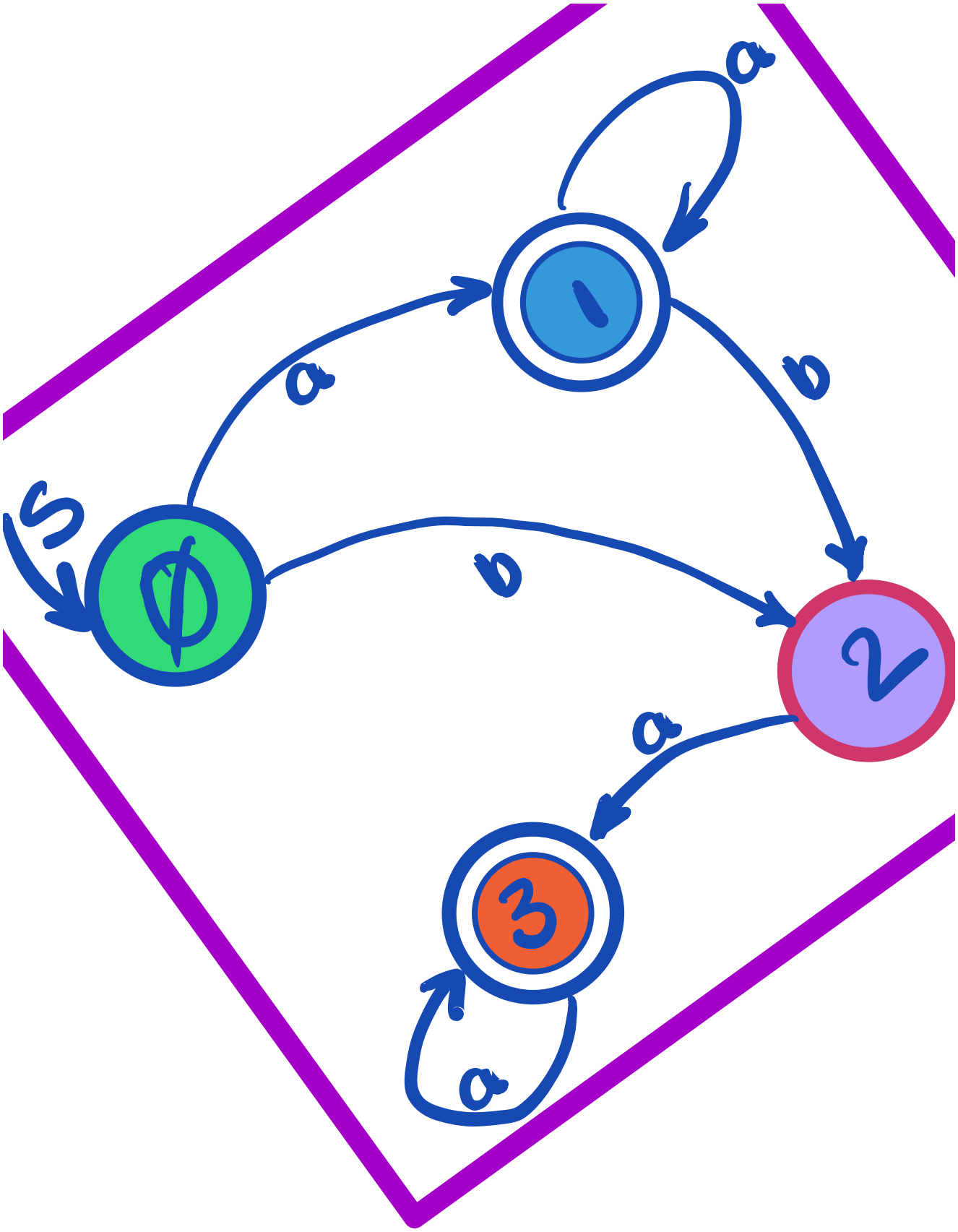

a
1
a
b
s
0
b
2
a
3
a

Problem:

A bat and a ball cost $1.10
the bat costs $1.00 more than the ball
how much is the bat?
how much is the ball?

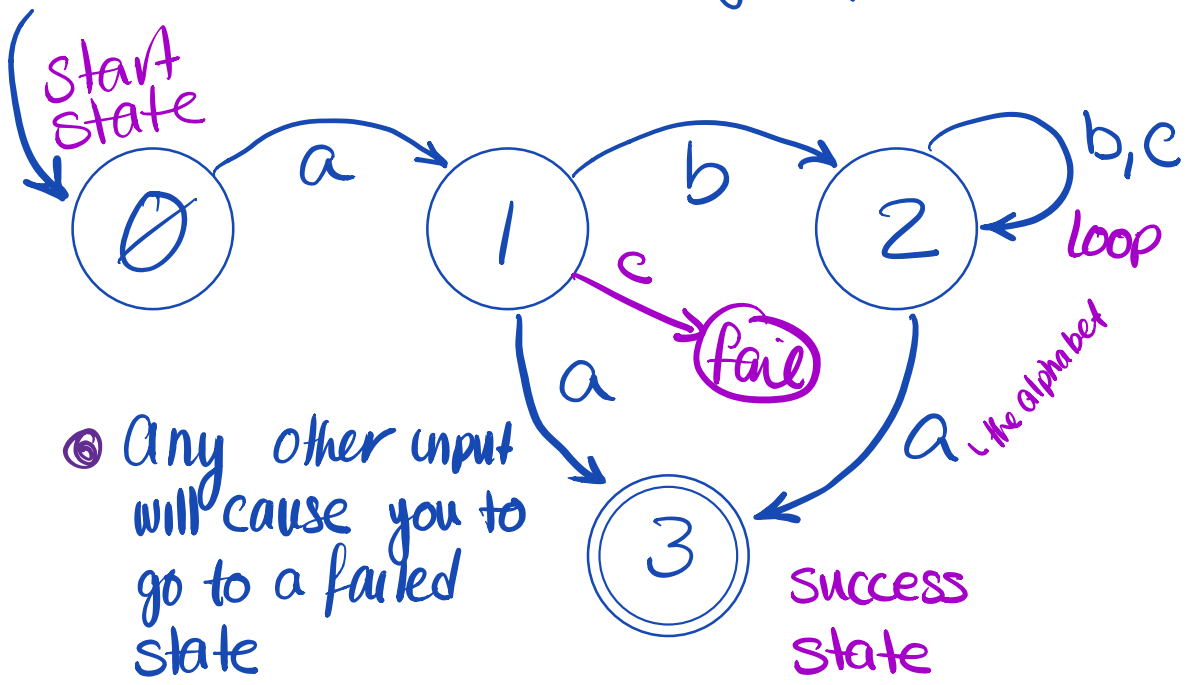
$$\begin{cases} x + y = 110 \\ x - y = 100 \end{cases}$$

easy to solve once the problem is modled by this system of equations

Not so easy without

Finite state machines:
– a computation Model –

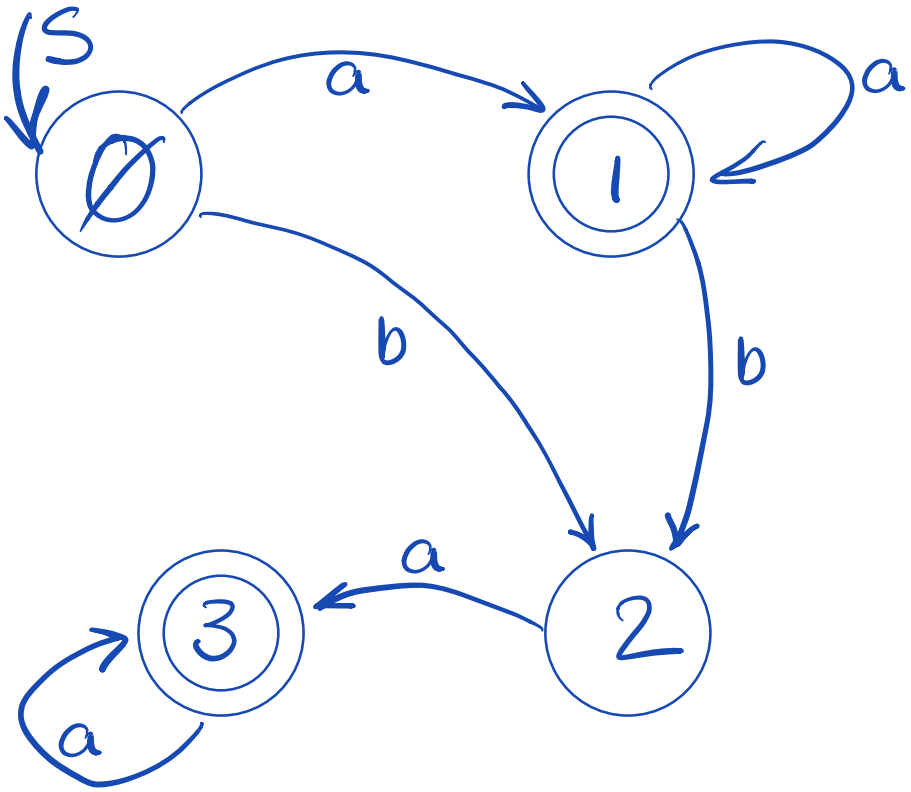
- finite number of states (bubbles)
- move from one state to another based on the input (archs)
- states are numbered & arcs, transitions are marked by input values



- any other input will cause you to go to a failed state

aa
aba
abca
abcccca
abcbcba

Example

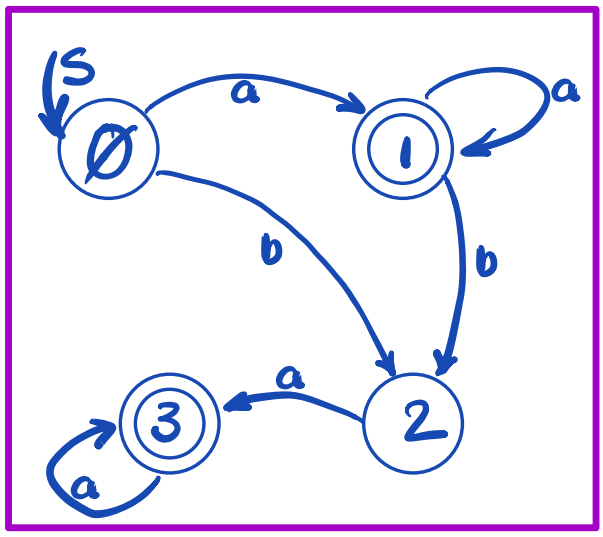


a
aaaa
aab ✗
aabaaa ✓
baaa ✓

use this machine to
- generate valid strings
- validate strings
- extract valid strings

aaabaaacdef

Process:
find the longest 'valid' string from input:



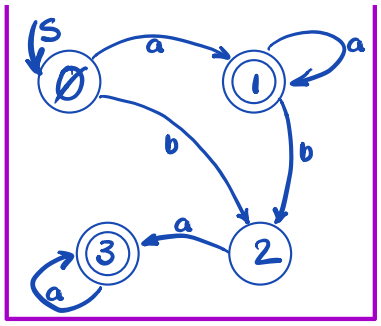
▼a
ab
b
baaac
babbc
aaabaaabcc
aabbaaaccc

we call this longest string a token.
once the token is extracted, the 'pos' is left at the character following the last character in the token:

▼aaabaaabcc

aaabaaa▼bcc

the table:



row numbers are state numbers

column ∅: t/f: success state

Pos

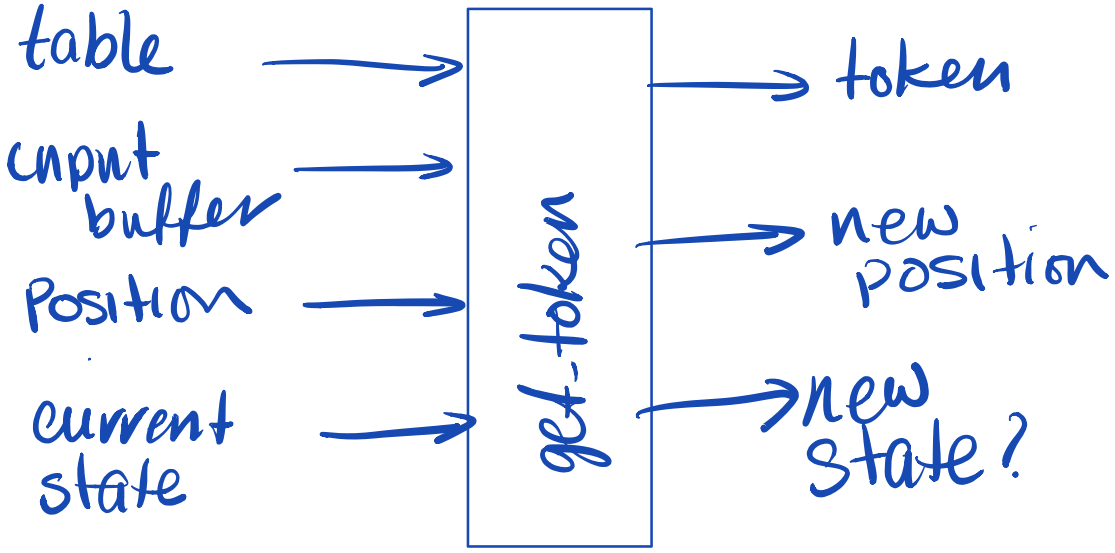
baaac

state=∅

state	∅	col		'a' 97	'b' 98		127
∅	0	← -1 →		1	2	← -1 →	
1	1	← -1 →		1	2	← -1 →	
2	0	← -1 →		3	-1	← -1 →	
3	1	← -1 →		3	-1	← -1 →	
4 ⋮							

- table columns 1 .. 127 represent ascii values for input
- table column zero holds success state markers: 1 for success, zero for not
- table rows ∅ .. n represent states of the machine
- table cells at table[s][c] indicate that if you are at state s & your input is c, go to this state. Table is init'd to -1's.

get_token



this function will take an input buffer, a starting position, a starting (current) state and the adjacency table.

when we are done,
token will hold the token at our last successful state,
position will be the position after the last successful state.
and return true if a token was successfully extracted. false otherwise.
obviously, if no token is extracted, position will remain unchanged.

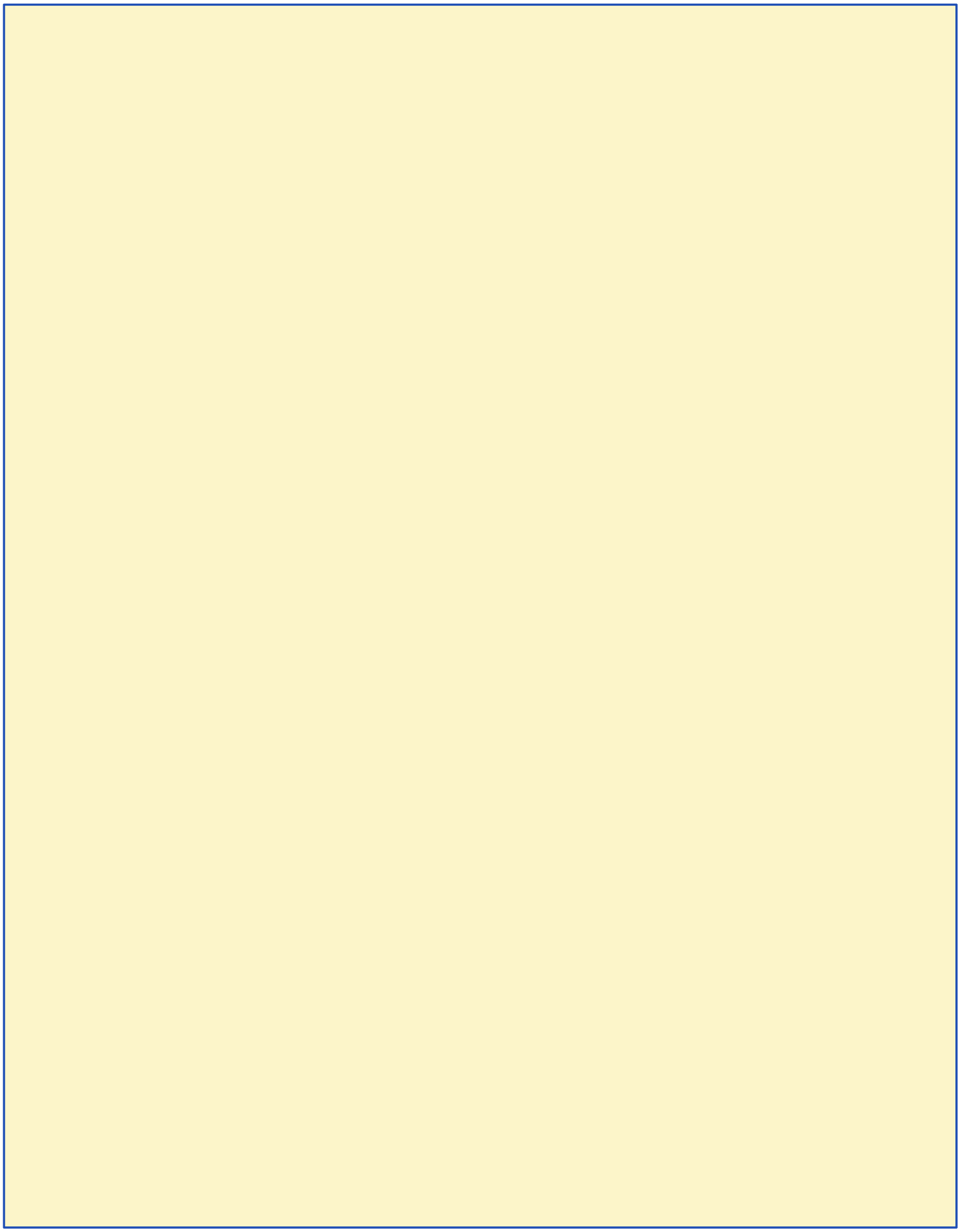
get_token

- a very simple, but delicate function:

  simply loop over the chars in the input array.
  - hop from one state to the next
  - remember the last position in the input that took you to a success state.

  stop the loop when you arrive at a fail state (-1) or you reach end of input.

get_token



managing the table:

unit → -1

mark_success : mark this row(state) as success

mark_cells :
given table, row & range of cols set them to this value:
mark_cells(state, table, 'a', 'z', 3)

these and all the other functions help you encode the the state machine in the table.

STokenizer class

```
STokenizer stk(buffer);
Token t;
stk >> t;  // get first token
while (!stk.done){
    // process token
    cout << "|" << t << "|" << endl;

    t = Token();
    stk >> t;     // get next token
}
```