### Multiple tracks on single tape

<table>
<thead>
<tr>
<th>% 0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each symbol is a k-tuple (here, k = 3)

Ex. Construct M to accept if input is a binary representation of a prime number n

(n is prime if has no other factors except n and 1 that divide it evenly)

Procedure: Check to see if 2, 3, ... n-1 divide n; if any do, reject. if none, accept.

How use tracks?

---

<table>
<thead>
<tr>
<th>1</th>
<th>% 0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On track 2, cycle through numbers m, starting with 2 and ending with n-1.
For each such m, use track 3 to see if n is divisible by m: start by copying track 1 to track 3. Then repeatedly subtract m on track 2 from track 3, until you get a remainder on track 3 which is less than m. If remainder = 0, then m divides n, and so M rejects. If remainder is not 0, add 1 to m to obtain the new m, and continue, till get to n.

How do subtraction? test if one number less than another?
## Shifting Symbols

TM can make space on its tape by shifting all non-blank symbols to right. To do so, use state to store symbol that is being moved, till get to deposit location (e.g., □) deposit the symbol, and move left again.

**Ex.** TM routine to move symbols 2 places to right uses states \( \{ q_1, q_2 \} \times \Gamma \times \Gamma \), new symbol \( X \)

1. \( \delta ((q_1, \square, A)) = ((q_1, \square, A), X, R) \)
   
   (Store first symbol read in last state component, replacing it by \( X \), and move Right)

2. \( \delta ((q_1, A, B)) = ((q_1, A, B), X, R) \)
   
   (Store new symbol read in last component, replace by \( X \), symbol in last component to the middle, and move Right)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3. ( \delta ((q_1, A, B), C)) = ((q_1, B, C), A, R) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Store symbol read in last state component, shifting B to middle, and deposits A 2 spaces to right)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. ( \delta ((q_1, A, B), \square)) = ((q_1, B, \square), A, R) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(When get to blank, stored symbols are deposited in order)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. ( \delta ((q_1, A, \square), \square)) = ((q_2, \square, \square), A, L) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(When all symbols deposited, got to state q2 and move left to find ( X ), the rightmost vacated space)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. ( \delta ((q_2, \square, \square), X) = ((q_2, \square, \square), X, L) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(T moves left, until a ( X ) is found. At that point, T will transfer control to a different state)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Ex.** \( \text{010} \)
Simulation

Suppose M with input w goes from C1, C2, ..., Cn
M’ simulates M if on input w, M’ goes through a
sequence of configurations representing C1,...,Cn
(Note that M’ can enter other configurations in between)
M’ must be able to
1. calculate rep. of Ci+1 from rep. of Ci
2. determine if M accepts on Ci using rep. of Ci.
Ex. M from M1, M2 with L(M) = L(M1) ∩ L(M2)
   Simulate M1 and M2 on track 2.
   % 0 1 1 0 $

   if M1 accepts, then M goes to step 2; ow, reject.
2. Copy input onto track 2. Simulate M2 on track 2.
   if M2 accepts, then M accepts; ow, reject.

Subroutines

Somewhat like simulation, but "separate" machines
M’ subroutine of M:
   M’, M should have different states
   To call M’, M should enter the start state of M’
   and follow the transitions of M’
   From a halting state of M’, M reenters state
   of M, and proceed
   input parameters to M’ should be in fixed place;
   output of M’ as well.
Ex. Shifting symbols as subroutine
Ex. arithmetic subroutines
**Variants of TM's**

Many different ways to define TM model:
- multiple tapes, 2-way tapes, nondeterminism

Our model, and all reasonable variants, have same power (i.e., accept same class of languages)

So the model is robust

**Ex.** \( \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,S\} \) (S for stay)

Is this class of TM more powerful? NO!

Given any TM \( M' \) in extended class, can define
- a regular TM M that does 2 moves (R,L)
- for every S move of \( M' \), and acts exactly like \( M' \) otherwise

---

**Two-way tapes**

Lemma: If L is accepted by a TM M with a 2-way tape, then L is accepted by a TM \( M' \) with 1-way tape.

Proof idea: Use 2 tracks:

\[
\begin{array}{c|c|c|c}
A_0 & A_1 & \ldots \\
\hline
S & A_0 & A_1 \\
\end{array}
\]

for single 2-way:

\[
\ldots A_{-1} A_0 A_1 \ldots
\]

\( M' \) must keep track of whether scanning symbol on top or bottom track, build into state:

- \( Q' = \{q_1\} \cup \{Q \times \{\bot\}\} \)
- \( \Gamma' = \{[x,y]\mid x, y \in \Gamma \text{ or } y = S\} \)
- \( \Sigma' = \{[a,\#]\mid a \in \Sigma\} \)
Lemma: If L is accepted by a multitape TM, then L is accepted by a single tape TM.

Proof idea: use tracks, with special markers for each tape head

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\} \]

Multiple tapes

M' will simulate M by making multiple passes over tape, 1 for each track. First, it needs to determine all the symbols being read, in order to determine the next move. So k symbols must be stored in state. Then, once the next move is determined, it must update each track with new symbol and new tape head position.

Nondeterministic TM's

\[ \delta : Q \times \Gamma \rightarrow P( Q \times \Gamma \times \{L,R\} ) \]

Lemma: If L is accepted by a nondet. TM N, then L is accepted by some deterministic TM D.

Proof idea: D will simulate N by trying all possible branches of N's computation on input w. If D finds an accept state on some branch, it accepts. Otherwise, D does not accept.

QU: How should D search the computation tree??

How represent a branch?
<table>
<thead>
<tr>
<th>Robustness of TM's</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is this surprising??</td>
</tr>
<tr>
<td>TM's ←→ &quot;idealized&quot; programming languages</td>
</tr>
</tbody>
</table>

Programming languages, if sufficiently general, can do the same computations
- Ex. C, Lisp, Java

TM's make formal our informal notion of algorithm

Church-Turing Thesis: The Turing machine model (and any reasonable variant of it) embodies our informal notion of algorithm.

Can't be proved!