CSE 396 — Lecture Tue 4/12 — Spring 2016 ①

Def^n: A {instantaneous description / configuration} (ID) I of a TM M specifies:

- the current state q of M
- the contents w of all tapes together
- the head position on each tape.

For one-tape TMs it is convenient to encode IDs by strings over the alphabet $\Gamma' = Q \cup \Gamma$ $(M = (Q, \Sigma, \Gamma, \delta, B, s, q_{acc}, q_{rej})$ wlog. $Q \cap \Gamma = \emptyset)$

ID $I = \underline{uqcv}$ means: { M is in state q scanning char c; u includes all non-blank cells to the left, v ditto to right.

[text p168 includes c as first char of v]

The *initial* ID of M on an input $x \in \Sigma^*$ is (scanning first bit of x).

$$\underline{I_0(x)} = sx \quad \text{or} \quad \wedge s x \$ \quad \text{using "UNIX convention"}$$

What if $x = \epsilon$? $I_0(x) = \underline{sB}$ or just s or $\wedge s \$$ no ambiguity.

not totally clear which one text would write

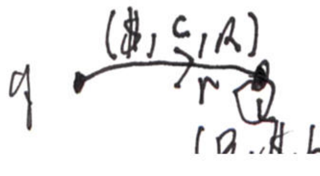
☆ Code M to write B only at left or right end. $\Rightarrow$ in $uqcv$, • u and v have no Bs

Use '$\underline{\%}$' as alias ~B otherwise • if $c = B$ then $u = \epsilon$ or $v = \epsilon$.

Say we read all of x, going to

OK to be scanning B temporarily at one end to "move an endmarker" to make more room

$$I = \wedge x q \$ \vdash_M \wedge x c r B \vdash_M \wedge x r c \$$$



~~$\vdash_M \wedge x c r \$$~~ - no, left move made M once again scan c.

Defn $I \vdash_M J$ if ID J can follow from I by executing one instruction of M.

②

Definition: A <u>computation</u> of a TM M on an input $x \in \Sigma^*$ is a sequence $(I_0, I_1, \ldots, I_t)$ of IDs such that:

- I_0 is the starting ID on input x. (whatever encoding you use for k-tape TMs)
- for all j, $1 \leq j \leq t$, $I_{j-1} \vdash_M I_j$ (text says this in prose)
- I_t is a <u>halting ID</u>, meaning: (General) $(\nexists J)\, I_t \vdash_M J$

(By conv.) I_t has state <u>q_{acc} or q_{rej}</u>.

The computation <u>accepts</u> if I_t has q_{acc}.
$\equiv I_t$ is an <u>accepting ID</u>.

Similarly to grammar derivations: define $I \vdash_M^0 I$ for any ID I,

$I \vdash_M^k J$ if there is some ID I' s.t. $I \vdash_M I' \wedge I' \vdash_M^{k-1} J$.

$I \vdash_M^* J$ if for some $k \geq 0$, $I \vdash_M^k J$.

$\vdash$ called "turnstile." Text says "yields." LaTeX typeset \vdash on Piazza

Def^n: $L(M) = \{x \in \Sigma^* : I_0(x) \vdash_M^* J$ for some <u>accepting ID</u> $J\}$

Some "Good Housekeeping" features: ① If M writes no internal blanks, then when it "wants to" go to q_{acc}, it can execute a routine that erases the entire tape except a single $\underline{1}$, leading to the unique <u>accepting ID</u> $I_f = \underline{q_{acc}1}$.

- Likewise, a <u>halting</u> rejecting computation can wlog. end in $I_{rej} = \underline{q_{rej}0}$.

§5.3
Forward reference: In any accepting (or rejecting) ID $u\,q\,c\,v$, we can let v' be the maximum prefix of v having chars in Σ only, and if $c \in \Sigma$, <u>output</u> $= cv'$. If $c \notin \Sigma$, <u>output</u> $= \epsilon$. ② If y is the intended output, can arrange $I_f = \underline{q_{acc}\,y}$.

Key Def^{n}s: A language $A \subseteq \Sigma^*$ is
- Turing recognizable (text)
- Turing acceptable ← (standard)
- computably enumerable (c.e.)
- recursively enumerable (r.e.)

if there is a det^c TM M s.t. $A = L(M)$.

A is **decidable** if in addition for all x, $I_0(x) \vdash_M^* I_f \vee I_0(x) \vdash^* I_{rej}$

Equivalently, say $M(x)\downarrow$ ($\downarrow$ = "halts") if , and M is **total** if $\forall x\, M(x)\downarrow$.

Then A is **decidable** $\equiv$ $A = L(M)$ for some **total** DTM M.

synonym **recursive**. Furthermore, suppose that for all $x \in \Sigma^*$:

$$\boxed{\begin{array}{l} x \in A \implies M(x) \text{ halts and outputs } 1 \\ x \notin A \implies M(x) \text{ halts and outputs } 0 \end{array}}$$

Then A is **decidable**.

Then as a {function computer / **transducer**}, M computes the **total** characteristic function χ_A

Greek chi → $\chi_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$ which for Myhill-Nerode we just write $A(x)$

Note: If we only have $\forall x$: $x \in A \Rightarrow \underbrace{M(x)\downarrow =}_{\text{halts and equals}} 1$, i.e. possibly $\underbrace{M(x)\uparrow}_{\text{doesn't halt}}$ when $x \notin A$, then A is **Turing-recognizable**.

A might be decidable by another machine M', or not [Thm. → not!]

Def^{n}: A function $f: \Sigma^* \to \Sigma^*$ is (total) computable if there is a total DTM M that computes f: $\forall x \in \Sigma^*: I_0(x) \vdash_M^* q\, f(x)$ where q is a halting state.

(by default total, $dom(f) = \Sigma^*$)

If f is a partial function and $\forall x \in dom(f)$ (Domain) $M(x)\downarrow = f(x)$, then f is **partial computable**.

(with domain A) ⟶

A is decidable $\iff$ $A(x)$ is total computable

A is c.e. $\iff$ $A(x)$ is partial computable

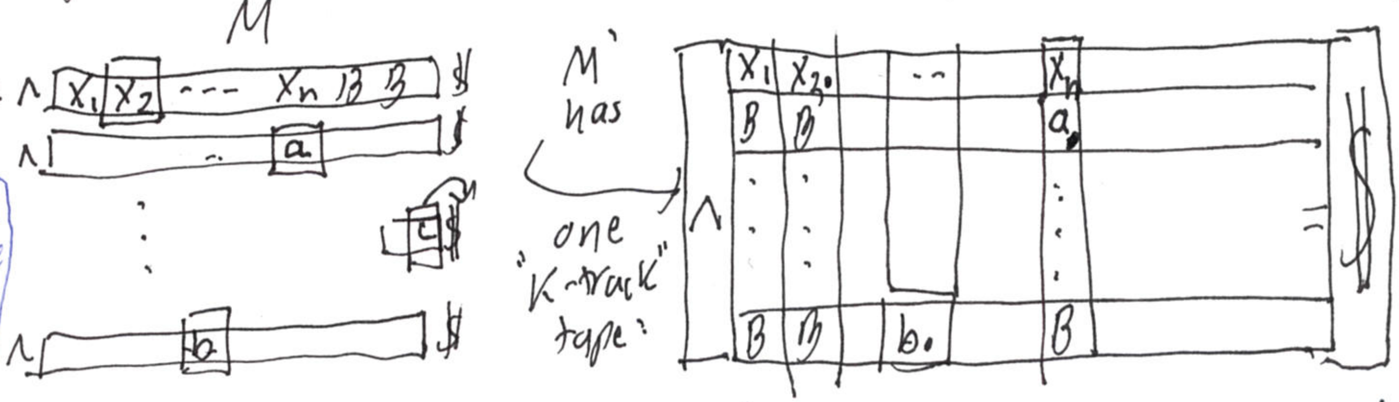
Rest of Ch 3: TMs and Programs.

(3.13 in text)

Theorem: For every K-tape TM $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ we can design a 1-tape TM $M' = (Q', \Sigma, \Gamma', \delta', B, s', F')$ s.t. $L(M') = L(M)$.

Moreover they compute the same (partial) function.

Idea:



Text instead "shuffles" the tapes, but not consistently

Use • as an extra char marker for the current location of each head

Hence $\Gamma' = \Gamma \cup (\Gamma \cup \Gamma_\bullet)^K$ where $\Gamma_\bullet$ is a dotted copy of Γ.

(includes B and $B_\bullet$)

In this manner, every ID I of M is "replicated" by an ID I' of M'.

If $I \vdash_M J$, then M' can build the corresponding J' from I', but it usually will require one L-to-R-to-L pass between $\wedge$ and \$.

Thus M' simulates M. $\boxtimes$

(I.e., M' is operationally equivalent to M.)

initially $s = n$, always $s \le t$

$s \le \max\{t, n\}$

Note: If s is the current/typical distance from $\wedge$ to \$, then each pass requires up to $4s$ steps (can do in $2s+2$)

Hence if it takes M t steps to halt, M' halts in $O(st)$ steps.

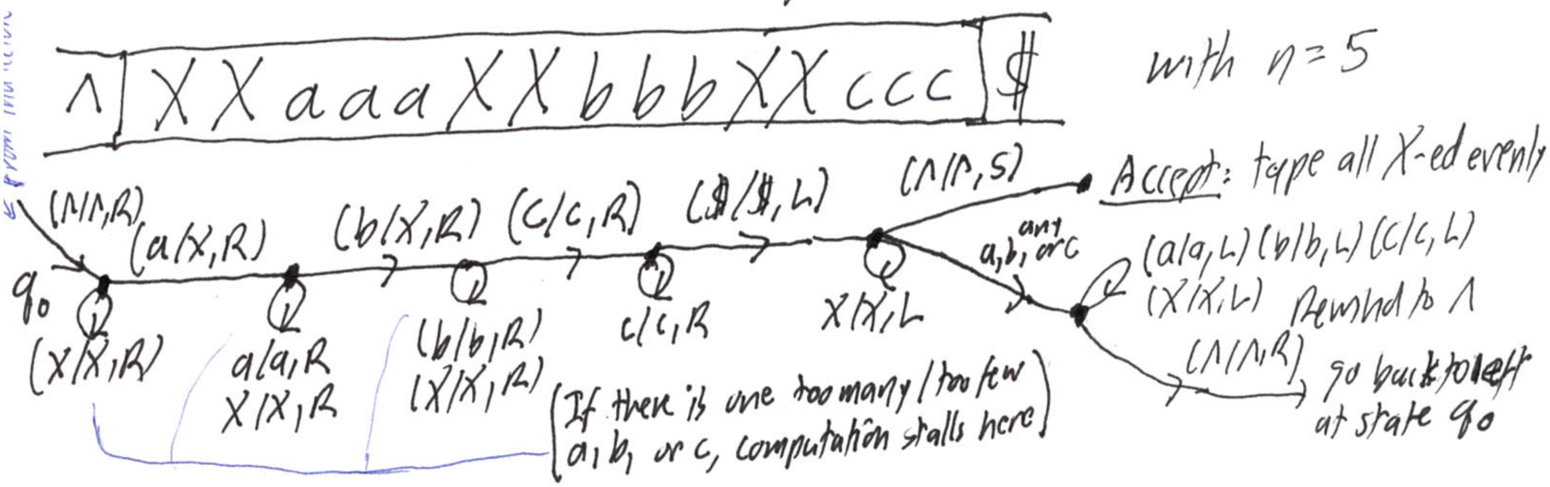
Especially when $t = O(n)$, M' can take $O(n^2)$ time.

At the end we will care about the time t and space s to solve problems.

For now note: Your 2-tape machines typically take $O(n)$ time, $O(n^2)$ 1-tape.

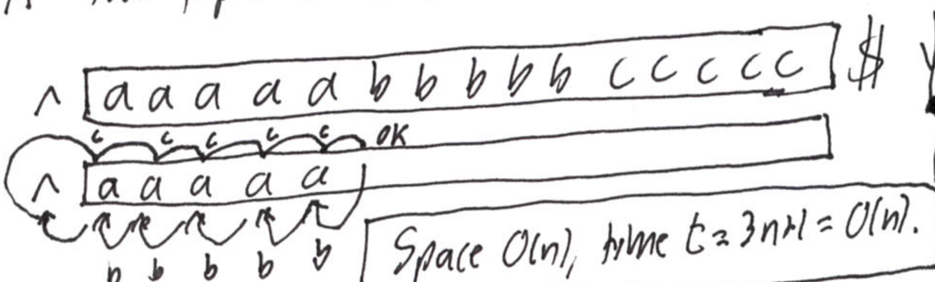
Example (writing end of lecture again on clean sheet) (expanded a bit)

$L = \{a^n b^n c^n : n \geq 1\}$. Recall our one-tape TM first verified that the input x belongs to $\underline{a^+ b^+ c^+}$. It then maintains the invariant that the tape always belongs to $\underline{\wedge X^* a^* X^* b^* X^* c^* \$}$, e.g.:

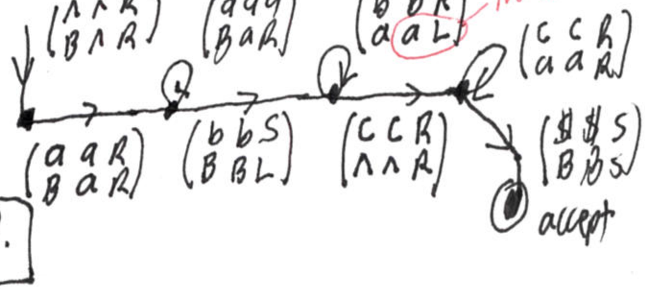


Here the distance between $\wedge$ and $\$$ stays at $3n+1$, fixing the space as $\underline{s} = 3n+1$. Each pass uses $\underline{2s}$ steps. The TM does n passes. $\therefore$ Total time $\underline{t} \approx n(6n+2) = \underline{\Theta(n^2)}$.
(L to R to L)

A two-tape TM M_2 can execute a simpler & quicker strategy in $O(n)$ time:



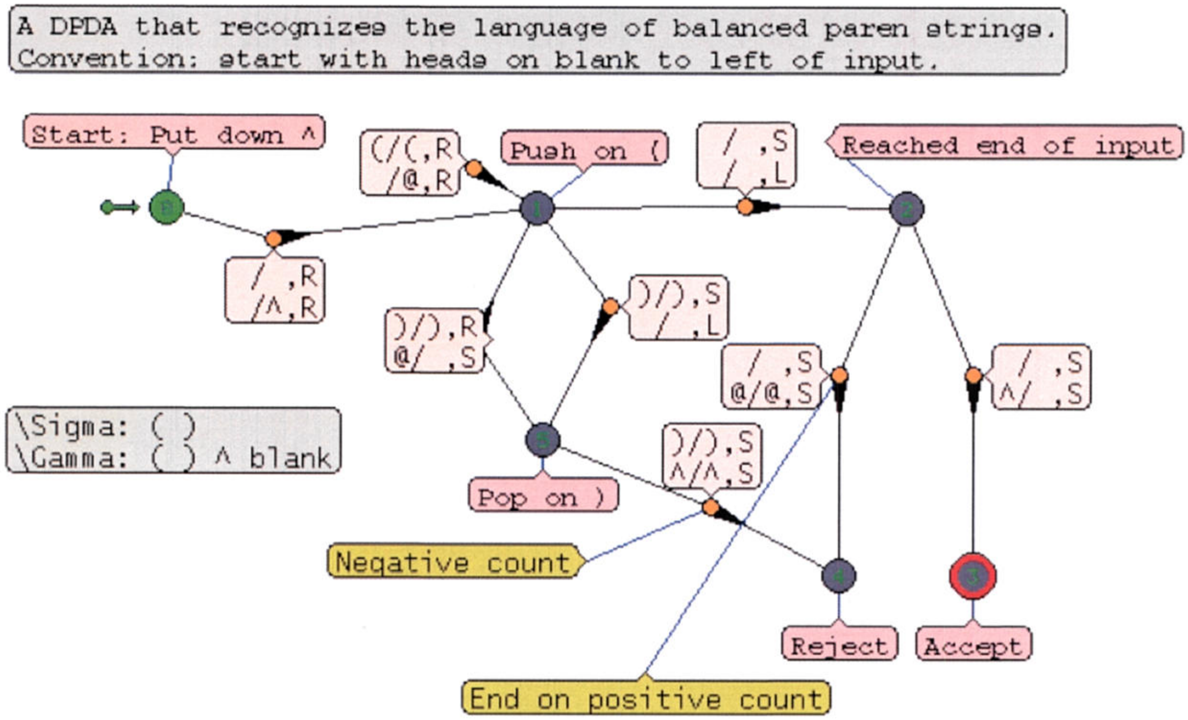
Space $O(n)$, time $t = 3n+1 = O(n)$.

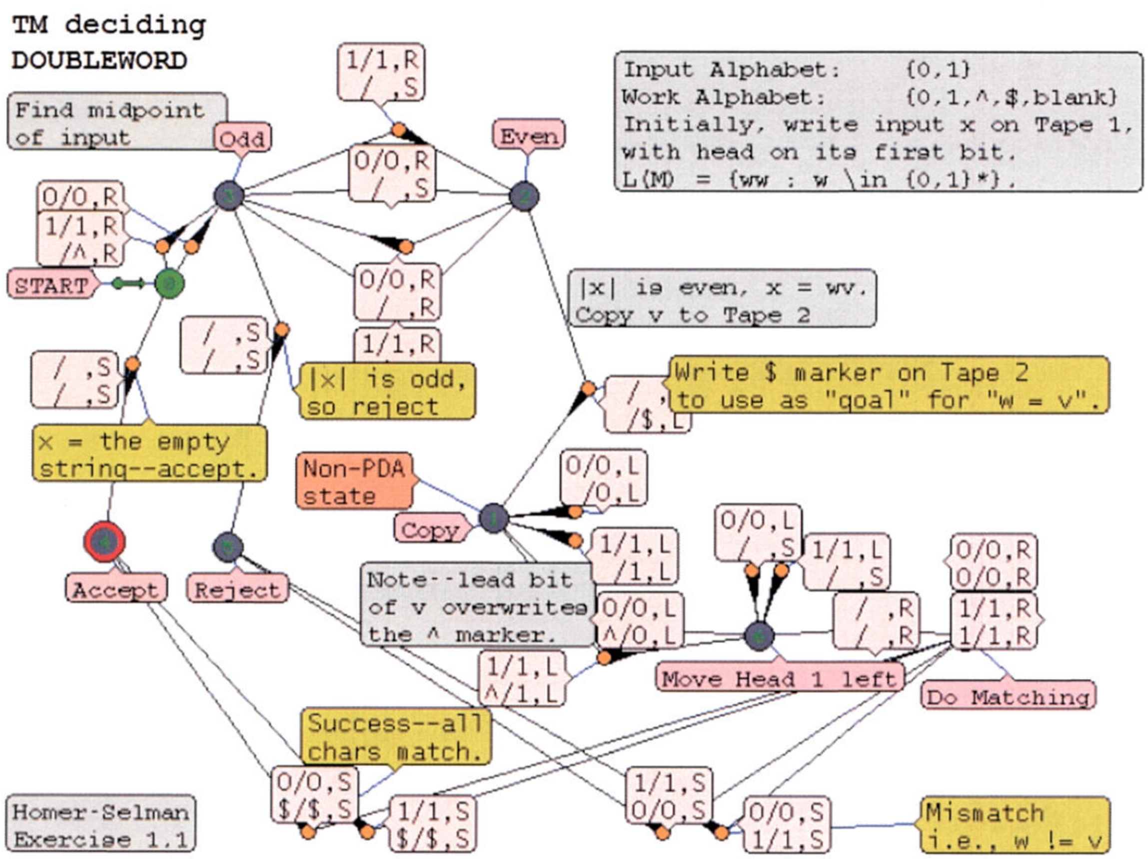


If you convert M_2 into a one-tape TM M_1, the time becomes $\Theta(ns) = \Theta(3n^2+n) = \Theta(n^2)$ quadratic again.

☆ "Polynomial time" ignores quadratic time loss, so 1 tape and k tapes are equivalent for it. The "Universal RAM Simulator" handout sketches how this equivalence extends to a "mini-assembly" language, and thence to time in real programming languages a-la CSE 331. "Church-Turing Thesis".

Two two-tape Turing Machines, the first a DPDA, the second not





~~CSE396~~ & 596 Spr 2016: To Skim for §3.2 of text

Universal RAM Simulator

Fall 2014

0. LDL n
1. LDR Y
2. LDI Y
3. STO Y
4. STI Y
5. ADD Y
6. SUB Y
7. SHF d
8. ABS
9. JMP L

Program Syntax: INSTR = $DIG^* \cdot ALPH^3 \cdot DIG^*$; PGM = ! $INSTR^*$

Register Syntax: REG = [DIG^* # (-?) DIG^*] REGS = REG^*

ALU Syntax: ALU = (-?) DIG^* All tapes get ∧ --- $ delimiters

Initial: ··· | ! INS_1; INS_2; ··· INS_m; # $x_1 x_2 \cdots x_n$ | ···

P	G	M	:	∧	! INS_1; INS_2; --- INS_m; $ Program P
R	E	G	:	∧	$
A	L	U	:	∧	$x_1 x_2 x_3 x_4 \cdots x_n$ $

Input Convention: Start P(x) with x in ALU

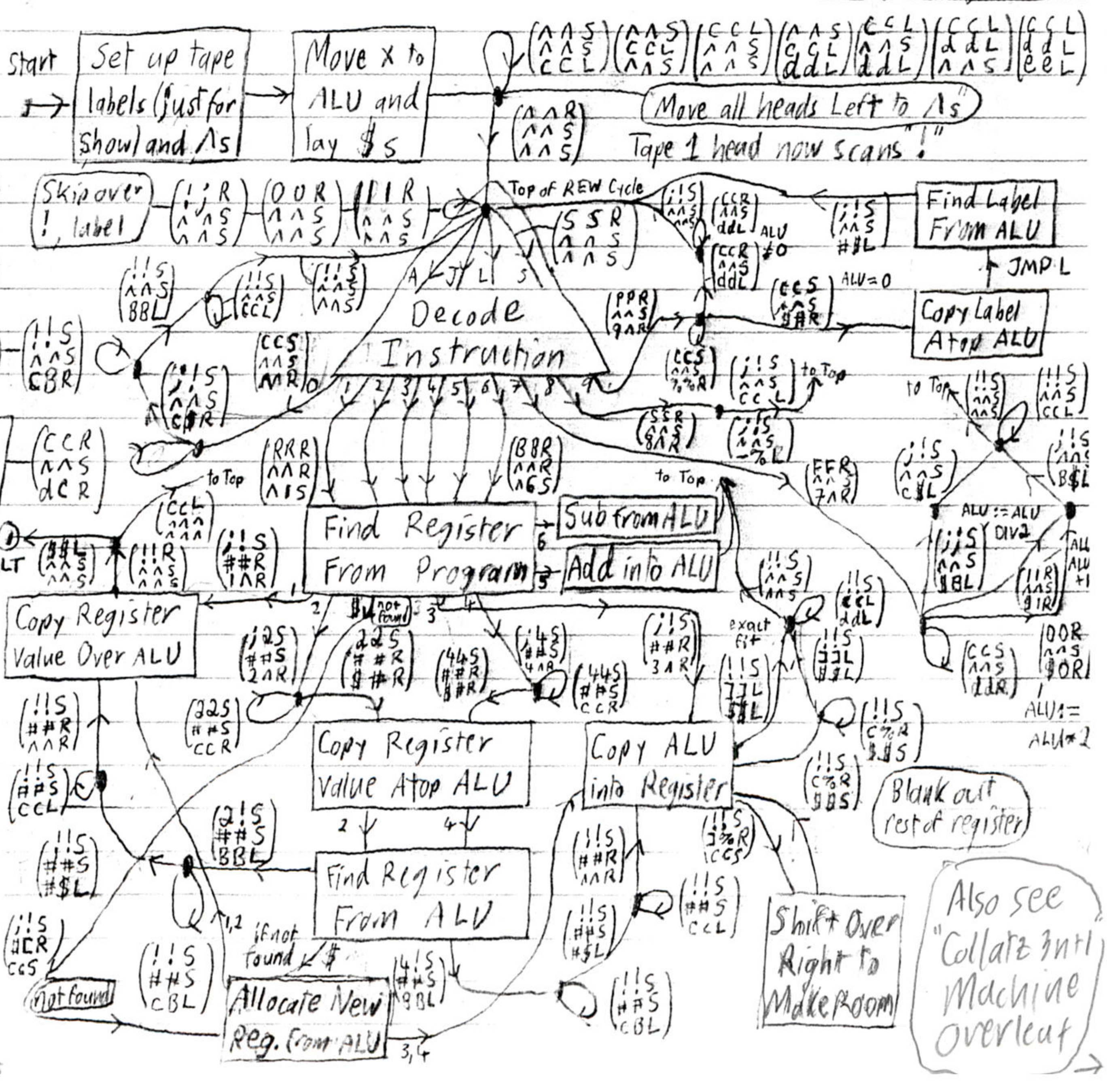
Output: Last Instr is LDR 1; putting P(x) in ALU

DIG = 0,1

DIG^*: 2-adic #s

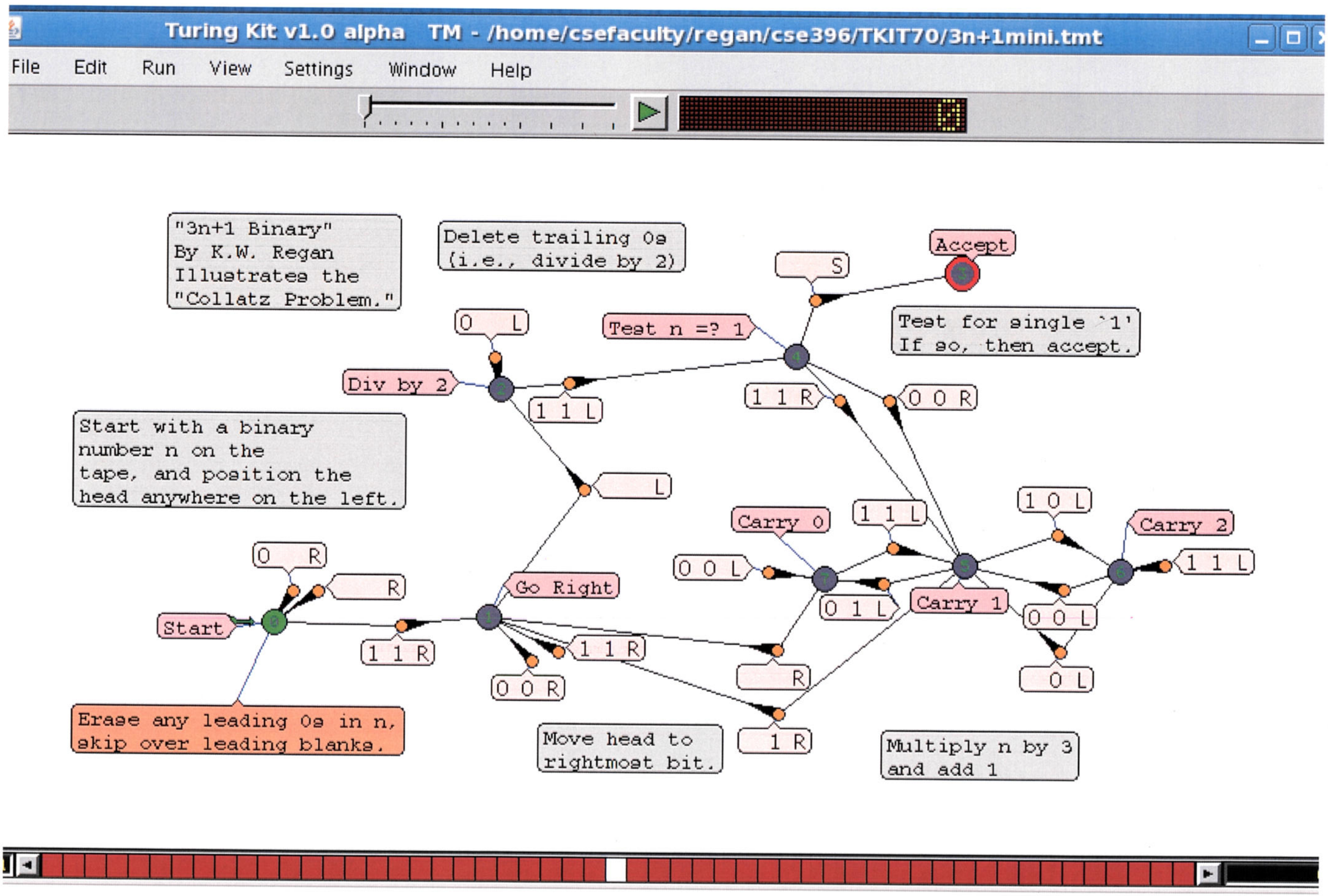
c,d,e: wildcards

... replaced by ...F, SHF 0, SHF 1



This uses "wildcard" chars c,d,e... ...ve tried to implement this feature in the ...unky Kit program but ...d not quite succeed. The point is that the instruction can ... finished in just ... more pages

Also see "Collatz 3n+1 Machine" overleaf →



Turing Kit v1.0 alpha TM - /home/csefaculty/regan/cse396/TKIT70/3n+1mini.tmt
File Edit Run View Settings Window Help
"3n+1 Binary"
By K.W. Regan
Illustrates the
"Collatz Problem."
Delete trailing 0s
(i.e., divide by 2)
Start with a binary
number n on the
tape, and position the
head anywhere on the left.
Erase any leading 0s in n,
skip over leading blanks.
Move head to
rightmost bit.
Multiply n by 3
and add 1
Test for single '1'
If so, then accept.
Start
Div by 2
Test n =? 1
Accept
Go Right
Carry 0
Carry 1
Carry 2
0 R
R
1 1 R
0 0 R
1 1 R
0 L
1 1 L
L
S
1 1 R
0 0 R
R
1 R
0 0 L
0 1 L
1 1 L
1 0 L
0 0 L
0 L
1 1 L