Design a PDA which accepts:

$L = \{ u\, u^R\, a^n\, b^p\, c^n : u \in \{a,b\}^+, n \geq 1, p \geq 2 \}$

Design a PDA which accepts:

$L = \{ u\, u^R\, a^n\, b^p\, c^n : u \in \{a,b,c\}^+,\ n \geq 1,\ p \geq 2 \}$

Is

cabbaabbcc

in L?

Does your machine accept this string?

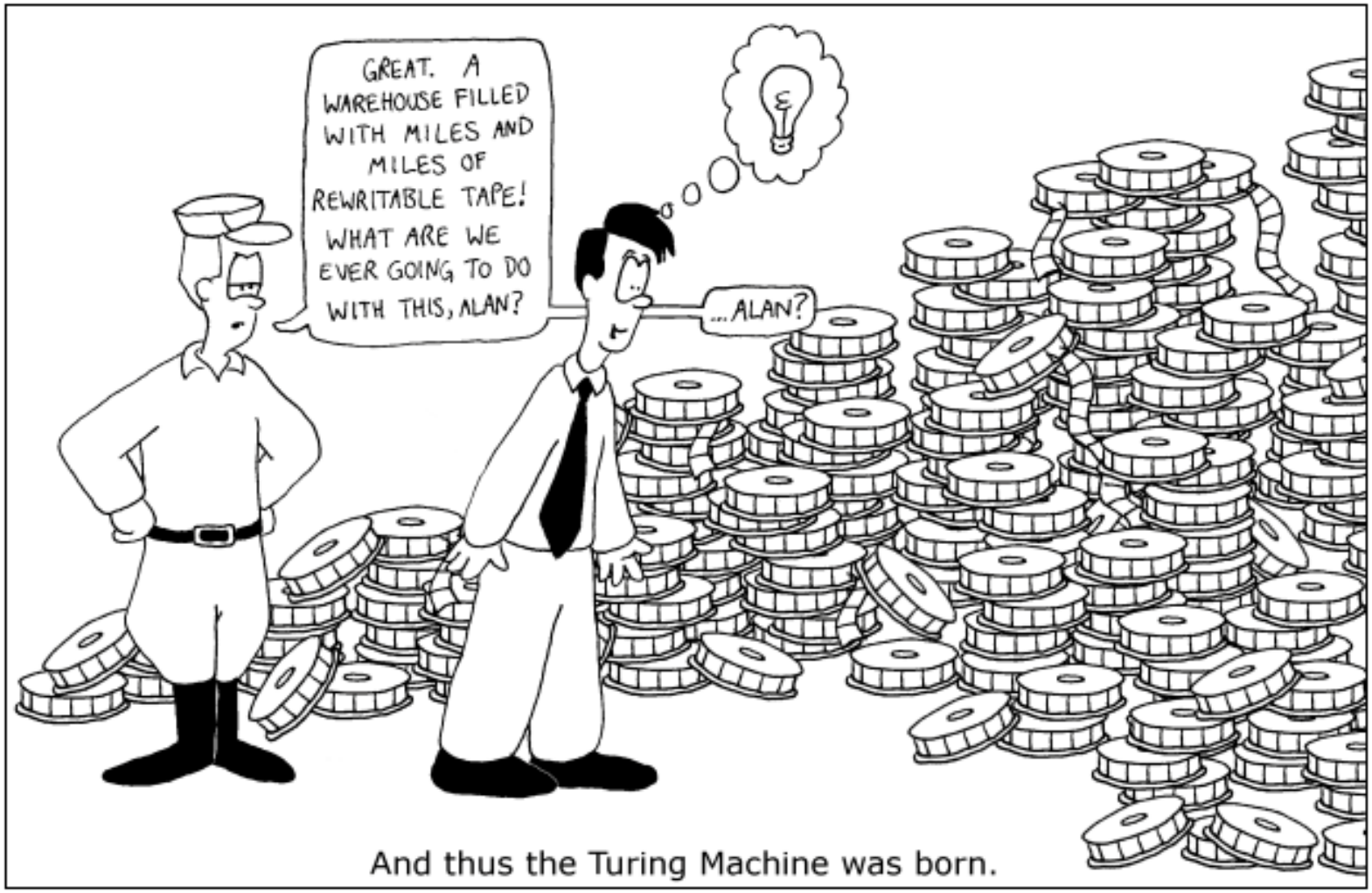
# Announcements:

My TM definitions are slightly different from the book- use my definitions on assignments and with the TM simulator.

Assignment #4: Due at the beginning of class, Fri. July 14. Recall that you need a passing average on your top 4 assignments to pass the class.

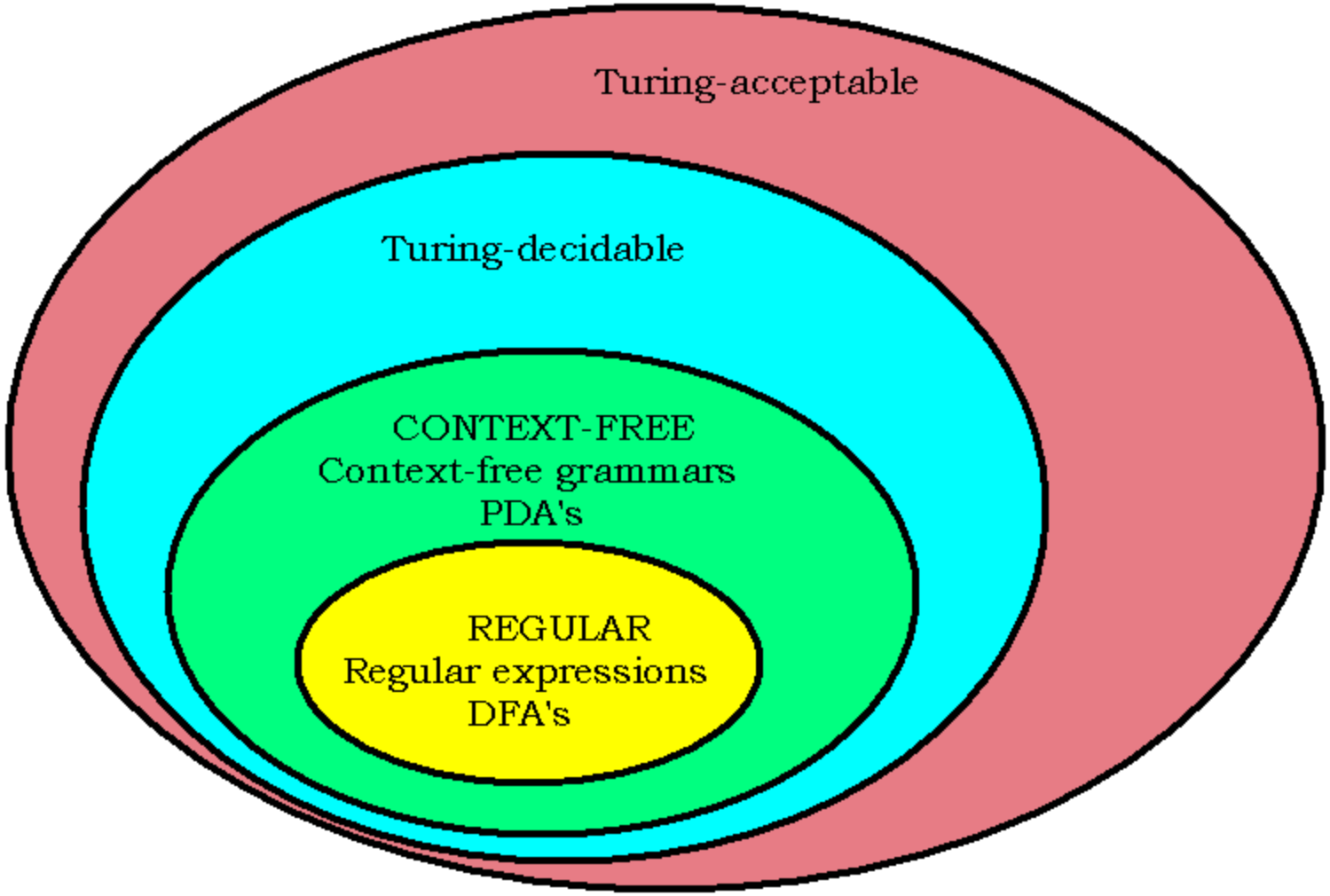
There is a tutorial on Tuesday July 11. Bring any questions you have about the assignment.

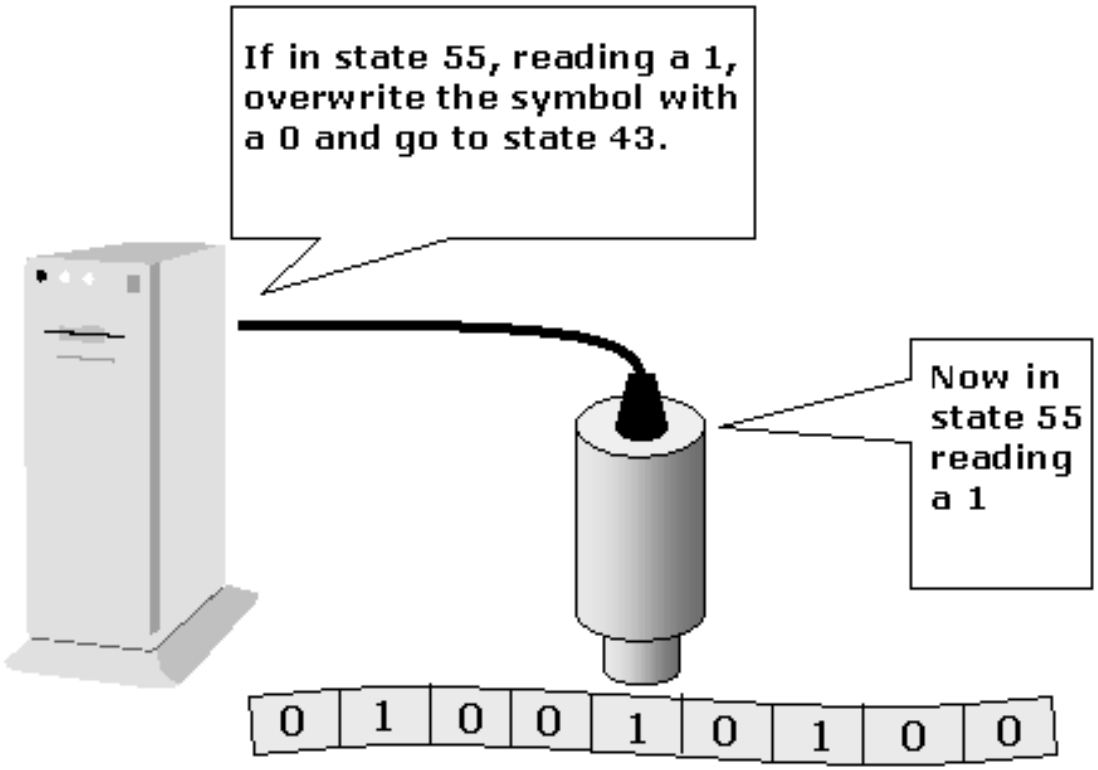
# Turing Machines




This cartoon appears in the book Computation Engineering: Applied Automata Theory and Logic by Dr. Ganesh Gopalakrishnan.

# Classes of Languages



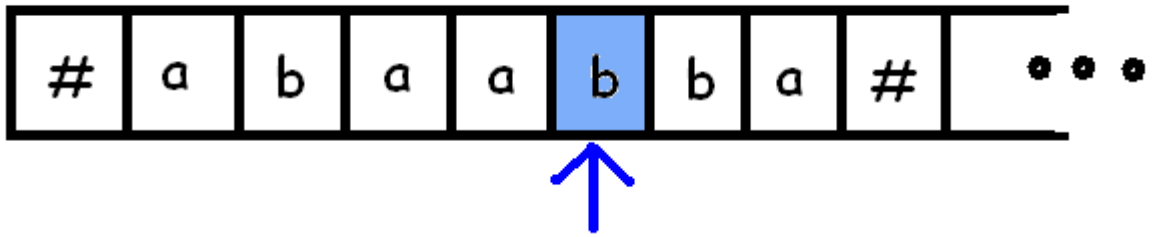



http://www.lambdassociates.org/webbook/chap1.htm

Turing machines are very simple (so it is easy to prove things about them) but are as capable as any current computer.

# Turing machines:

## Operation: add read/write tape to DFA.




Move:

Based on current state and symbol scanned:

1. change states, and
2. either replace tape square contents with a symbol or move head one square left or one square right.

## Input conventions:

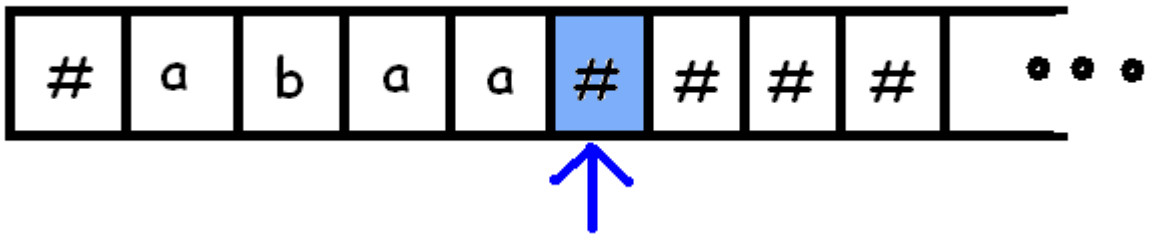
Tape has a left hand end but extends infinitely to the right (one-way infinite tape).

All squares are initially blank- represented by the symbol #.

On an input w, the tape starts out as:

# w [#]  (the [] contain the symbol scanned).

For example, on input w= abaa:




Special halt state called h.

$L = \{ u\, u^R : u \in \{a, b\}^* \}$.

Design a TM which accepts L:

It halts on input w if w in L and either

hangs or computes forever when w is not in L.

A TM halts immediately when entering the special halt state h.

A TM hangs if the head falls off the left hand end of the tape or if it encounters an undefined transition.

## Pseudo code for my algorithm:

1. Find leftmost input symbol and erase it.

   If there is no input left at this stage, halt.

2. Move head to right hand end of input using the state to remember the symbol at the left hand end of the tape.

3. If the symbol does not match, make the TM hang. Otherwise erase the symbol and go back to step 1.

TM rules: State Symbol Next state Head instruction

// Move left to leftmost symbol.

| | | | | |
|---|---|---|---|---|
| Rule 1: | start | # | left | L |
| Rule 2: | left | a | left | L |
| Rule 3: | left | b | left | L |

// When we find # at LH end, move R to check LH symbol.

| | | | | |
|---|---|---|---|---|
| Rule 4: | left | # | check | R |

// If blank we are done- halt.

| | | | | |
|---|---|---|---|---|
| Rule 5: | check | # | h | # |

// Blank out LH symbol and remember it using state.

| | | | | |
|---|---|---|---|---|
| Rule 6: | check | a | rema | # |
| Rule 7: | check | b | remb | # |

// Move off blank remembering LH symbol using state.

| | | | | |
|---|---|---|---|---|
| Rule 8: | rema | # | righta | R |
| Rule 9: | remb | # | rightb | R |

```
// Move to RH symbol remembering symbol using state.
Rule  10: righta  a  righta  R
Rule  11: righta  b  righta  R
Rule  12: rightb  a  rightb  R
Rule  13: rightb  b  rightb  R
// Check symbol at RH end.
Rule  14: righta  #  checka  L
Rule  15: rightb  #  checkb  L
// If it matches, blank it out and start over.
Rule  16: checka  a  start   #
Rule  17: checkb  b  start   #
// If not, hang because no transitions defined from undef.
Rule  18: checka  b  undef   b
Rule  19: checka  #  undef   #
Rule  20: checkb  a  undef   b
Rule  21: checkb  #  undef   #
```

# An accepting computation:

(start, #abba[#])
- (left, #abb[a])
- (left, #ab[b]a)
- (left, #a[b]ba)
- (left, #[a]bba)
- (left, [#]abba)
- (check, #[a]bba)
- (rema, #[#]bba)
- (righta,##[b]ba)
- (righta,##b[b]a)
- (righta,##bb[a])
- (righta,##bba[#])
- (checka,##bb[a])
- (start, ##bb[#])
- (left, ##b[b])
- (left, ##[b]b)
- (left, #[#]bb)
- (check,##[b]b)
- (remb, ##[#]b)
- (rightb,###[b])
- (rightb,###b[#])
- (checkb,###[b])
- (start,###[#])
- (left, ##[#])
- (check,###[#])
- (h, ###[#])

# A non-accepting computation:

1. (start, #aaba[#])
2. (left, #aab[a])
3. (left, #aa[b]a)
4. (left, #a[a]ba)
5. (left, #[a]aba)
6. (left, [#]aaba)
7. (check, #[a]aba)
8. (rema, #[#]aba)
9. (righta, ##[a]ba)
10. (righta, ##a[b]a)
11. (righta, ##ab[a])
12. (righta, ##aba[#])
13. (checka, ##ab[a])
14. (start, ##ab[#])
15. (left, ##a[b])
16. (left, ##[a]b)
17. (left, #[#]ab)
18. (check, ##[a]b)
19. (rema, ##[#]b)
20. (righta, ###[b])
21. (righta, ###b[#])
22. (checka, ###[b])
23. (undef, ###[b])

No valid transition,
TM hangs.

# Input to TM simulator:

```
// This TM halts if the input is of the form u u^R where u
// is in {a,b}* and hangs otherwise.
start
// Move left to leftmost symbol.
start # left L  // start algorithm to check if u u^R
left a left L   // Go left to LH end of input
left b left L   // Go left to LH end of input
// When we find # at LH end, move R to check LH symbol.
left # check R // Found # at LH end
// If blank we are done- halt.
check # h # // Done and answer is yes if input is all blank
// Blank out LH symbol and remember it using state.
check a rema # // Blank out LH symbol remember "a" using state
check b remb # // Blank out LH symbol remember "b" using state
// Move off blank remembering LH symbol using state.
rema # righta R // Start going right (remember "a")
remb # rightb R // Start going right (remember "b")
```

```
// Move right to RH symbol remembering LH one using state.
righta a righta R // Move right (remember "a")
righta b righta R // Move right (remember "a")
rightb a rightb R // Move right (remember "b")
rightb b rightb R // Move right (remember "b")
// Check symbol at LH end.
righta # checka L // Found RH end- check now if "a"
rightb # checkb L // Found RH end- check now if "b"
// If it matches, blank it out and start over.
checka a start # // First symbol matches last one- start again
checkb b start # // First symbol matches last one- start again
// If not, hang because no transitions defined from undef
checka b undef b // No match- move to state with no transitions defined
checka # undef # // No match- move to state with no transitions defined
checkb a undef b // No match- move to state with no transitions defined
checkb # undef # // No match- move to state with no transitions defined
$
abba // A string which is in L
aaba // A string which is not in L
```

# The format of the input to the TM simulator is as follows:

<Name of start state>
<current state> <current symbol> <next state> <head instruction>
...
<current state> <current symbol> <next state> <head instruction>
$
<input string w1>
<input string w2>

...

# Rules for TM Descriptions

1. The state name h is used to denote the halting state.
2. Use the symbol # to represent a blank.
3. If a line starts with // it is a comment. Comments can also be added on the same line as an instruction at the end of the line (start with //).
4. Each of the state names is an arbitrary string. The current symbol and new symbol each must be a single symbol.
6. The head instruction is either L (move the head one square left) or R (move the head one square right) or a symbol to replace the current tape square contents.
7. The $ indicates the end of the TM description.