Introduction to Turing Machines

Prof. Susan Older

4 November 2019

Turing Machines: A Little History

1928: David Hilbert poses the **Entscheidungsproblem**.

Is there an effective method (i.e., an algorithm or mechanical procedure) that, when given an arbitrary statement in first-order logic, always correctly determines whether that statement is universally valid?

1935-6: Church & Turing independently show the answer is "no".

Both of them first needed to define effective method precisely:

- Alonzo Church introduces the λ-calculus, which later becomes the basis for functional-programming languages such as Haskell.
- Alan Turing introduces the notion of a Turing machine.
  The idea of a universal machine that could simulate other machines lies at the heart of today's computing.
- Turing also shows the two notions to be equivalent notions of computation.

Turing Machines: The Inspiration

What does algorithm (or effective method) mean?

Turing looked at what a **computer**[1] does:

- She makes marks on paper.
- She sometimes shifts attention from what was previously written to what she's writing now.

What determines what she writes next?

- The symbols she's currently looking at
- Her current state of mind

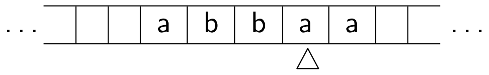
[1] Job title of person doing calculations

Turing Machines: The Basics

A Turing machine is a hypothetical computing device that comprises:

- A tape of unbounded length containing cells

  Each cell is either empty or contains a symbol; only finitely many cells contain symbols.
- A tape head that can read a cell and move to the right or left
- A notion of state, which (along with current symbol) determines TM's behavior



Turing Machines: More Basics

Running a Turing machine (think "program"):

1. Write input string on the tape.
2. Execute the Turing machine (see below).
3. If/when the Turing machine halts, the result is what's written on the tape.

Behavior of TM depends on current state and current symbol:

1. Read current symbol
2. Erase symbol and write new symbol (or leave cell blank)
3. Move one cell to the right or to the left
4. Update the state

Specifying TM Behavior: A Very Simple Example

Let's describe a TM that:

- Starts at the leftmost input symbol
- Converts as into bs (and vice versa)
- Stops when it sees a blank cell

The state-transition table:

	Current Symbol	
State	a	b
scan	b,R,scan	a,R,scan

How to read the table:

- If you're in state scan and you see symbol a, then (i) replace it with b, (ii) move to the right, and (iii) change to state scan.
- If you're in state scan and you see symbol b, then (i) replace it with a, (ii) move to the right, and (iii) change to state scan.

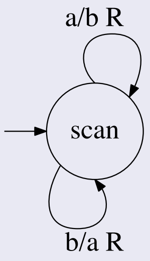
∗ If there's no instruction for current state/symbol: Halt!

Same Simple TM: A Different Description

The state-transition table:

	Current Symbol	
State	a	b
scan	b,R,scan	a,R,scan

The state-transition diagram (a.k.a. bubble diagram):



- States are represented by circles ("bubbles").
- Transitions between states are represented by arrows.
- Each transition arrow is labeled by (in order): current symbol, symbol to be written, and direction tape head should move.

∗ The start state is indicated by an arrow that does not originate from a state.

Same Simple TM: An Executable Version

A Haskell version:

```
import Turing

-------------------------------------------------
-- TM to swap 'a's and 'b's in the input string
-------------------------------------------------
convertAB :: Prog
convertAB = [(("scan",'a'),('b',Rght,"scan")),
             (("scan",'b'),('a',Rght,"scan"))]
```

To try it out in the interpreter:

```
*Main> stepRun convertAB "aaababbbaa"
```

Another TM Task: Odd or Even?

Desired behavior of TM:

- Input is a series of as and bs.
- Tape head starts at the leftmost input symbol.
- TM should determine whether the number of bs in the input string is even or odd:
  - If even, the symbol E should appear at end of string.
  - If odd, the symbol O should appear at end of string.

The general approach:

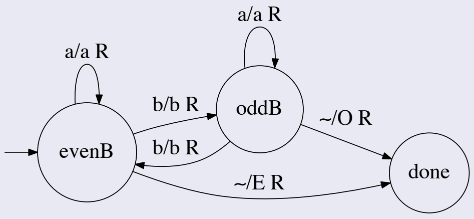
1. Skip over any as.
2. Each time a b is read, switch states to indicate current status (even or odd).
3. When a blank symbol is encountered, write E or O (depending on current state) and halt.

Another TM Task: Odd or Even? (Bubble Diagram)

The general approach:

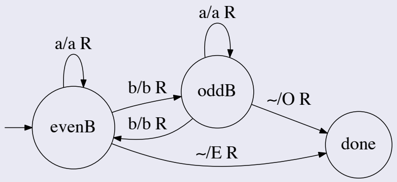
1. Skip over any as.
2. Each time a b is read, switch states to indicate current status (even or odd).
3. When a blank symbol is encountered, write E or O (depending on current state) and halt.

The state-transition diagram (a.k.a. bubble diagram):



Another TM Task: Odd or Even? (State-Transition Table)

The state-transition diagram (a.k.a. bubble diagram):



The state-transition table:

	a	b	∼
evenB	a,R,evenB	b,R,oddB	E,R,done
oddB	a,R,oddB	b,R,evenB	O,R,done

- The start state is listed first in table (e.g., evenB).
- No transitions are possible from state done (thus omitted from table).
- If TM ever encounters a symbol in a state with no specification for that symbol, HALT.

Another TM Task: Increment a Binary Number

Desired behavior of TM:

- Input is a series of 1s and 0s, representing a binary number.
- Tape head starts at the leftmost input symbol, finishes to the immediate left of the leftmost symbol of result.
- TM should increment the binary number by 1.

The general approach:

1. Move tape head to least significant (i.e., rightmost) bit.
2. If the symbol is 0 (or blank), change it to 1 and finish up (step 3).

   If, instead, the symbol is 1, replace with 0, and continue "carrying" for as long as necessary.
3. To finish up, move tape head to left of result.

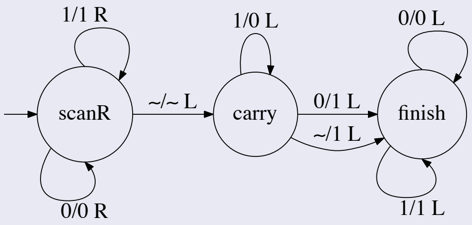
TM: Increment a Binary Number (Bubble Diagram)

The general approach:

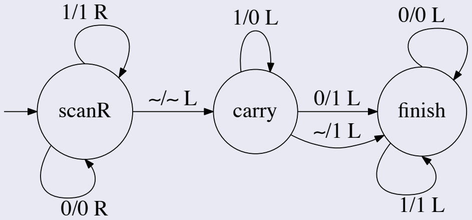
1. Move tape head to least significant (i.e., rightmost) bit.
2. If the symbol is 0 (or blank), change it to 1 and finish up (step 3).

   If, instead, the symbol is 1, replace with 0, and continue “carrying” for as long as necessary.
3. To finish up, move tape head to left of result

The state-transition diagram (a.k.a. bubble diagram):



TM: Increment a Binary Number (State-Transition Table)



The state-transition table:

	0	1	∼
scanR	0,R,scanR	1,R,scanR	∼ ,L carry
carry	1,L,finish	0,L,carry	1,L finish
finish	0,L finish	1,L,finish	

- The start state is listed first in table (e.g., scanR).
- ∼ represents an empty cell (i.e., blank symbol).
- If TM ever encounters a symbol in a state with no specification for that symbol, Halt.