8 Turing Machines

The Turing machine model pre-dates electronic computers and comes from the work of Hilbert (early 1900's) and Turing and Church (1930)'s.

- Not trying to model (electronic) computers, but computation or algorithm
- Identified three essential ingredients:

  1. _____
  2. _____
  3. _____

- implicitly there is a fourth ingredient:

  4. _____

- The Turing machine is a specific formalization of these basic ingredients

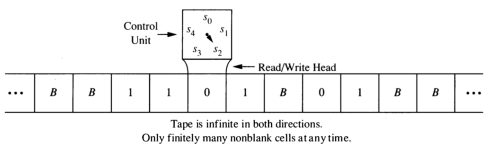
8.1 Definition of a 1-tape Turing Machine (Syntax)

A Turing machine consists of the following:

- **input alphabet**: A finite set of symbols used for inputs.
- **tape alphabet**: Input alphabet plus and at least one additional symbol, the **blank symbol**.
  - It is important that the blank symbol is not one of the input symbols.
  - The tape alphabet may have any finite number of additional symbols beyond the blank symbol and the input alphabet.
  - Traditionally, the blank symbol is denoted $\not{b}$ or b or B or $\sqcup$ or with an underscore.
- **states**: A finite set.
- **initial/start state**: One of the states to start in
- **halting states**: Some of the states are designated as halting state. These can further be divided into accepting states and rejecting states.
- **program**: A list of **instructions**. Each instruction has 5 parts:
  - `<current tape symbol>`: any tape symbol
  - `<current state>`: any non-halting state
  - `<new tape symbol>`: any tape symbol
  - `<new state>`: any state
  - `<direction>`: left, right, or stay
    - Some flavors of Turing machine do not allow stay.

8.2 Execution of a Turing Machine program (Semantics)

Imagine a machine with an infinite tape divided into cells. Each cell contains exactly one symbol from the tape alphabet. At the start of the execution, the input is written on the tape, the read/write head is located at the left most symbol of the input, and all cells that don't contain part of the input contain the blank symbol. The tape may be **1-way infinite**, in which case there is a left end to the tape, or **2-way infinite**, extending infinitely in both directions. Here is a picture of a 2-way infinite tape.



At each step in the execution of a Turing machine program, the read/write head is located at one cell of the tape. The program uses the current state and contents of the tape at that position to determine its action:

- write a symbol in the current cell (overwriting what was there before),
- move the head left or right (or stay put)
- update the state

This is repeated until one of two things happens:

- there is no instruction to execute, or
- the state is a halting state

When either of these two things happens, the machine **halts**. Otherwise it runs forever.

8.3 ASCII Representations of Turing Machines

There are several websites that provide Turing machine simulators. Here are two:

- http://morphett.info/turing/turing.html
  - plain output, and less flexible (initial state must always be called 0, for example), but the program encoding is terse.
- https://turingmachinesimulator.com/
  - nice looking and more flexible, but each instruction takes up 3 lines (current state and tape symbol on one line; new state, direction, and new symbol on the next line; then a blank line)

Each uses a slightly different ASCII representation of a Turing Machine. They are all pretty similar but differ in

- How the input alphabet, tape alphabet, start state and accepting states are described
- How they represent the blank tape symbol. (Often there is a fixed symbol for this like `b`, `B` or `_`.)
- Whether they are case sensitive, disallow certain characters, observe white space, etc.
- How left/right/stay are denoted
- The order in which the 5 parts of an instruction are given
- Which of various variations on the Turing machine theme they support (and how you tell the simulator which flavor to use).

8.3.1 Example 1 (13.5.1 in Rosen 7)

```
; in style of <http://morphett.info/turing/turing.html>
; state symbol symbol direction state
; s0 is initial state in Rosen
0  * * * s0
s0 0 0 r s0
s0 1 1 r s1
s0 _ _ r halt
s1 0 0 r s0
s1 1 1 l s2
s1 _ _ r halt
s2 1 0 r halt
```

1. For each input string below, determine the configuration (tape contents, location of read-write head, and state) of the Turing machine when it halts.
   Do this by hand, then check your work using the simulator.

   a. λ b. 01 c. 0101 d. 010110 e. 001

8.3.2 Example 2

```
; in style of <http://morphett.info/turing/turing.html>
; state symbol symbol direction state
0 * * * [init]          ; [init] is "initial state"
[init] 0 0 r [10]
[init] 1 0 r [01]
[init] _ _ * halt-accept
[00]   0 0 r [10]
[00]   1 1 r [01]
[00]   _ _ * halt-accept
[01]   0 0 r [11]
[01]   1 1 r [00]
[01]   _ _ r [01]
[10]   0 0 r [00]
[10]   1 1 r [11]
[11]   0 0 r [01]
[11]   1 1 r [10]
```

2. For each input string below, determine the configuration (tape contents, location of read-write head, and state) of the Turing machine when it halts. Do this by hand, then check your work using the simulator.

   a. λ
   b. 0110
   c. 01111
   d. 01110

8.3.3 Table notation

Some authors like to represent Turing machines with tables. The goal is the same: we need to specify the actiion of the machine for any tape symbol and state combination. In this table the state and tape symbol are listed in the rows and columns and the action in the interior cells.

3. Convert the Turing machine in the table below

δ	a	b	blank
q0	(q0, a, R)	(q1, b, R)	(qrej, blank, L)
q1	(q1, a, R)	(qacc, b, L)	(qrej, blank, L)

into an ASCII representation for use at http://morphett.info/turing/turing.html.

8.4 The language recognized by a Turing Machine

We will say that a Turing machine M accepts a string x (by state) if on input x it halts in an accepting state. The language recognized by such a Turing machine is

$$L(M) = \{x \mid M \text{ accepts } x\}$$

4. If $x \notin L(M)$, what are the two ways the machine M might behave on input x?
5. What language does the machine in Example 2 above recognize?

8.4.1 Designing a Turing Machine

For each of the following, design a Turing machine that always halts and use one of the simulators to run your program on some test strings. Explain in words what each of your states is for.

6. Design a Turing machine that recognizes $(\mathbf{0} \cup \mathbf{1})\mathbf{1}(\mathbf{0} \cup \mathbf{1})^*$. Is every regular language recognized by a Turing machine? How do you know?
7. Design a Turing machine that recognizes $\{0^n 1^n \mid n \geq 0\}$.
8. Design a Turing machine that has input alphabet { `(`, `)`, `0`, `1`, `+` } and recognizes strings where the parentheses are properly nested.
   - Note: You don't have to do a full check that the input is a valid expression (that would be a bit more involved, go ahead and add that if you like). The intent here is just to do the parenthesis matching part.
   - Bonus: See how few states you can use.

8.5 Computing functions with a Turing machine

Turing machines can also compute functions. The output of a Turing machine is the tape contents when it halts.

9. Design a Turing machine that adds in unary. Use the input alphabet $\{1, +\}$.

   In unary, a non-negative integer n is represented by $n + 1$ 1's. So 3 is represented as 1111.

   Example: Our machine should convert `111+1111` into _____.

8.5.1 Neat computation

A Turing machine computes **neatly** if when it halts (a) the read/write head is on the left-most non-blank tape symbol (or any blank symbol if the tape is all blanks), and (b) there are no internal blanks (ie, all the non-blank symbols are contiguous on the tape).

10. Modify your previous machine so that it computes neatly. What is the advantage of neatly computing Turing machines?
11. Write a Turing machine that neatly computes the function $f(x) = 2x$ in unary.