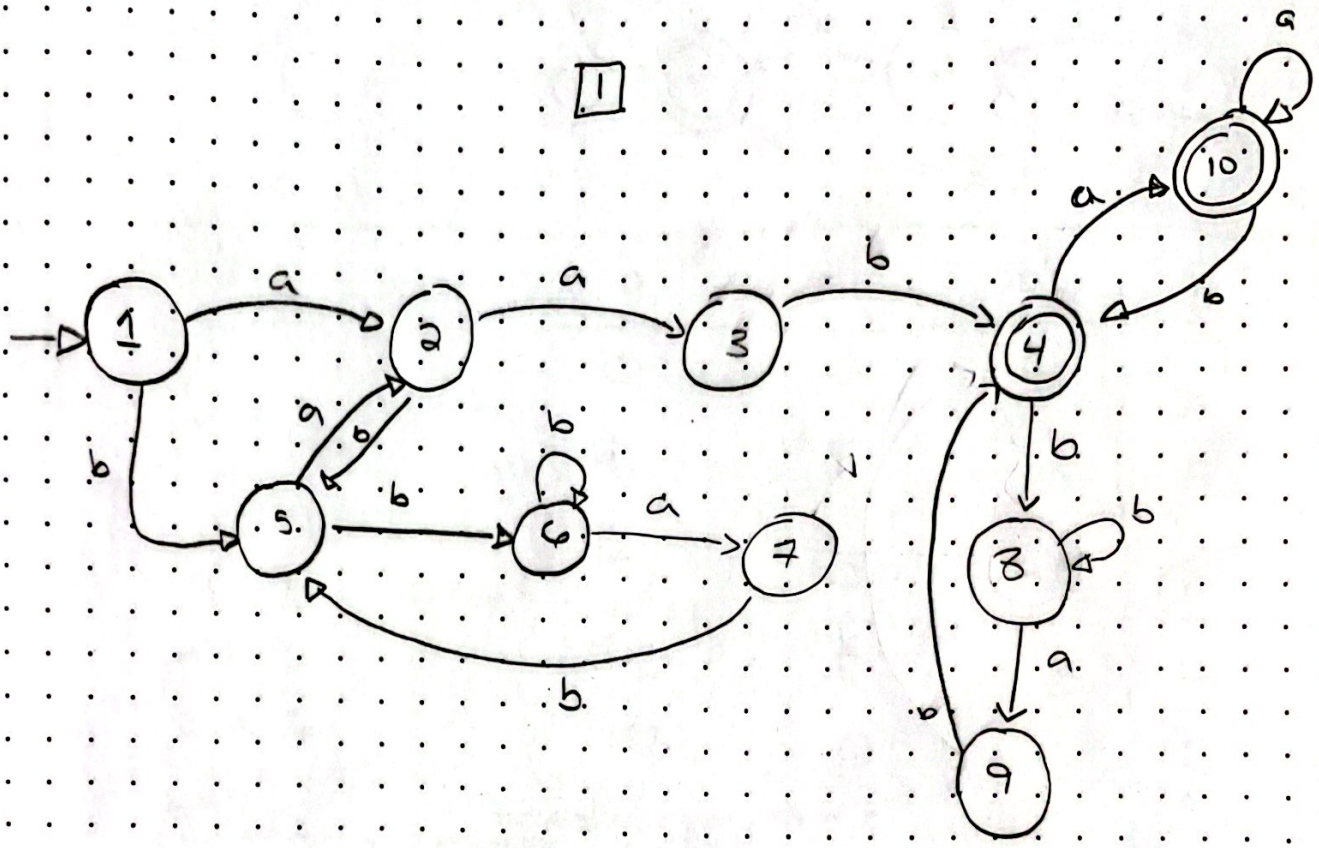
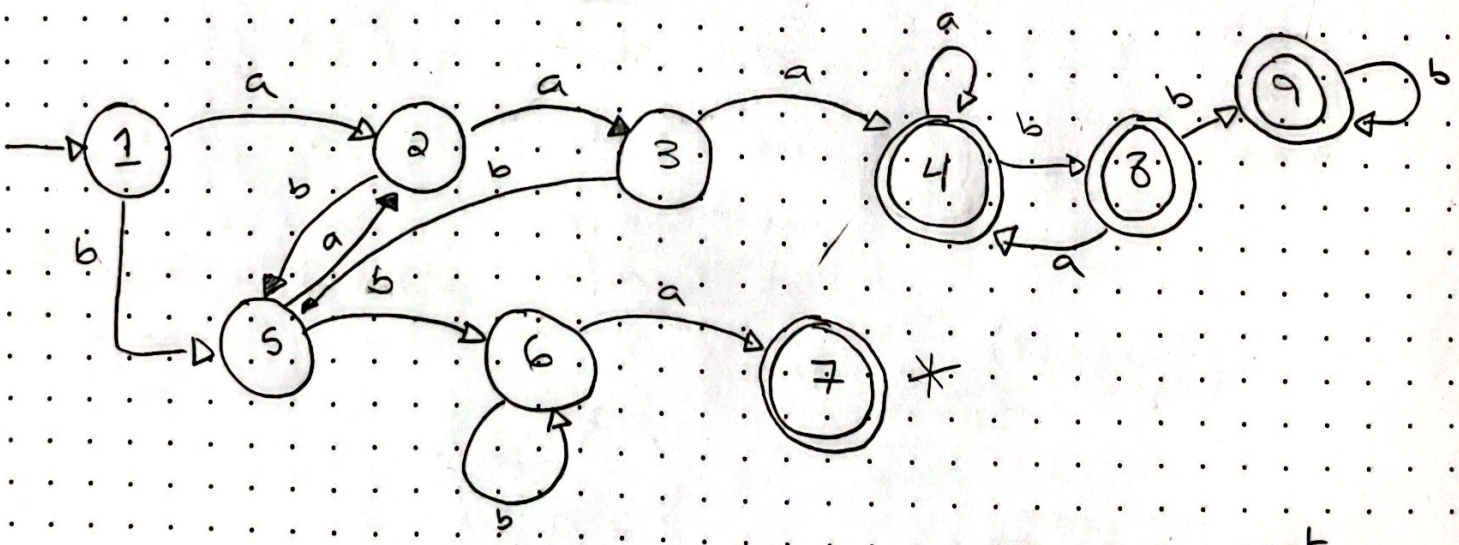
Lecture 2

DFA's

① - has 'aab' but not 'bbaa'

② - has 'bba' or 'aaa' but not both



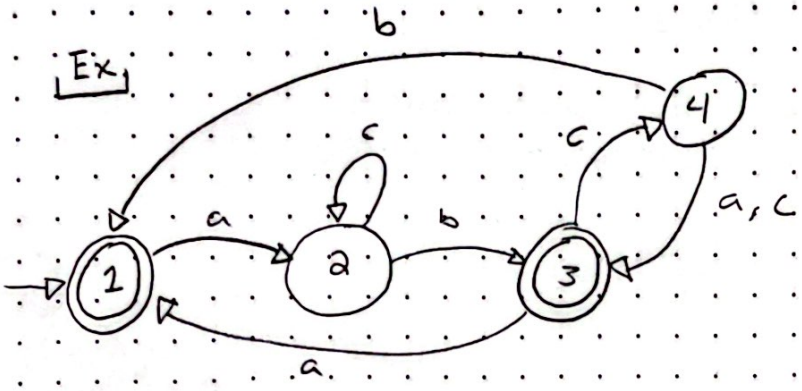


* not finished on bottom half. Does not prevent 'aaa' at this point

One pass through data, cannot go back through string

$\Sigma = \{a, b, c\}$

Ex



δ	a	b	c
→ 1	2	0	0
2	0	3	2
3	1	0	4
4	3	1	3
0	0	0	0

strings

abb

cab

bac

baac

```
string alphabet = "abc";
int state_cnt = 4;
int start = 1;
bool final[] = { f, t, f, t, f }
int cur = start;
int trans[state_cnt + 1][alphabet.length()] = {
    { 0, 0, 0 },
    { 2, 0, 0 },
    { 0, 3, 2 },
    { 1, 0, 4 },
    { 3, 1, 3 }
};
```

```
if (final[cur])
```

```
char *s = "abaab"

while (*s != '\0' && cur != 0)
{
  cur = trans[cur][alphabet.find(*s)];
  s++;
}
```

---

file

```
abc      - alphabet
4        - # states
1        - start state
1 3      - accept states

0 3 2
4 1 1    > transition table
0 0 3
1 2 3

aab
bcaaa    > data
caa bbbbb
```

empty string

output (example)

```
good  aab
bad   bcaca
bad   <empty>
good  caabbbbb
```

Jan 23rd

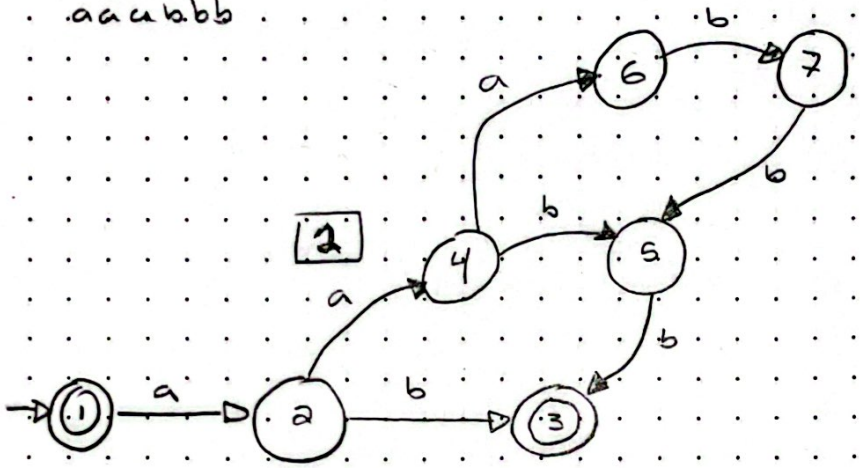
DFA's cannot remember data / inputs

examples of limitations of DFA's

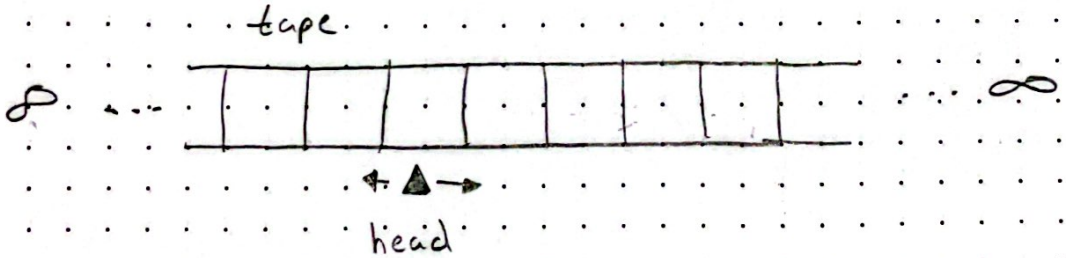
① - palindromes
② - $a^n b^n$

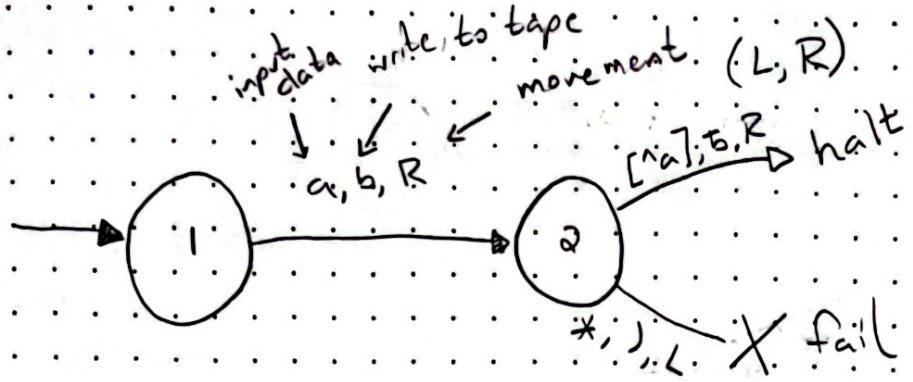
> both impossible to do w/ DFA's (no storage)

$\in$
ab
aabb
aaabbb



Turing Machine





input data

members of alphabet

- a
- [a c]
- [^ ab]
- *
- b (blank)

write/draw

- some member of alphabet
- * (copy of input)
- b (blank)

choose either 'a' or 'c'

neither 'a' or 'b'

anything goes.

Example

① - $a^m n^m$

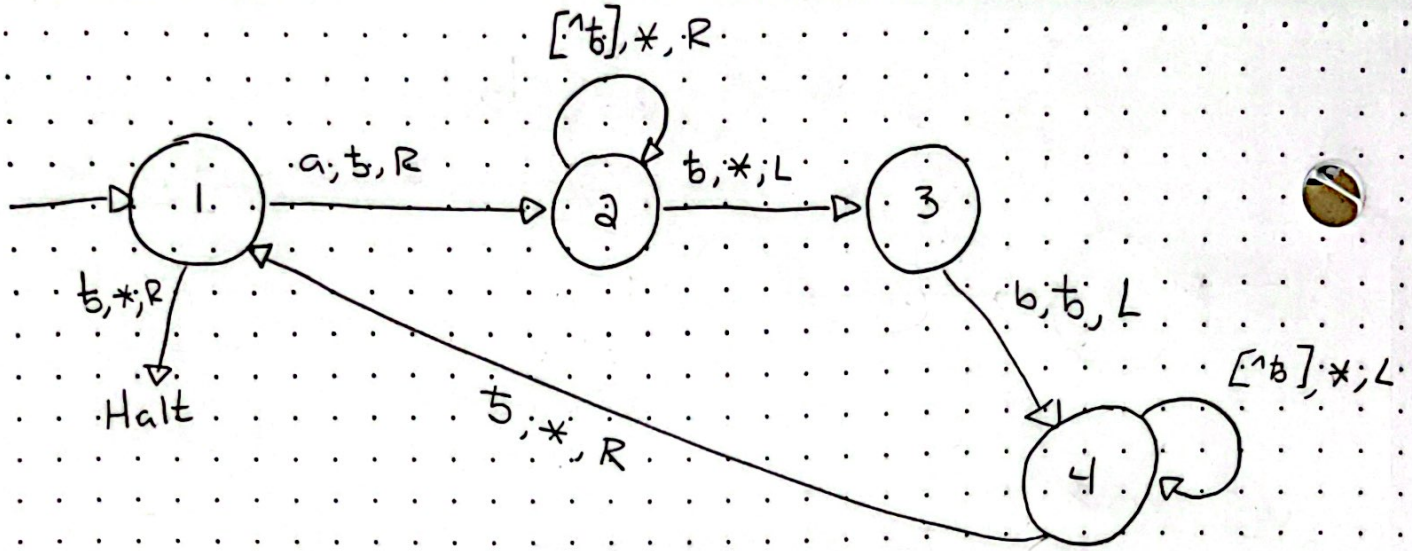
② - $a^m b^n$, $N < m$

1

- erase pairs of a's and 'b's
- check if any chars remain

␣ a a a b b b ␣

a a b b b



2

aaab ✓

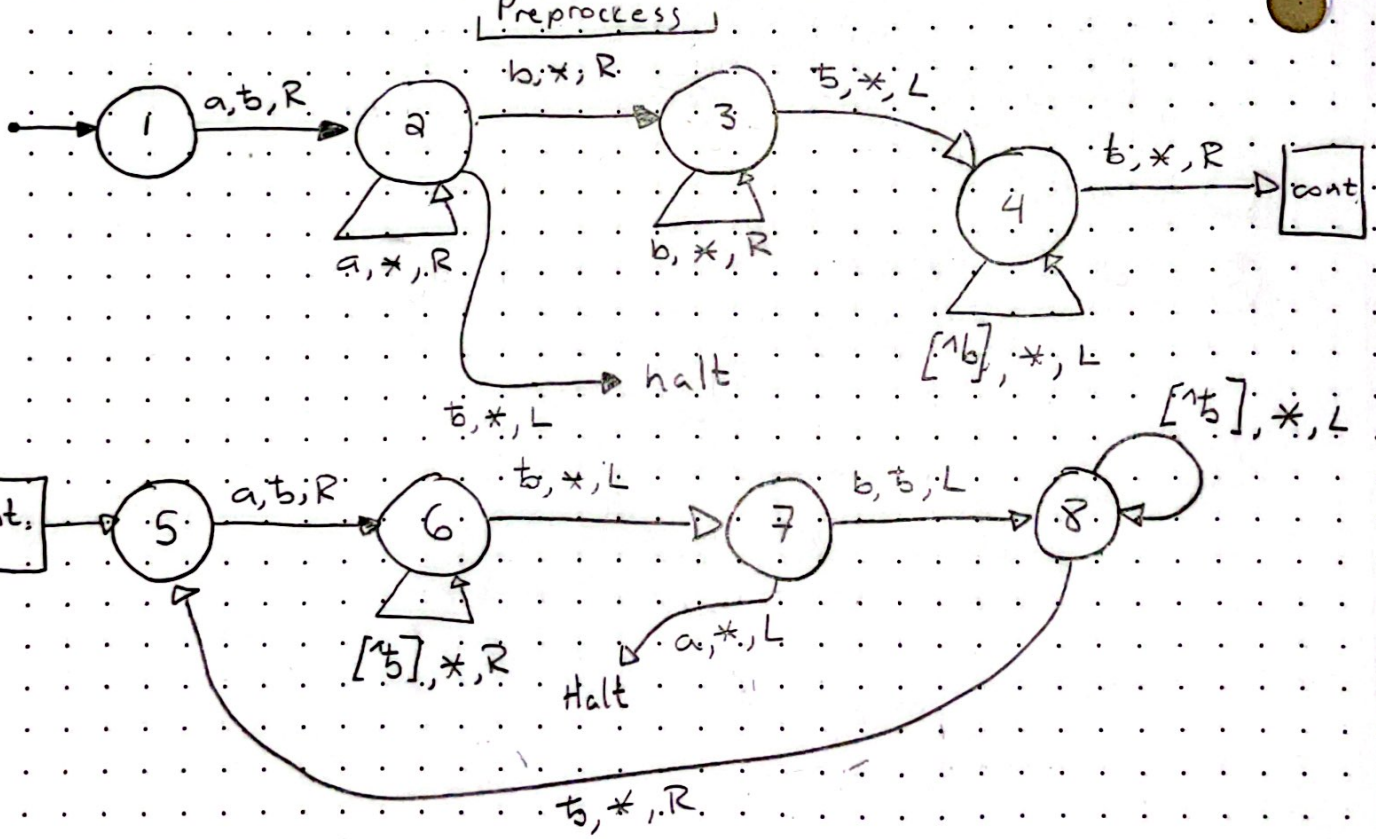
aaaabbb ✓

a ✓

ϵ ×

aabb ×

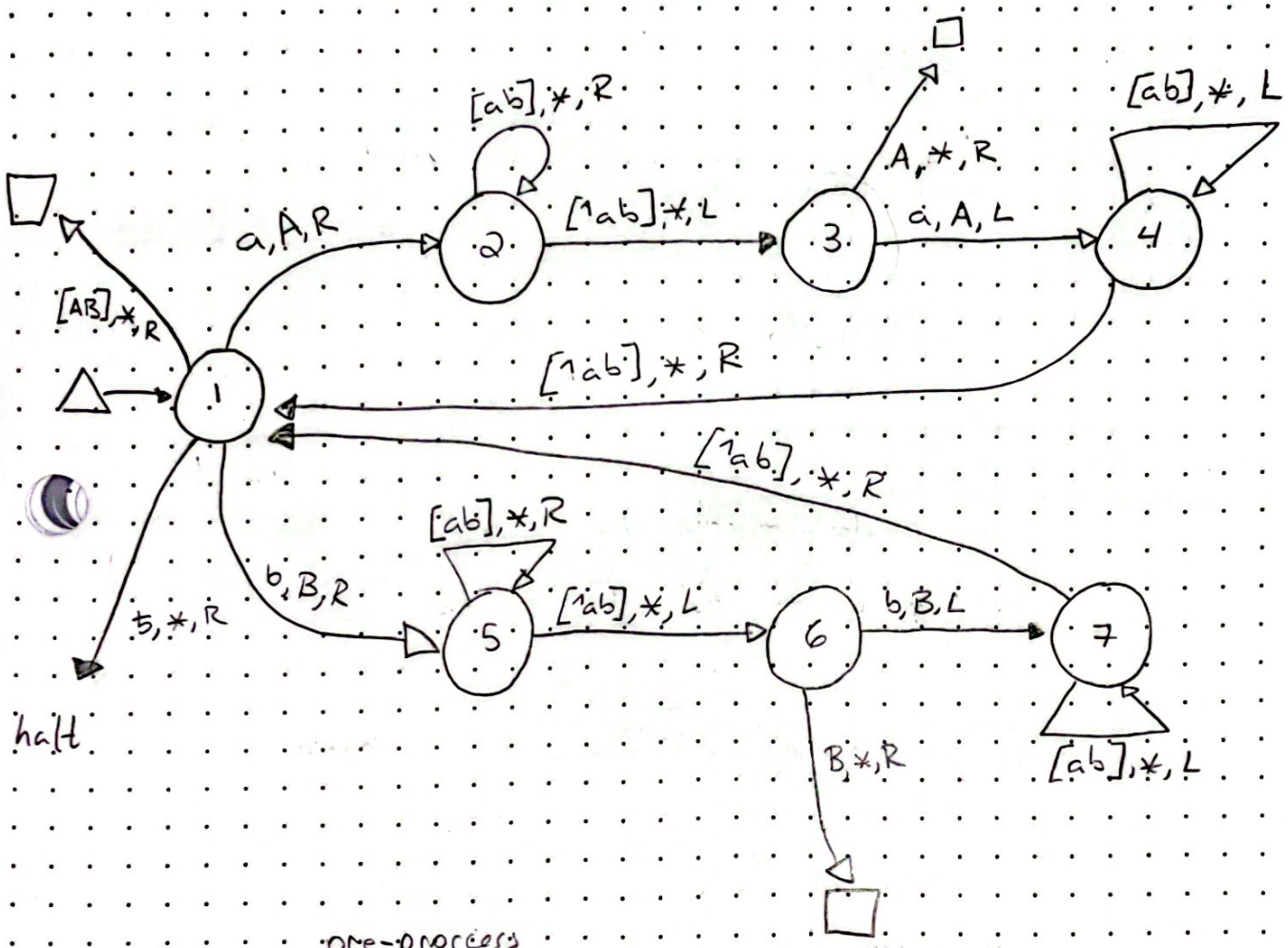
b ×



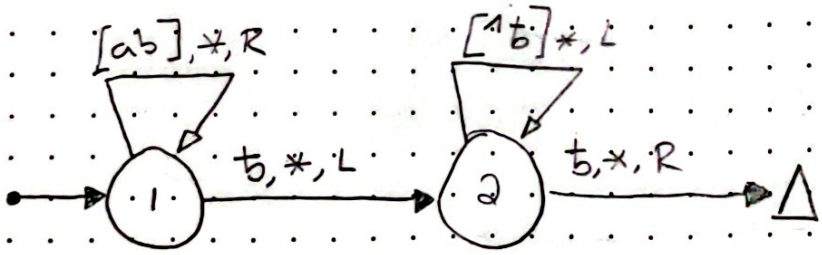
Jan 25th

TM to find palindromes

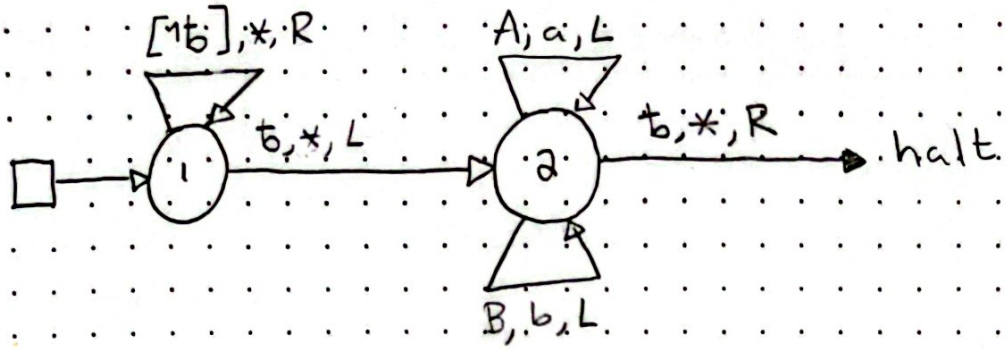
- abba
- bab
- ϵ
- a
- aaba
- baa



pre-processor



Post-process



example 2

abaaabbaaaabababb

we want to remove "runs" of letters

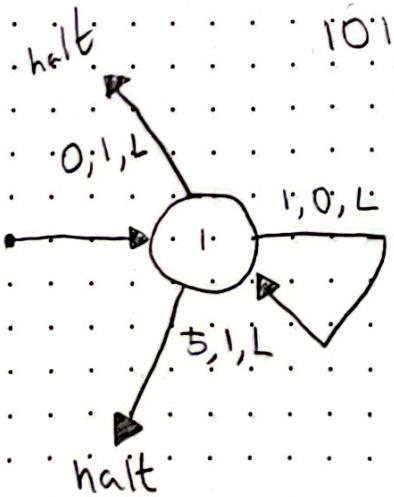
ababababab

we could rewrite our string in the whitespace to the right of the sting

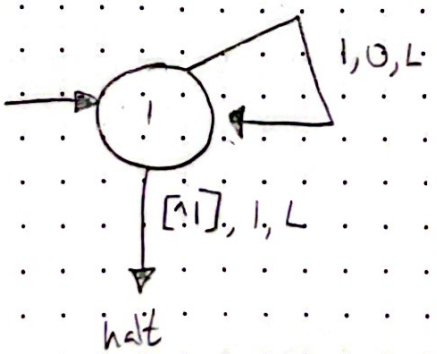
~~abaaabb~~ a a a a b a b b b _ _ ababababab

example 3

1011 ^ we want to add '1'



optimized



IO is all buffered (important to know for debugging/security)

g++ -g ~~~.cpp -fsanitize=address

tells compiler to compile in debug mode

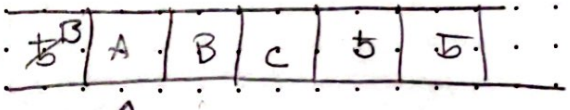
Building our own VM

bool alphabet[256]

tape — alphabet — Equals flag

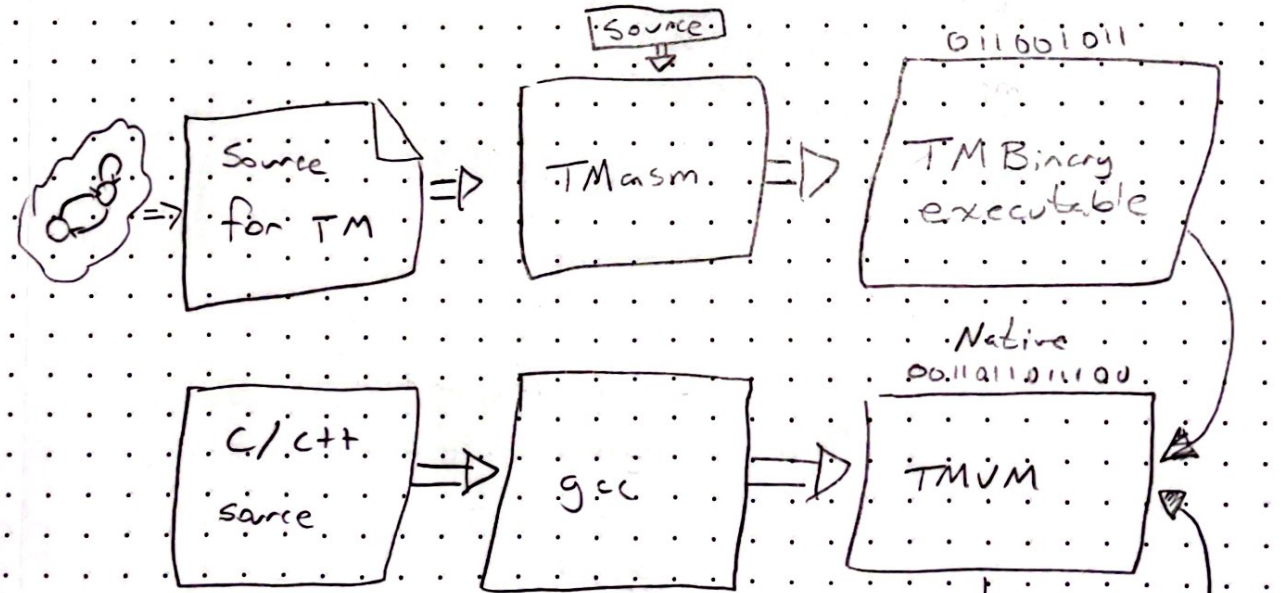
head — RAM

State transitions



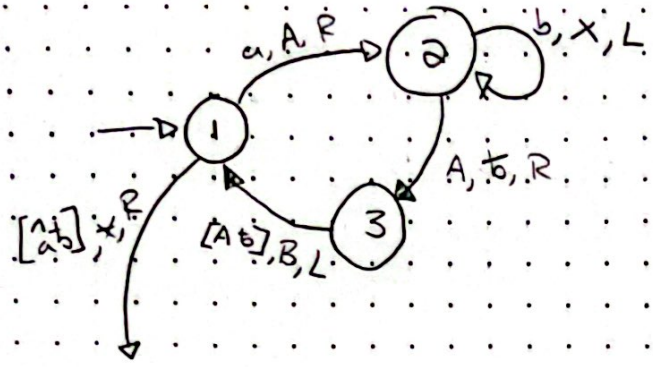
0 1 2 3 ...

0 1 2 3 4 ...



description of how our sim works

success / tape state



example of assembly

```
    mov   ax, 4
!top
    cmp   ax, 0
    jle   !done
    add   bx, cx
    dec   ax
    jmp   !top

!done
```

! = label (location in memory we might need to return to)

instruction ↓ mov

parameters ↓ ax , 4

Flags (will be 1 or 0 for T or F.)

S	G	L	E	Z

S - sign (neg or pos)
G - greater than
L - less than
E - equal to
Z - is zero

Δ → ! 1

```
cmp   'a'
je    !1_a
cmp   'a', BLANK
jne   !1_^a BLANK
```

```
alpha 'a'
"     'b'
"     'A'
"     'X'
```

Thurs.

Feb 1st

Assembly for DFA

alpha "ababAXB"

```
alpha 'a'
alpha "bAXB"

!1
    cmp   'a'
    je    !1_a
    cmp   'a'
    OR    BLANK
    jne   !1_^aBLANK
    fail

!1_a
    draw  'A'
    Right
    jmp   !2

!1_^a BLANK
    Right
    halt.
```

```
!2
    cmp   'b'
    je    !2_b
    cmp   'A'
    je    !2_A
    fail

!2_b
    draw  'X'
    Left
    jmp   !2

!2_A
    draw  BLANK
    Right

!3
    cmp   'A'
    OR    BLANK
    je    !3_A BLANK

!3_ABLANK
    draw  'B'
    Left
    jmp   !1
```

opcode

0 alpha – letter

1 cmp – letter / BLANK is_blank is_cmp

OR –

2 jmp – (11) address is_eq is_ne

jne –

jmp –

3 draw – letter / BLANK is_blank

4 move – signed_amt

5 stop – is_halt

1 1 unconditional

0 0 do nothing

(arrows mean instruction has been converted into parameter of previous instructions)