**TM Construction**

Methods for programming TM’s

1. Storage in states
   
   Old states $Q$  New states $Q \times I_1 \times \ldots \times I_k$

   Ex. $L$ given by $ab^* \cup ba^*$

   Construct TM $M$ that stores first symbol, then makes sure not in rest of input.

   $Q = \{q_0,q_1,q_2\} \times \{a,b,\square\}$

   ![TM Diagram]

   **Multiple tracks on single tape**

   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?
Multiple tracks on single tape

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>%</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</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 \{q1, q2\} x Γ x Γ, new symbol X

1. \( \delta \left( (q1, □, □) A \right) = (q1, □, A, X, R) \)
   (Store first symbol read in last state component, replacing it by X, and move Right)

2. \( \delta \left( (q1, □, A) B \right) = (q1, A, B, X, R) \)
   (Store new symbol read in last component, replace by X, symbol in last component to the middle, and move Right)
### Shifting Symbols

3. \[ \delta \left((q_1, A, B), C\right) = \left((q_1, B, A), A, R\right) \]
   
   (Store symbol read in last state component, shifting B to middle, and deposits A 2 spaces to right)

4. \[ \delta \left((q_1, A, B), \square\right) = \left((q_1, B, \square), A, R\right) \]
   
   (When get to blank, stored symbols are deposited in order)

5. \[ \delta \left((q_1, A, \square), \square\right) = \left((q_2, \square, \square), A, L\right) \]
   
   (When all symbols deposited, got to state q2 and move left to find X, the rightmost vacated space)

6. \[ \delta \left((q_2, \square, \square), X\right) = \left((q_2, \square, \square), X, L\right) \]
   
   (T moves left, until a X is found. At that point, T will transfer control to a different state)

   Ex. 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 \( L(M) = L(M1) \cap L(M2) \)

Simulate M1 and M2 on track 2.

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

   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', 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