatenate macro's to compute compositions of computable functions

$\Rightarrow$ T-comp fn are closed under composition!

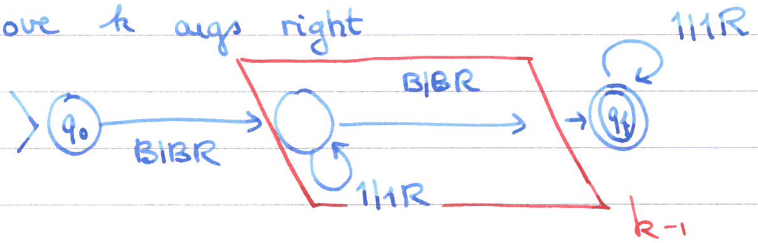
---

other macros :

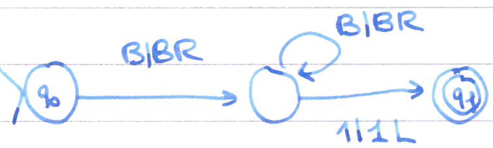
ML_k move k arguments left



MR_k move k args right



FR find right

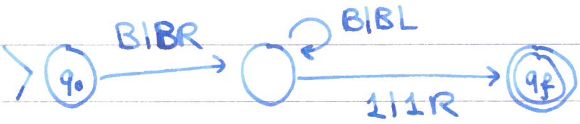


$\underline{B}\ B^i\ \bar{n}\ B$

$\updownarrow$

$B^i\ \underline{B}\ \bar{n}\ B$

FL find left

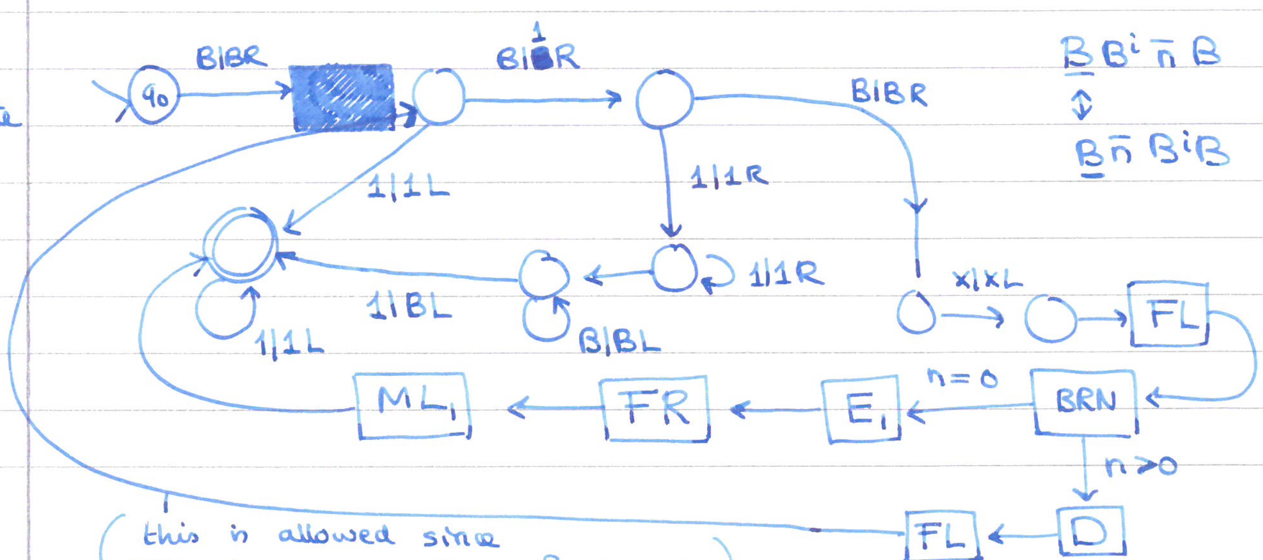


$B\ \bar{n}\ B^i\ \underline{B}$

$\updownarrow$

$B\ \bar{n}\ \underline{B}\ B^i$

T translate



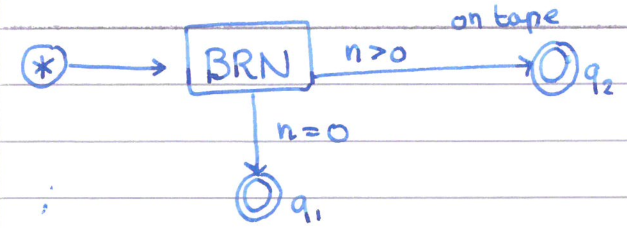
$\underline{B}\ B^i\ \bar{n}\ B$

$\updownarrow$

$\underline{B}\ \bar{n}\ B^i B$

(this is allowed since FL has no 1|- - from q_f)

BRN
branch



i.e.

$$\underline{B}\,\bar{0}\,B \quad\updownarrow\quad \underline{B}\,\bar{0}\,B \;\; (q_1) \qquad\qquad n>0:\ \underline{B}\,\bar{n}\,B \quad\updownarrow\quad \underline{B}\,\bar{n}\,B \;\; (q_2)$$

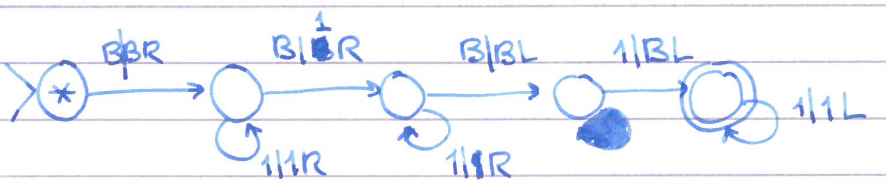
note we assume Erase(k) does not read any tape symbol beyond the first k arguments & trailing one B :

$$\underbrace{B\bar{n}_1 \cdots \bar{n}_k B}_{\text{only this}}.$$

Then $P_i^{(k)}$ is given by

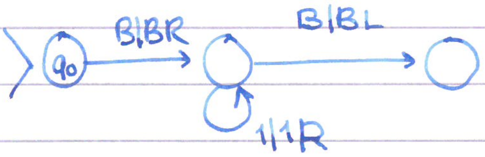
$$\rangle (*) \to \boxed{E_{i-1}} \to \boxed{T} \to \boxed{MR_1} \to \boxed{FR} \to \boxed{E_{k-i}} \to \boxed{FL} \to \boxed{ML_1} \to (*)$$

A
add



S

$$\text{sub}(n,m) = n \dot{-} m = \begin{cases} 0 & n < m \\ n-m & \text{otherwise} \end{cases}$$



def a number-theoretic function $\mathbb{N}^n \nrightarrow \mathbb{N}^k$ is Turing-computable if there is a Turing machine that computes it

9.4.3 thrm Since such T.M must satisfy points 1) – 5) of 9.1.1, they can be "composed".

- there is only a B|BR transition out of q_0
- there is no B|– – transition out of q_f

so "merging" q_0 of M_2 with q_f of M_1 gives a new machine $\rightarrow \boxed{M_1} \rightarrow \boxed{M_2} \rightarrow$ which is therefore deterministic and computes precisely $m_2 \circ m_1$

ex. MULT : by Peano arithmetic,

$n \cdot s(m) = (n \cdot m) + n$

$n \cdot 0 = 0$

approach :

$\_\ \overline{m}\ \ \overline{n}$

$\_\ \overline{m}\ \ \overline{n}\ \ \overline{0}$

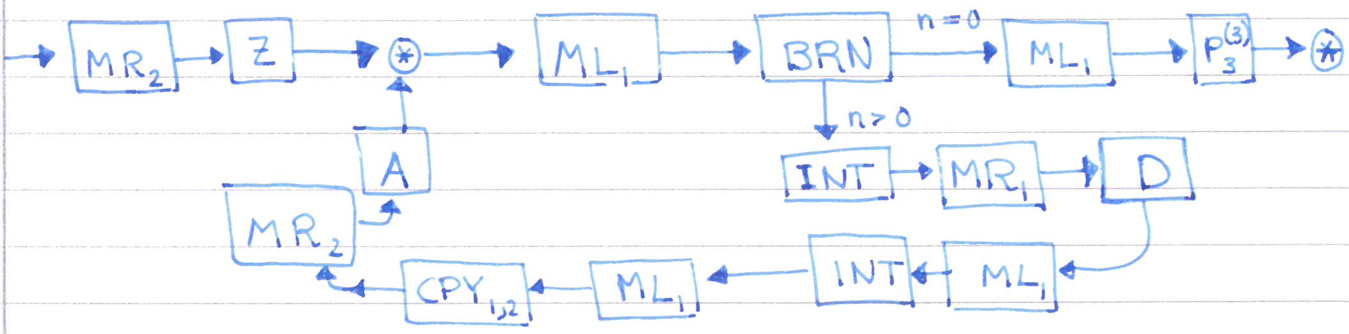
$\_\ \overline{m}\ \ \overline{n}\ \ \overline{a}$

$n > 0$

$\_\ \overline{m}\ \ \overline{n-1}\ \ \overline{a}\ \ \overline{m}$

$\_\ \overline{m}\ \ \overline{n-1}\ \ \overline{a+m}$

Therefore



def 9.4.1 Here, the (only natural) convention is that $f \circ g(n)\uparrow$ if $g(n)\uparrow$ or $f(g(n))\uparrow$

9.4.2 (Vectorized composition) If $g_1 .. g_n : \mathbb{N}^k \nrightarrow \mathbb{N}$ then we let $(g_1, .., g_n) : \mathbb{N}^k \nrightarrow \mathbb{N}$ through

$$(g_1 .. g_n)(\vec{x}) = \begin{cases} \uparrow & \text{if } \exists j \ \ g_j(\vec{x})\uparrow \\ (g_1(\vec{x}) .. g_n(\vec{x})) & \text{otherwise} \end{cases}$$

& in this way, we can compose :

$f \circ (g_1 .. g_n)$ where $g_i : \mathbb{N}^k \nrightarrow \mathbb{N}$

$f : \mathbb{N}^n \nrightarrow \mathbb{N}$

9.5.2 thrm A Turing machine is defined by its transition function, "δ" in Sudkamp. Such a transition function is determined by a finite set of instructions in $Q \times \Gamma \times Q \times \Gamma \times \{L,R\}$.

$\Rightarrow$ We can embed the set of Turing computable functions TC in $\mathcal{P}_{fin}(Q\times\Gamma\times Q\times\Gamma\times\{L,R\})$ when $\mathcal{P}_{fin}(X) = \{S \subseteq X \mid S \text{ finite}\}$.

$$TC \hookrightarrow \mathcal{P}_{fin}(Q\times\Gamma\times Q\times\Gamma\times\{L,R\})$$

it can be shown that $\mathcal{P}_{fin}(S) \cong S^*$ and if S is finite then $|S^*| = \omega$

$\Rightarrow$ $|TC| \leq \omega$

$\Rightarrow$ There are only countably many TC functions. But there are at least $|\mathbb{N}|^{|\mathbb{N}|} = \omega^\omega > \omega$ (partial) number-theoretic functions

$\Rightarrow$ There are many more incomputable functions than computable functions

13.1.- (explicit example: Ackermann's function)

i) $A(0,y) = y+1$

ii) $A(x+1,0) = A(x,1)$

iii) $A(x+1,y+1) = A(x, A(x+1,y))$

note: this is not primitive recursion, so you would need to prove existence first.

13.6.2 For every one-variable $f \in PR$, $\exists x$ $f(x) < A(x,x)$

13 (Primitive recursion) given a <u>well-order</u> $(L, \preceq)$ with limit elements $I \subset L$ and successors $s(x)$, $x \in L$,
If $g : L^n \times I \to S$ is a map and
$h : L^{n+2} \to S$ is a map (where S is any set)
then there is a unique map $f : L^{n+1} \to S$ s.t

$$\forall \vec{x} \in L^n \ \forall \ell \in I \qquad f(\vec{x}, \ell) = g(\vec{x}, \ell)$$
$$\forall \vec{x} \in L^n \ \forall y \in L \qquad f(\vec{x}, s(y)) = h(\vec{x}, y, f(\vec{x}, y))$$

f is "defined" by primitive recursion from g, h.

proof uniqueness: suppose $f, \tilde{f}$ both suffice, but
$f(\vec{x}, y) \neq \tilde{f}(\vec{x}, y)$, where we assume wlog that
$y = \min \{ \tilde{y} : \exists \vec{x} \in L^n \ f(\vec{x}, \tilde{y}) \neq \tilde{f}(\vec{x}, \tilde{y}) \}$

- if y is a limit element, clearly $f(\vec{x}, y) = g(\vec{x}, y) = \tilde{f}(\vec{x}, y)$
- if not, $y = s(z)$, $z \in L$, and $f(\vec{x}, z) = \tilde{f}(\vec{x}, z)$
  $\Rightarrow f(x, s(z)) = h(x, z, f(\vec{x}, z)) = h(x, z, \tilde{f}(x, z)) = \tilde{f}(\vec{x}, z)$

$\lightning$ so f is unique.

existence: for $y \in L$, we can define $f_y : L^n \times L_{\preceq y} \to S$
By
- $f_\ell(\vec{x}, \ell) = g(\vec{x}, \ell)$ for limit ℓ
- $f_{s(y)}(\vec{x}, u) = \begin{cases} f_y(\vec{x}, u) & \text{if } u \prec y \\ h(\vec{x}, y, f_y(\vec{x}, y)) & \text{if } u = y \end{cases}$

The definition of f_y does not involve a recursive equation. Moreover, we see $f_y \subseteq f_z$ for $y \preceq z$, so $\bigcup_{z \in L} f_z$ is a function, and it is defined on $L^n \times L$.

$\Rightarrow f = \bigcup_{z \in L} f_z$ is our unique and required fnc. $\square$

def (Prim. rec. func. set) The set PR constitutes the smallest subset of functions $\mathbb{N}^n \to \mathbb{N}$ such that
1) $z = c_0^{(1)} \in PR$, $s : x \mapsto "x+1" \in PR$, $p_i^{(n)} \in PR$
2) PR is closed under (vectorized) composition
3) PR is closed under primitive recursion.

vbd $add(n,0) = n$, $add(n, m+1) = add(n,m)+1$

vbd $mult(n,0) = 0$, $mult(n, m+1) = mult(n,m)+n$

thrm 13.1.3 Every $f \in PR$ is Turing-computable.

proof We already have macros for $Z, S, P_i^{(n)}$, and we know that compositions of T.c fnc. are T.c.
We only need to show that a fnc. defined through primitive recursion, can be effectively computed.

We show this for $\vec{x} \in \mathbb{N}$. The case $\vec{x} \in \mathbb{N}^n$ is analogous

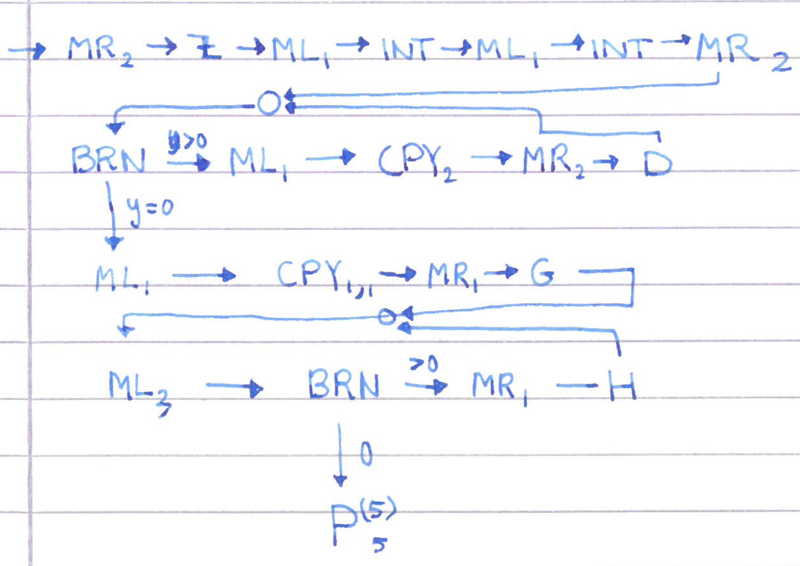
Start : 1) $\underline{B} \vec{x} B \vec{y}$

2) $\underline{B} \vec{0} \underline{B} \vec{x} B \vec{y}$ $y>0$

$y=0$ 3) $B \vec{0} B \vec{x} B \vec{y} \underline{B} \vec{x} B \overline{y-1}$

4) call g; ML_2 $B \vec{0} B \vec{x} B \vec{y} \cdots \underline{B} \vec{x} B \overline{g(0) = f(\vec{x}, 0)}$

5) 0? ML call h; ML



This writes a call "stack" on the tape, so to speak.

There are other versions that do not expand a stack first, but simply keep a counter.

This proves that primitively recursively defined fnc. are Turing computable $\square$

§13.2 an overview of useful PR functions:

table 13.1

add	$= (+)$	$x+0 = x \quad x+(y+1) = (x+y)+1$
mult	$= (\cdot)$	$x \cdot 0 = 0 \quad x \cdot S(y) = (x \cdot y) + x$
pred		$\text{pred}(0) = 0 \quad \text{pred}(x+1) = x$
sub	$= (\dot{-})$	$\text{sub}(n, 0) = n$, $\text{sub}(n, S(m)) = \text{pred}(\text{sub}(n,m))$
exp	$=$ ^	$\exp(x,0) = 1 \quad \exp(x, S(y)) = x \cdot \exp(x,y)$
const_y		$\text{const}_y(x) = y$

def 13.2 — A number-theoretic predicate is a number-theoretic fnc. with range $\{0,1\}$.

table 13.2

sgn	$\text{sgn}(0) = 0 \quad \text{sgn}(y+1) = 1$
cosgn	$\text{cosgn}(0) = 1 \quad \text{cosgn}(y+1) = 0$
lt	~~lt(x,0) = 0 lt(x,y+1) =~~ $\text{sgn}(x \dot{-} y)$
gt	$\text{sgn}(y \dot{-} x)$
eq	$\text{cosgn}(\text{lt}(x,y) + \text{gt}(x,y))$
neq	$\text{sgn}(\text{lt}(x,y) + \text{gt}(x,y))$

logical operators : not p_1 "$\equiv$" $\text{cosg} \circ p_1$

$p_1 \wedge p_2$ "=" $p_1 \cdot p_2 = \text{mult} \circ (p_1, p_2)$

$p_1 \vee p_2$ "=" $\text{sgn}(p_1 + p_2) = \text{sgn} \circ \text{add} \circ (p_1, p_2)$

13.2.1 thrm — if $g \in PR$ and $f : \mathbb{N}^n \to \mathbb{N}$ with $g(\vec{x}) \neq f(\vec{x})$ for a finite set $\{\vec{x}_1 \dots \vec{x}_n\} \subseteq \mathbb{N}$. Then $f \in PR$

proof

$$f(\vec{x}) = \left[\prod_{i=1}^{n} \text{ne}(\vec{x}, \vec{x}_i)\right] \cdot g(\vec{x}) + \sum_{i=1}^{n} \text{eq}(\vec{x}, \vec{x}_i) \cdot \underbrace{f(\vec{x}_i)}_{\text{constant}}$$

This is clearly a "finite" composition of PR functions □

13.2.2 thrm — Permuting variables preserves PR.

proof — $\sigma \in \Pi_n$. Then $\sigma = \left(p^{(n)}_{\sigma^{-1}(1)} \dots p^{(n)}_{\sigma^{-1}(n)}\right)$

so $\sigma \in PR$. It follows $f \circ \sigma \in PR$, $\forall f \in PR$, by def □

§13.3

13.3.1 thrm If $g : \mathbb{N}^n \times \mathbb{N} \to \mathbb{N}$ is PR then

i) $f(\vec{x}, y) = \sum_{i=0}^{y} g(\vec{x}, i)$

ii) $f(\vec{x}, y) = \prod_{i=0}^{y} g(x, i)$ are PR.

proof i) $f(\vec{x}, 0) = 0$

$f(\vec{x}, y+1) = f(\vec{x}, y) + g(\vec{x}, y+1)$

ii) $f(\vec{x}, 0) = 1$

$f(\vec{x}, y+1) = f(\vec{x}, y) \cdot g(\vec{x}, y+1)$

13.3.2 If $\ell : \mathbb{N}^n \to \mathbb{N}$, $u : \mathbb{N}^n \to \mathbb{N}$ are PR, then

i) $f(\vec{x}) = \sum_{i=\ell(\vec{x})}^{u(\vec{x})} g(\vec{x}, i)$ ii) $f(\vec{x}) = \prod_{i=\ell(\vec{x})}^{u(\vec{x})} g(\vec{x}, i)$ are PR

proof i),ii) assure $\ell(\vec{x}) \leq u(\vec{x})$ with predicate

$W(\vec{x}) := lt(\ell(\vec{x}), u(\vec{x}))$ (wrong)

$G(\vec{x}) := ge(\ell(\vec{x}), u(\vec{x}))$ (good)

define $\tilde{g}(\vec{x}, y) = g(\vec{x}, \ell(\vec{x}) + y)$

this is PR since in the base, we depend only on g, which is another fnc than $\tilde{g}$, and in the recursive case we depend only explicitly on $g(\vec{x}, y)$

$\tilde{g} \in PR, \ell, u \in PR$ gives $\vec{x} \mapsto \sum_{i=0}^{u(\vec{x})} \tilde{g}(\vec{x}, y + u(\vec{x}) \dot{-} \ell(\vec{x})) \in PR$

To ensure that $f = \tilde{g}$ only when $u(\vec{x}) \geq \ell(\vec{x})$

we use W and G.

$\tilde{f}(\vec{x}, y) = \sum_{i=0}^{y} \tilde{g}(\vec{x}, i) \in PR$ (thrm 13.3.1)

$\tilde{\tilde{f}}(\vec{x}, y) = \prod_{i=0}^{y} \tilde{g}(x, i) \in PR$

$\Rightarrow$ i). $f(\vec{x}) = W(\vec{x}) \cdot 0 + G(\vec{x}) \cdot \tilde{f}(\vec{x}, u(\vec{x}) \dot{-} \ell(\vec{x}))$

ii). $f(\vec{x}) = W(\vec{x}) \cdot 1 + G(\vec{x}) \cdot \tilde{\tilde{f}}(\vec{x}, u(\vec{x}) \dot{-} \ell(\vec{x}))$ are PR

□

def 13.3.– (Bounded minimization) let $p : \mathbb{N}^{n+1} \to \mathbb{N}$ be PR and $p(\vec{x}, y) \in \{0,1\}$ $\forall \vec{x}\ \forall y$.

$$\mu z \le y . [\, p(\vec{x}, z) \,] = \begin{cases} \min \{ z \in \mathbb{N} \mid z \le y,\ p(\vec{x}, z) = 1 \} \\ y+1 \quad \text{if the above set is } \emptyset . \end{cases}$$

thrm 13.3.3 $f(\vec{x}, y) := \mu z \le y . [\, p(\vec{x}, z) \,]$ is PR if p is.

proof

$$f(\vec{x}, 0) = 0 \text{ if } p(\vec{x}, 0) \text{ else } 1$$
$$= \text{cosg}(\, p(\vec{x}, 0))$$
$$f(\vec{x}, y+1) = f(\vec{x}, y) \text{ if } f(\vec{x}, y) \neq y+1$$
$$\text{else } (\, y+1 \text{ if } p(\vec{x}, y+1)$$
$$\text{else } y+2\,)$$
$$= ne(\, f(\vec{x}, y), y+1)\, f(\vec{x}, y) +$$
$$eq(\, f(\vec{x}, y), y+1) \cdot (\, p(x, y+1) \cdot (-1) + y+2\,)$$

clearly PR.

$\vec{x} \mapsto \text{cosg}(p(\vec{x}, 0))$ is our base fnc.

The recursive fnc is only dependent explicitly on y, $f(\vec{x}, y)$ and $\vec{x}$.