CSCE 355
4/10/2023

Church-Turing thesis: (1)

Turing Machines capture the (agreed upon) intuitive notion of computation/algorithm.

TM semantics: Fix a TM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$

Def: An instantaneous description (ID) or configuration of M is a string of the form

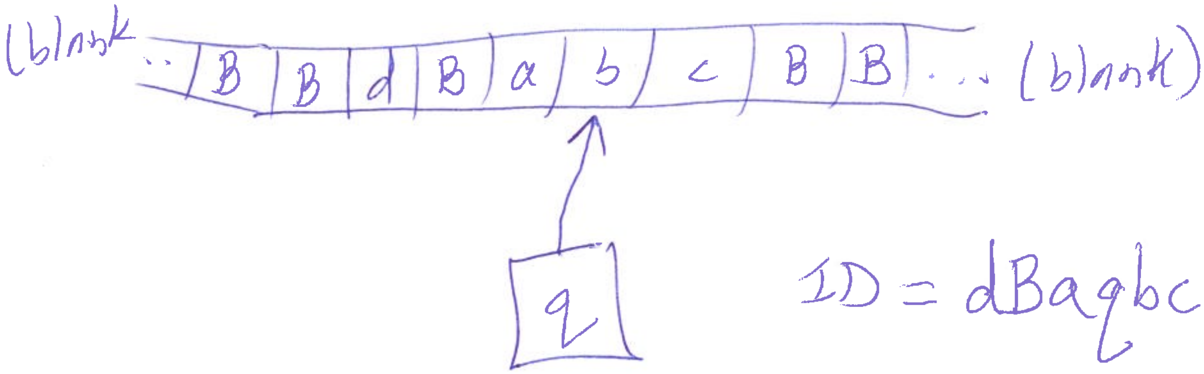
$$\underbrace{\alpha q \beta}_{ID} \quad \text{where } \alpha, \beta \in \Gamma^* \text{ and } q \in Q$$

[recall $Q \cap \Gamma = \emptyset$ and $ID \in (Q \cup \Gamma)^*$ where exactly one symbol of ID is in Q]

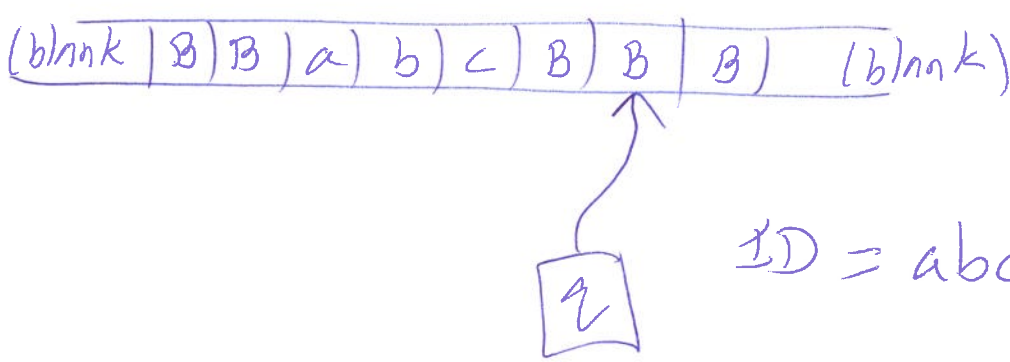
Meaning of $\alpha q \beta$:

1. M's current state is q,
2. $\alpha\beta$ is a contiguous portion of the current tape contents that is long enough to include all nonblank symbols.
3. The scanned cell corresponds to the

first symbol of β (if $\beta \neq \varepsilon$)
[if $\beta = \varepsilon$, then the head is scanning one cell past the right of α, and this cell & all cells to the right are blank.]



$ID = dBaqbc$



$ID = abcBq$

<u>Convention</u>: Free to add or remove B symbols from the beginning or end of an ID, to get the same ID, so

$$ID = dBaqbc = BBdBaqbcBB = \ldots$$

<u>Def</u>: Let ID_1 & ID_2 be ID's of M. Say that ID_2 is the <u>immediate successor</u>

of ID_1 ($ID_1 \vdash ID_2$) if, say, either

$$ID_1 = \alpha q a \gamma \quad (\alpha, \gamma \in \Gamma^*, \; a \in \Gamma, \; q \in Q)$$

and ~~then with~~ $\delta(q,a) = (r, b, \rightarrow)$ for some $r \in Q$ and $b \in \Gamma$, and

$$ID_2 = \alpha b r \gamma,$$

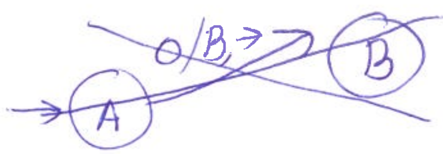
or, $ID_1 = \alpha c q a \gamma$ ($\alpha, \gamma \in \Gamma^*$; $a, c \in \Gamma$; $q \in Q$)

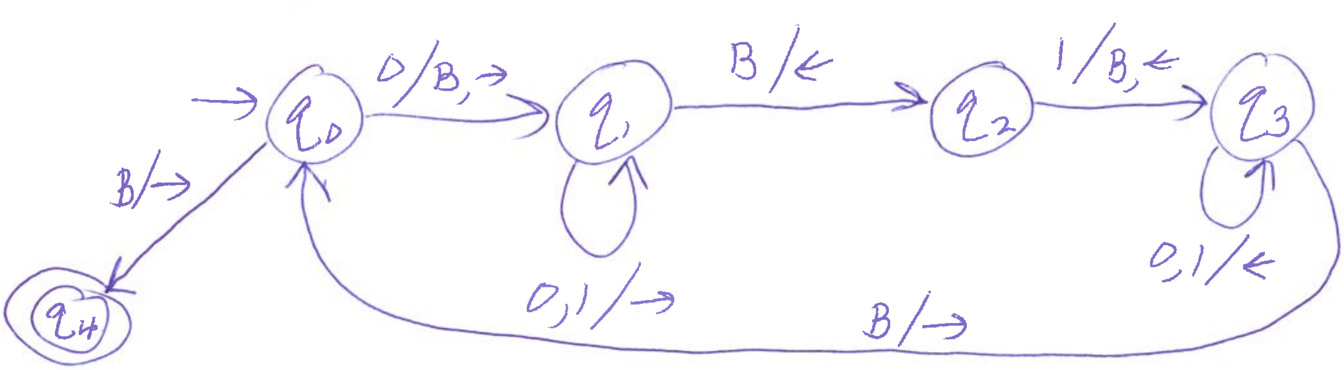
and $\delta(q,a) = (r, b, \leftarrow)$ (some $r \in Q$, $b \in \Gamma$)

and

$$ID_2 = \alpha r c b \gamma.$$

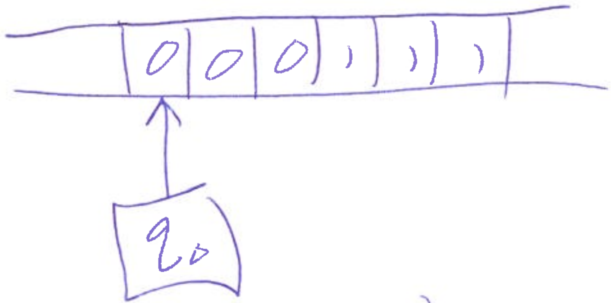
Ex:





Input: 000111

Initially:



$$q_0 000111 \vdash \underbrace{(Bq_1 00111 =)}_{\text{optional}} q_1 00111 \vdash 0q_1 0111$$

$$\vdash 00q_1 111 \vdash 001q_1 11 \vdash 0011q_1 1 \vdash 00111q_1$$

$$\underbrace{(= 00111q_1 B)}_{\text{optional}} \vdash 0011q_2 1 \vdash 001q_3 1$$

$$\vdash 00q_3 11 \vdash 0q_3 011 \vdash q_3 0011 \vdash q_3 B0011$$

$$\vdash q_0 0011 \vdash \cdots \vdash q_0 01 \vdash q_1 1 \vdash 1q_1$$

$$\vdash q_2 1 \vdash q_3 \vdash q_0 \vdash \underbrace{q_4}_{\text{final ID (accepting)}}$$

~~Def: Given~~ __Note__: Every ID of M has at most one immediate successor (no successors iff δ is undefined for that ID).

__Def__: M a TM as above, $w \in \Sigma^*$. The __initial ID of M on input w__ is $\boxed{q_0 w}$.

The <u>computation (path) of M on input w</u>

is the sequence

$$ID_0 \vdash ID_1 \vdash ID_2 \vdash \cdots$$

such that $ID_0 = q_0 w$ is the initial ID,

$$ID_i \vdash ID_{i+1} \quad \text{for } i = 0, 1, 2, \ldots$$

The computation may be <u>finite</u>, if

ID_k has no successor (for some k)

(then ID_k is the <u>final</u> ID), or infinite:

ID_i always has a successor for all $i = 0, 1, 2, \ldots$

We say that M <u>accepts</u> w if its computation on w is finite, and the final ID is $\alpha q \beta$ for some $q \in F$ ($\alpha, \beta \in \Gamma^*$ arbitrary).

Say that M <u>rejects</u> w if its computation is finite, and the final ID is $\alpha q \beta$ for some $q \in Q \setminus F$.

If computation is infinite, we say that M <u>loops</u> on input w.

<u>Def</u>: M a TM as above. The <u>language recognized</u> by M is

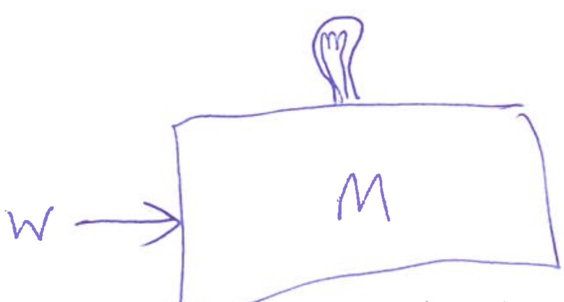
$$L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$$

<u>Note</u>: If $w \notin L(M)$, then M may either reject w or loop on w.

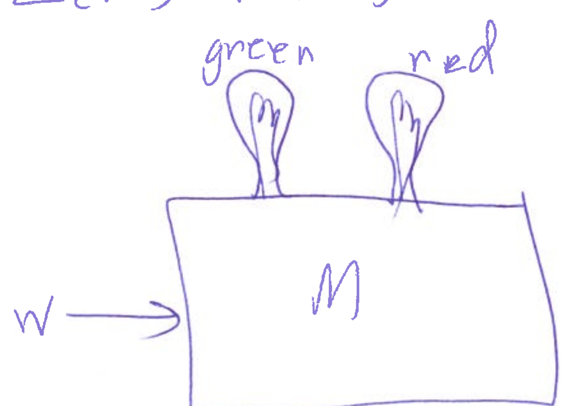
<u>Def</u>: M is <u>total</u> (or a <u>decider</u>) if M either $\underbrace{\text{accepts or } \cancel{\text{re}} \text{ rejects}}_{\text{halts on}}$ every input.

M <u>decides</u> L(M) in this case.

<u>Def</u>. Let L be a language. L is Turing-recognizable if $L = L(M)$ for some M.
L is <u>decidable</u> if $L = L(M)$ for some decider M.



recognizer



decider

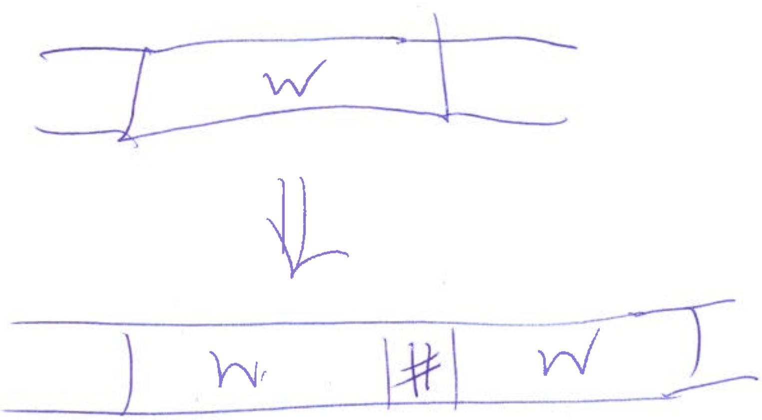
TM abilities:

— Copying a binary string.

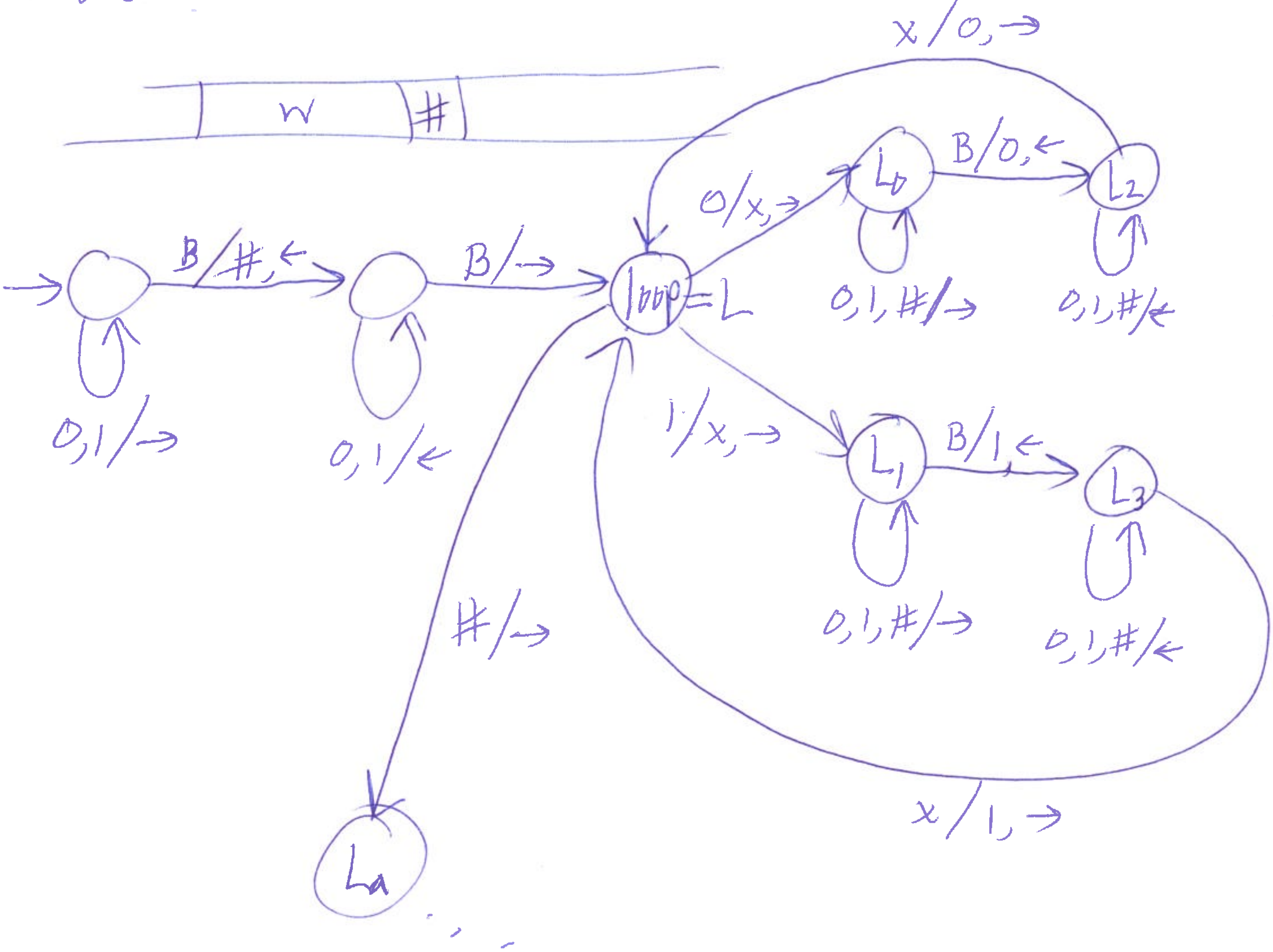
$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, \#, x, B\}$

B = blank symbol



Idea:



CSCE 355

4/12/2023

Ex: $w = \overset{\uparrow}{0}11\#$ (TM from last time) ①

$L011\# \vdash xL_0 11\# \vdash x1L_0 1\# \vdash x11L_0 \#$

$\vdash x11\#L_0 B \vdash x11L_2 \#0 \vdash x1L_2 1\#0 \vdash xL_2 11\#0$

$\vdash L_2 x11\#0 \vdash 0L11\#0 \vdash 0xL_1 1\#0 \vdash 0x1L_1 \#0$

$\vdash 0x1\#L_1 0 \vdash 0x1\#0L_1 \vdash 0x1\#L_3 01$

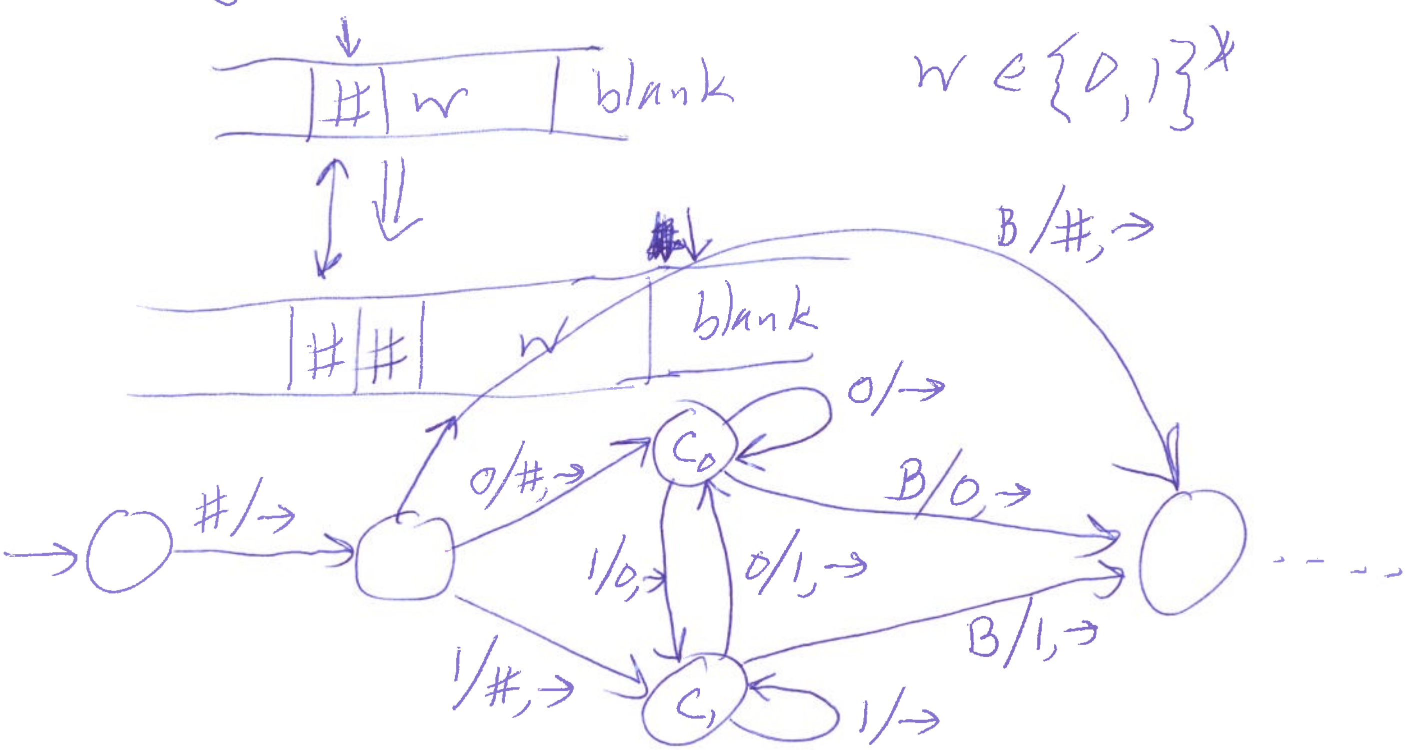
$\vdash 0x1L_3 \#01 \vdash 0xL_3 1\#01 \vdash 0L_3 x1\#01$

$\vdash 01L1\#01 \vdash 01xL_1 \#01 \vdash \ldots \vdash 01L_3 x\#011$

$\vdash 011L\#011 \vdash 011\#L_a 011 \ldots$

Moving data on the tape

$w \in \{0,1\}^*$

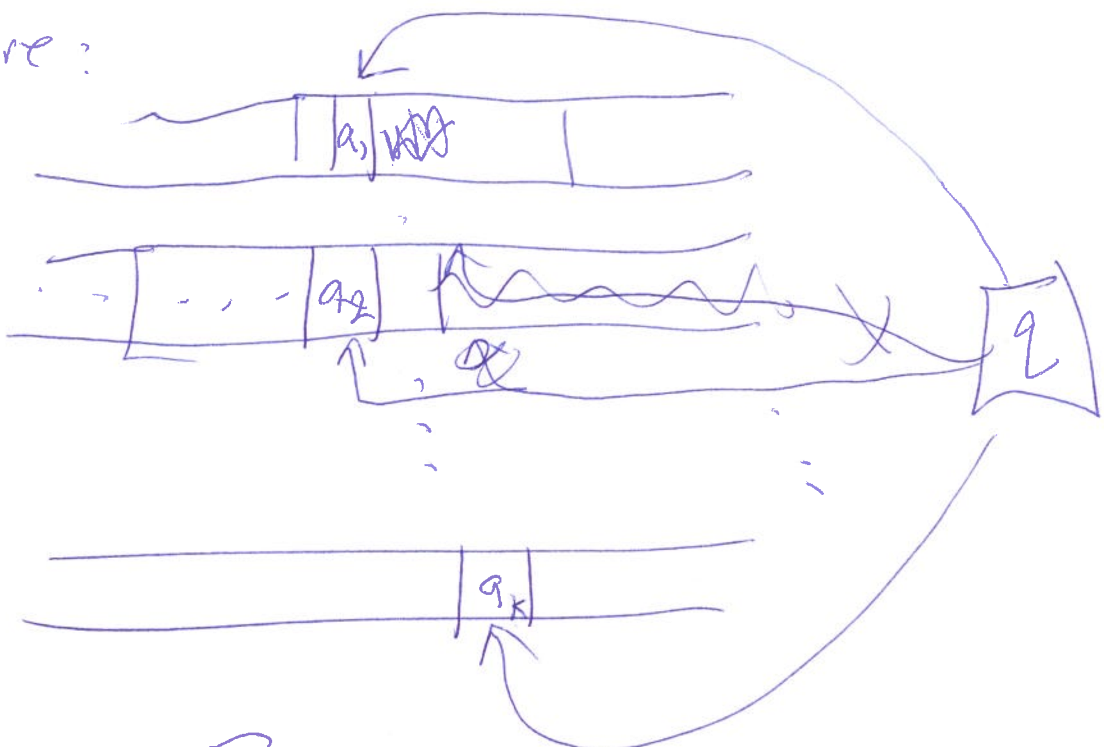


Multitape TMs

<u>Definition</u>: Fix an integer $k \geq 1$. A k-tape TM is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$ where everything is as before except

δ is a partial function mapping elements of $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{\leftarrow, \downarrow, \rightarrow\}^k$

Picture:



$a_1, \ldots, a_k \in \Gamma$ $\qquad r \in Q$

$$\delta(q, (a_1, \ldots, a_k)) = (r, (b_1, \ldots, b_k), (d_1, \ldots, d_k))$$

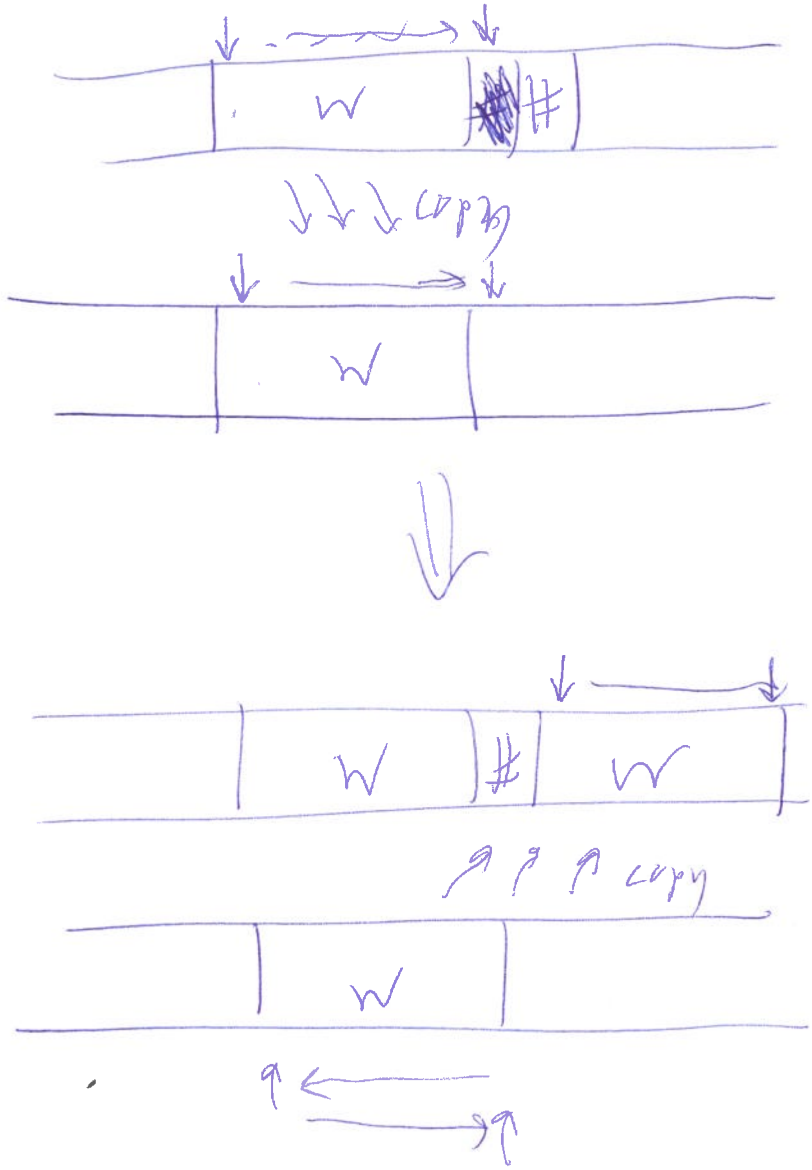
$b_1, \ldots, b_k \in \Gamma$

$d_1, \ldots, d_k \in \{\leftarrow, \downarrow, \rightarrow\}$

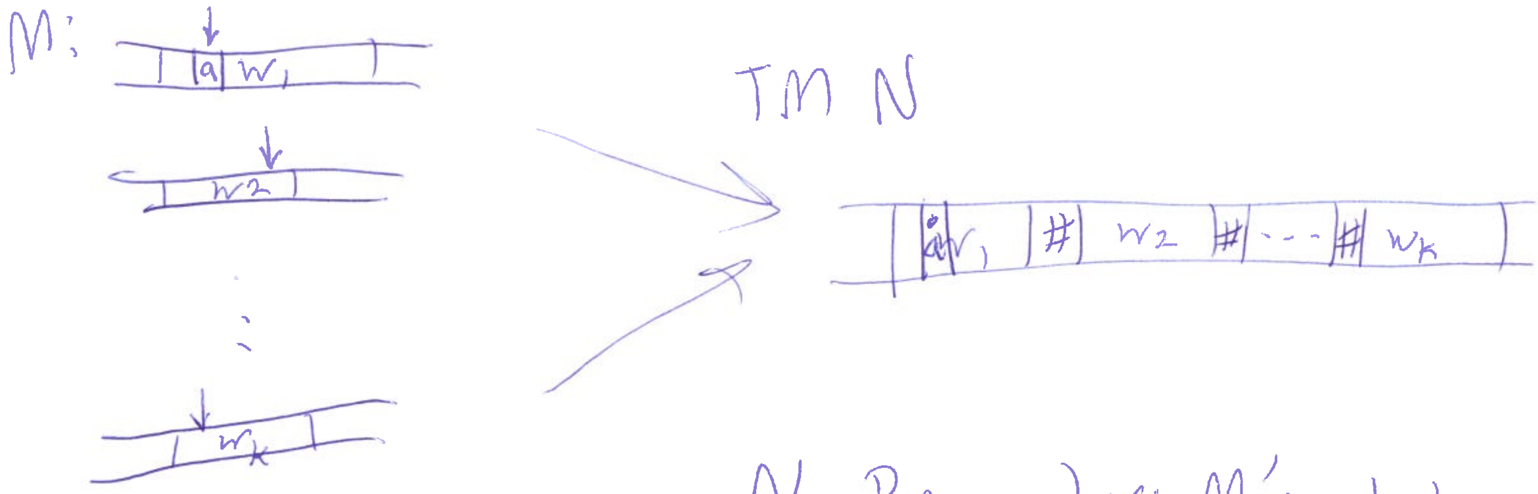
$\leftarrow$ = move left
$\rightarrow$ = move right
$\downarrow$ = stay put

Initially: - Input is on tape 1
- 1st head scanning leftmost symbol of input (if there is one)
- ~~Oth~~ Other tapes all blank.

Copying with a 2-tape machine:



Simulating a k-tape TM with a 1-tape (standard) TM M:



N Remembers M's state in its own state

M is k-tape TM
N is the simulating 1-tape TM

N has marked versions of M's tape symbols:

M's tape alphabet $= \{0, 1, \ldots\}$
N's " " $= \{0, \dot{0}, 1, \dot{1}, \ldots\}$
$\dot{a}$ = "marked" version of a

Each w_i on N's tape includes a single marked ~~sy~~ symbol, in the position in w_i that is being scanned by M's i tape head

To simulate a single step of M:

N can expand as M does, by bumping symbols to the right to make room

Pass 1:

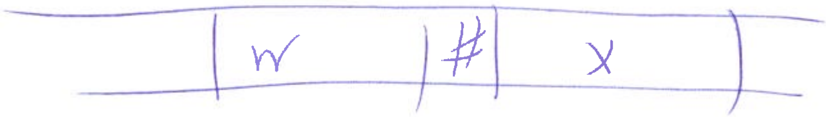
N's head sweeps left to right, picking up which symbols are marked, and remembering them in its state.

Pass 2: based on M's state and k-tuple of marked symbols, sweep the head right to left, simulating M's action in the vicinity of each marked symbol.

Upshot: If you can do something with, say, 3 tapes, you can do it with only one tape.

From now on, assume as many tapes (a fixed number) as is convenient.

Comparing two numbers in binary

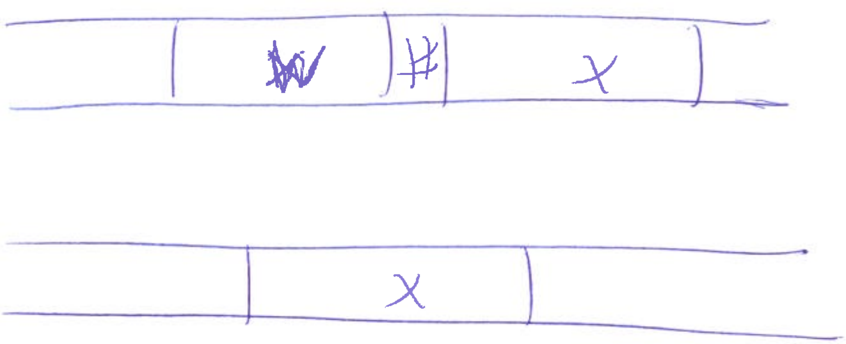


w, x binary strings.

is $\underline{w} < \underline{x}$, $\underline{w} > \underline{x}$, $\underline{w} = \underline{x}$?

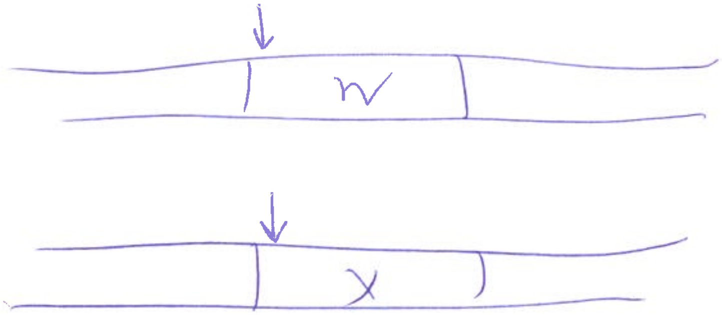
($\underline{w}$ is the natural number rep by w in binary
$\underline{x}$ - - - - - - - - x - - - -)

2 tapes:



if $|w| \neq |x|$, then pad the shorter string with leading 0's to make it the same length

Assume $|w| = |x|$



starting at the left, scan both heads right looking for unequal digits.

If all equal, then $\underline{w} = \underline{x}$

If 0 in w and 1 in x, then $\underline{w} < \underline{x}$

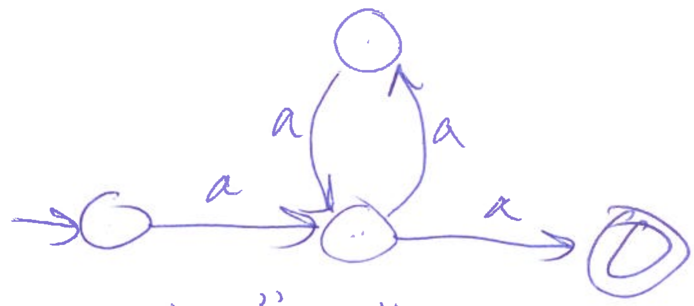
" 1 " " 0 " ", " $\underline{w} > \underline{x}$.

Formal description — transition diagram

Implementation-level description — general desc. of TMs actions, head movements, tape contents, etc.

High-level description — algorithm, model independendent.

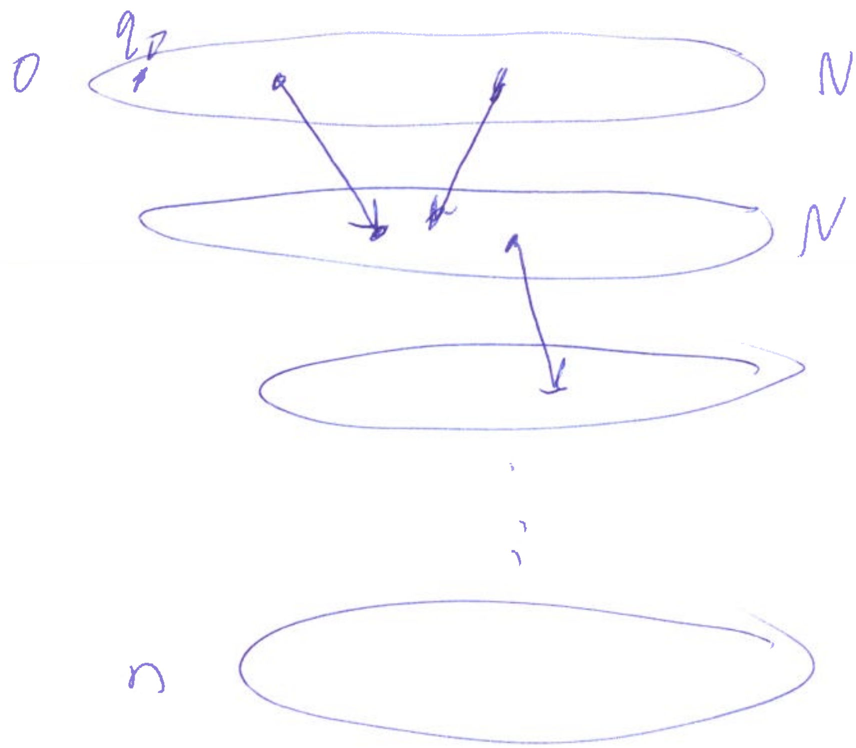
CSCE 355
4/17/2023



Accepts "aaaa"

N
NFA $\langle Q, \ldots, \delta \ldots \rangle$

Input string w, $|w| = n$.

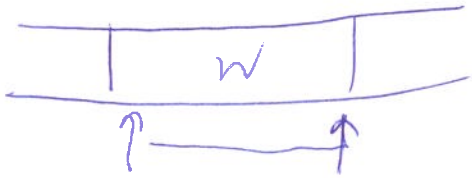


Accept the Church-Turing thesis

TMs = algorithms

Loose end: Increment, decrement

<u>Increment:</u>



$w \in \{0,1\}^*$

- move head to right end of w
- while seeing a 1, change to 0 and move left ("carry")
- change 0 (or B) to 1 and stop.

Done.

Decrement:

- If $w = 0\cdots0$, stop.
- Else
  - starting from right;
  - ~~while~~ while seeing a 0, change to 1 and move left ("borrow")
  - change 1 to 0.

Done

To add x ~~to~~ to y:

while $x \neq 0$:
    decrement x
    increment y

new value of y is sum, & $x=0$

[inefficient!]

Multiplication:

To multiply x by y:

prod := 0
while $x \neq 0$:
  decrement x
  add y to prod
end while.

"Forget TMs", work with algos:

M := "On input w:
(TM)
1. :
2. :
:
3.
:
"

Universal TM

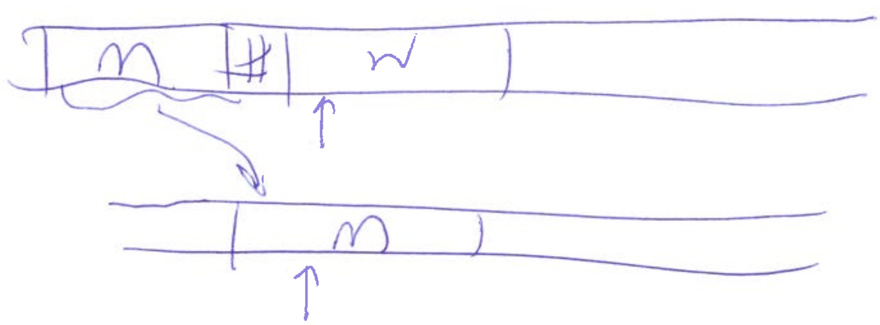
N := "On input M, where M is a TM:
1. Run M for 10 steps"

U := "On input M#w, where M is a TM, and w is a string over M's input alphabet:

1. Simulate M on input w"
[and do whatever M does.]

If M accepts w, then U accepts M#w
" " rejects ", " " rejects "
" " loops on " " " loops on "

U



U^c := "On input M#w#t where M is a TM, w a string, and t a natural number:

1. Run M on input w for t steps
2. If M halts within t steps, then do whatever M does
3. else reject.

Def: The <u>acceptance problem</u> for TMs is the language

$$A_{TM} := \{ M\#w : M \text{ is a TM that accepts input } w \}$$

Theorem: A_{TM} is undecidable.

Proof: Suppose there is a TM D that decides A_{TM}. Let F be the TM

F := "On input M, where M is a TM:

1. Run D on input M#M
   // ask D whether M accepts its own
   //   description as input
2. If D accepts M#M, then reject
   // F rejects M

   If D rejects M#M, then accept "
   // F accepts M

Consider F running on input F:

1. Run D on input F#F
2. If D accepts F#F, then F rejects F
   If D rejects F#F, then F accepts F

∴ D is wrong on input F#F ↯

∴ no such D can exist. $\boxed{}$

Actually $\{ M : M$ is a TM that accepts string $M \}$
~~$\{ M\#M : M$~~ is undecidable.

Prop: A_{TM} is T-recognizable. In fact,

universal TM $U :=$ "On input M#w:
1. Run M on input w"

recognizes A_{TM}: $A_{TM} = L(U)$.

To show a lang L undecidable:
use the ~~tepl~~ template for a proof:
"Assume L is decidable (let D decide L)
then blah blah ... blah
∴ A_{TM} is decidable ↯
∴ L is undecidable."

7

Def: $HALT_{TM} := \{ M\#w : M$ is a TM that halts on input $w \}$

Prop: ~~HA~~ $HALT_{TM}$ is undecidable.

Proof: (use the template): Suppose $HALT_{TM}$ is decided by some TM D. Then let

$N :=$ "On input $M\#w$, where M is a TM:

1. Modify M as follows:
   a) add a new state q_{loop} to M
   b) Any undefined transition of M from a rejecting state define them to go to q_{loop} instead
   c) All transitions from q_{loop} loop back to q_{loop}.

Let M' be this modified TM

// On any input w:
// if M accepts w, then M' accepts w
// if M rejects or loops on w, then
// M' loops on w
// M accepts iff M' halts on w.

2. Run D on input $M'\#w$ (and ~~D~~ do what D does)"

For any M, w,

$M\#w \in A_{TM} \overset{\text{by def of } A_{TM}}{\Longleftrightarrow}$ M accepts $w \overset{\text{by construction of } M'}{\Longleftrightarrow} M'$ halts on w

$\overset{\text{D decides } HALT_{TM}}{\Longleftrightarrow}$ D accepts $M'\#w$

$\Longleftrightarrow$ N accepts $M\#w$

And N is a decider

$\therefore$ N decides A_{TM} ↯

$\therefore$ D does not exist. ∎

$HALT_{TM}$ — The halting problem for TMs.

Def: $ALL_{CFG} := \{ G : G$ is a context-free grammar and $L(G) = \Sigma^*$, where Σ is the terminal alphabet of $G \}$

Wed: ALL_{CFG} is undecidable.

CSCE 355
4/19/2023

Recall

ALL_{CFG}

$::= \{ G : G$ is CFG that derives all strings over its input alphabet$\}$

Theorem: ALL_{CFG} is undecidable.

Pf: Outline: Assume otherwise. Then use a decider for ALL_{CFG} to decide A_{TM}. ↯

Proof: ~~~~ Let D be a decider for ALL_{CFG} (we will derive a contradiction):
Build a decider for A_{TM} using D as a subroutine:
Here is the decider:
"Given input M#w where M is TM & w is a string:

details later

1. Construct a PDA P that accepts all strings except ~~~~ ones that encode accepting computations of M on input w.

// So M accepts w ⟹ P will reject the input string encoding the accepting computation

// M does not accept w ⟹ P will accept all strings.

2. Convert P to an equivalent CFG G. ②
(computably!)

3. Run (simulate) the decider D for ALL_{CFG}
on input G

// If D accepts G, then
G derives all strings,
so P accepts all strings,
M does not accept w

// otherwise, if D rejects G,
then G does not derive all strings,
so P does not accept all strings,
so M accepts w

If D accepts G, then output "no"
else output "yes"))

Answers whether M accepts w
∴ decides A_{TM} ↯
∴ D does not exist ∎

Details for step 1:

Def: M a TM, w an input string (M halts on w)

A rigid trace of M on w is the sequence of IDs:

$$\underbrace{ID_0}_{\substack{\text{initial}\\ \text{ID}}} \vdash ID_1 \vdash \cdots \vdash \underbrace{ID_k}_{\substack{\text{halting}\\ \text{ID}}}$$

where all IDs are padded with blanks enough so that all IDs represent the same segment of tape (hence ~~each~~ ID strings are all equal length)

P branches nondeterministically in several modules (sub-PDAs), each one look for some "error" in the input, indicating that it is not a rigid trace of an accepting comp. of M on w.

submodules

P_1 looks for some kind of syntax errors:

- malformed ID: no symbols from M's state set or 2 or more ~~state~~ state symbols.

If found, then accept. (else reject)

P_2 looks to see if input is not a <u>rigid</u> trace, where all IDs have the same length:

Nondeterministically chooses two successive IDs:

$$\cdots + \overbrace{\qquad\qquad\qquad}^{ID_a} + \overbrace{\qquad\qquad}^{ID_b} + \cdots$$

pushes tokens onto stack while reading ID_a, pops them off while reading ID_b, accepts if stack empties prematurely ~~before~~ (before finishing ID_b) or still has tokens after ID_b finished. (meaning ID_a & ID_b have different lengths).

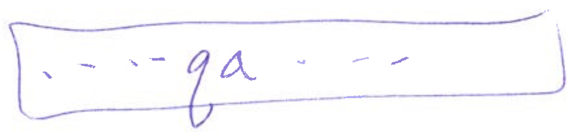
$\therefore$ P_2 will accept (on some branch) iff $\exists$ two IDs of different length

P_3 checks the first ID (initial ID) and accepts if it is not of the form $\cdots BBq_0wBB\cdots$ where q_0 is M's start state & w is the given input to M.

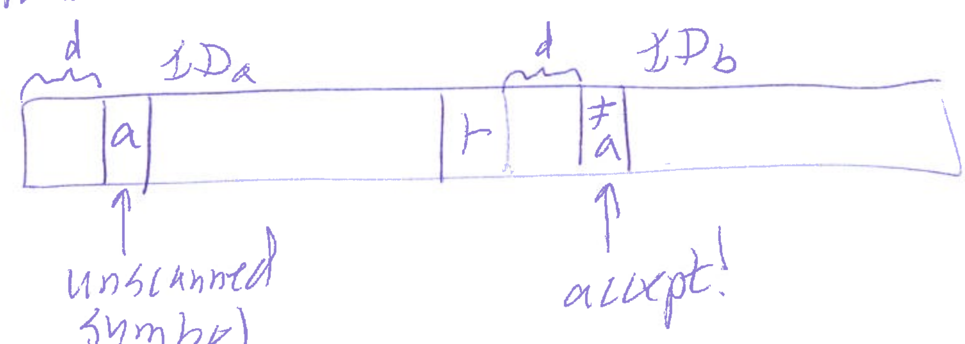
P_4 checks the final ID (not followed by a "$\vdash$" symbol) and accepts

M not accepting w — if ~~the state symbol~~ either the state symbol is rejecting, or

last ID not a halting ID — δ-function of M is well-defined for this ID



P_5 nondet. chooses two successive IDs



P_5 accepts if some unscanned symbol changes from one ID to the next.

nondet. chooses an unscanned symbol in ID_a, looks at the corresponding position in ID_b and accepts if the symbol is different from ID_a.

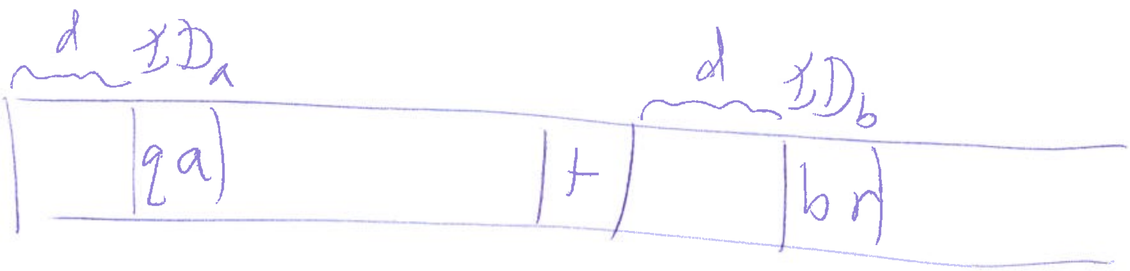
To find corresponding positions:
push onto stack d symbols (any d nondet.)
rejects if symbol is scanned in ID_a
pops stack reading ID_b to find the corresponding position

⑥

P_6 checks if the state, scanned symbol in an ID leads to the correct vicinity in the next ID:

Ex:

two successive IDs chosen nondet. by P_6



$\delta(q, a) = (r, b, \rightarrow)$

accept if this is not the case.

similarly if the head moves left

Completes the description of P.

M accepts w $\iff$ there is a rigid accepting trace T of M on input w $\iff$ all modules of P on all nondet. branches reject input T.

$\iff$ P does not accept all strings. //

Editing problem:

An editing system is a tuple

$E := \langle \Sigma, \{(x_1, y_1), \ldots, (x_k, y_k)\} \rangle$ where

Σ is an alphabet and

each $x_i, y_i \in \Sigma^*$ $(k \geq 0)$

Given an editing system E and a string w, a legal edit of w (w.r.t. E) is a string obtained by replacing some substring x_i in w with y_i.

Editing problem: Given an editing system E and an ~~st~~ initial string w, is there a finite sequence of legal edits, starting with w and ending with ε (empty string).

Ex: ~~w~~ $\Sigma = \{a, b\}$

$w = bbaa$

$E = \langle \{a,b\}, \{(ba, aab), (bb, b), (b, aa)$

$(aaaaa, \varepsilon)\} \rangle$

$\times$ $b\underline{ba}a \to \underline{ba}aba \to aa\underline{ba}ba \to aaaab\underline{ba}$

$\to aaaaba \to \times$

Does this work?

Theorem: The Editing Problem is undecidable.

Pf outline: any decider for the editing problem can be used as a subroutine to decide A_{TM}.

CSCE 355
4/24/2023

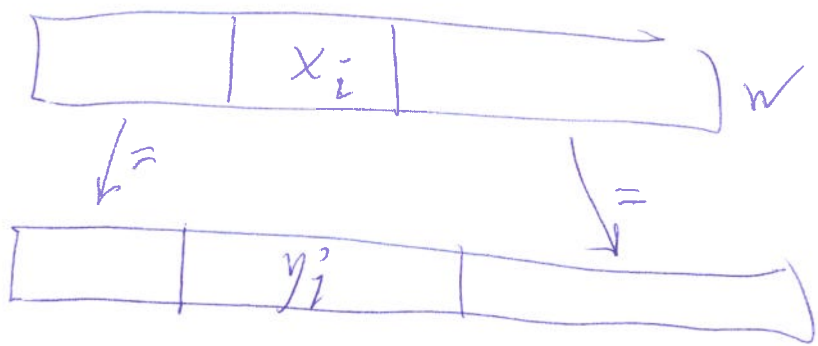
<u>Thm</u>: EP is undecidable. 1

<u>Recall</u>: An editing system has an alphabet Σ and a finite set of pairs $\{(x_1,y_1),\ldots,(x_k,y_k)\}$ where $x_i, y_i \in \Sigma^*$

some i, $1 \le i \le k$

An allowed edit:

Replace some x_i substring with y_i.



<u>EP</u>: Given an editing system $\langle \Sigma, E \rangle$ where $E = \{(x_1,y_1),\ldots,(x_k,y_k)\}$ for some k, and an ~~initial~~ initial string $w \in \Sigma^*$

<u>Question</u>: Is there a finite sequence of edits, starting with w and ending with $\underline{\varepsilon}$?

empty string.

<u>Proof</u>: Suppose EP is decidable. Use decider for EP as a subroutine to decide A_{TM} $\lightning$

$\therefore$ EP has no decider.

Given an arbitrary TM ~~M and w~~ ②

$M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$

and arbitrary input $w \in \Sigma^*$, we computably construct a pair $\langle \langle \Delta, E \rangle, \boxed{u} \rangle$ such that u edits to ε iff M accepts w. So deciding the former means we can decide the latter.

[Edits using $\langle \Delta, E \rangle$ mimic the steps in the computation of M on input w.]:

$\Delta := \Gamma \cup Q \cup \{\$, \#\}$ [any $\$, \# \notin \Gamma \cup Q$]

$u := \$ q_0 w \$$ (initial ID of M on input w between 2 $'s)

E consists of the following pairs:

1. Transition pairs (of M) For every $q \in Q$ and $a \in \Gamma$ such that $\delta(q, a) = (r, b, \rightarrow)$ (some $r \in Q$, $b \in \Gamma$), ~~include~~ include the pair

$$(qa, br)$$

For every $q \in Q$, $a \in \Gamma$ such that

$$\delta(q,a) = (r, b, \leftarrow)$$

and for every $c \in \Gamma$, include ~~the~~ the pair

$$(cqa, rcb)$$

2. Expansion pairs: For every $p \in \Gamma \cup Q$ ~~add~~ include the pairs

$$(\$p, \$Bp) \text{ and } (p\$, pB\$)$$

[allows padding with blanks on either end]

3. Contracting pairs For every (accepting) state $q \in F$ and every $a \in \Gamma$ such that $\delta(q,a)$ is undefined [M accepts!] include the pair $(qa, \#)$. [# marks acceptance]

For every $p \in Q \cup \Gamma$, include the pairs $(p\#, \#)$ and $(\#p, \#)$

Finally, include the pair $(\$\#\$, \varepsilon)$

End of construction.

Correctness verbally explained.

u edits to ε iff M accepts w ☐

Review:

- Regular langs

- CFLs

- TMs & (un)decidability

CFG → parse tree

↓

PDA

<u>Ex:</u>

$S \to aSbS$

$S \to T$

$T \to cT \mid \varepsilon$

$\delta(q, a, a) = \{(q, \varepsilon)\}$

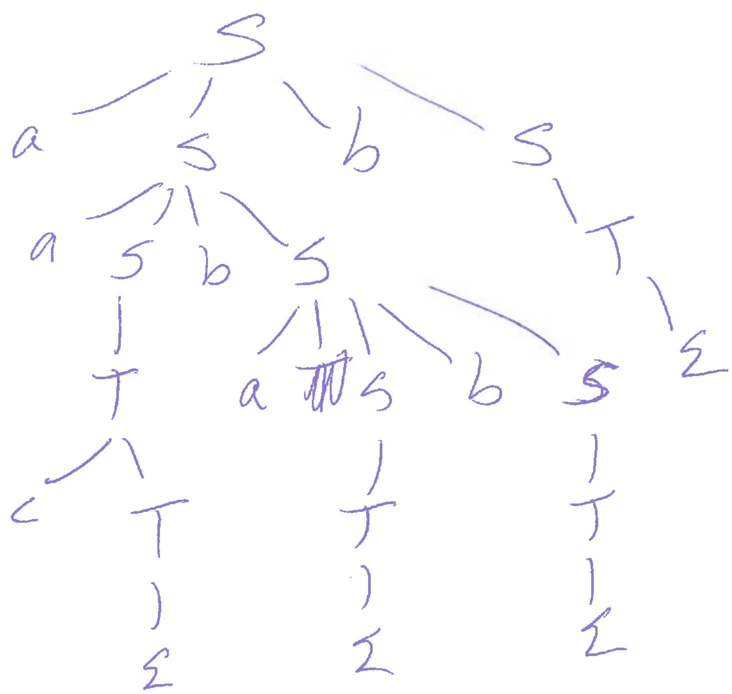
$\delta(q, b, b) = \{(q, \varepsilon)\}$

$\delta(q, c, c) = \{(q, \varepsilon)\}$

$\delta(q, \varepsilon, S) = \{(q, aSbS), (q, T)\}$

$\delta(q, \varepsilon, T) = \{(q, cT), (q, \varepsilon)\}$

Parse tree for $aacbabb$



Restricted PDA $\longrightarrow$ CFG

$\downarrow$

computation path

Balanced parens

$0 = ($

$1 =)$

$0, Z_0 / {+}Z_0$ = push +

$0, + / {+}{+}$ = push +

$1, + /$ pop

$\varepsilon, Z_0 /$ pop

$\Gamma = \{Z_0, +\}$

Input

010011

$(q, 010011, Z_0)$

$\downarrow$ $(q, 010011, \varepsilon)$

$\rightarrow$ $(q, 10011, +Z_0)$

$\delta(q, 0, Z_0) = \{(q, \text{push } +)\}$ ~~(push +)~~

$\delta(q, 0, +) = \{$ ~~(push +)~~ $(q, \text{push } +)\}$

$\delta(q, 1, +) = \{(q, \text{pop})\}$

$\delta(q, \varepsilon, Z_0) = \{(q, \text{pop})\}$

$(q, 10011, +Z_0) \rightarrow (q, 0011, Z_0)$

$(q, 0011, Z_0) \rightarrow (q, 011, +Z_0)$

$(q, 0011, Z_0) \rightarrow (q, 0011, \varepsilon)$

$(q, 011, +Z_0) \rightarrow (q, 11, ++Z_0) \rightarrow (q, 1, +Z_0) \rightarrow (q, \varepsilon, Z_0) \rightarrow (q, \varepsilon, \varepsilon)$

accept

Equiv grammar: $V = \{S, [qZ_0q], [q+q]\}$ 7

$$S \to [qZ_0q]$$

$$[qZ_0q] \to \varepsilon \mid 0[q+q][qZ_0q]$$

$$[q+q] \to 1 \mid 0[q+q][q+q]$$

$a^i b^j c^k \quad i < k$