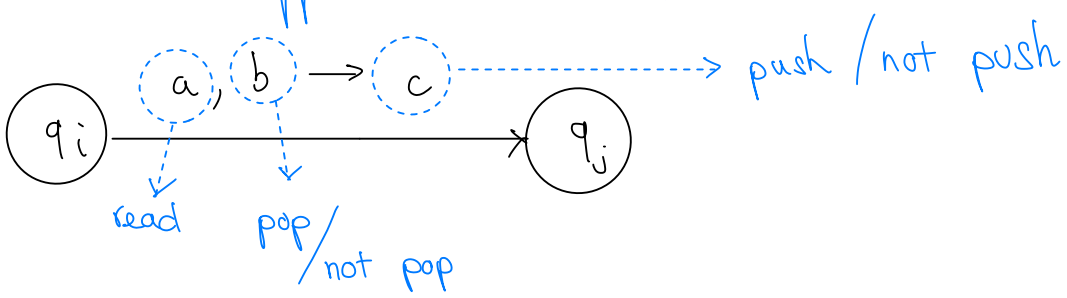
PUSHDOWN AUTOMATA & GRAMMAR FLOW GRAPH

Pushdown Automata: (PDA)

Used for Context-free language. The only different between pushdown automata and finite automata is the stack.

How a trancition happen in PDA?



Example: Recognize this simple language

$S \rightarrow a\ S\ b$ (equal number of a's followed by equal number of b's)

- "a a b b" ✓
- "a b" ✓
- "a b a" ✗

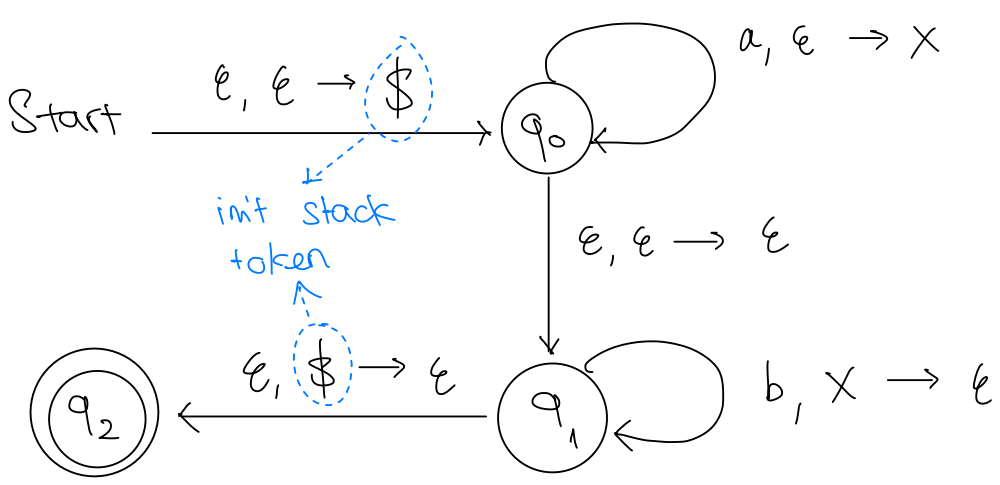
→ Finite Automata cannot recognizes this language because it has no memory beyond its current state (and we need the memory to evaluate if a's and b's are equal)

And: Assume Finite Automata accept this string $a^{\wedge}k\ b^{\wedge}k$, which is a valid string, then it must also accept $a^{\wedge}k+1\ b^{\wedge}k+1$

Clearly, we have infinite accepting states here so Finite Automata cannot capture this

```
→(q0)--0→(q1)--0→(q2)--0→(q3)--0→...infinite states needed
        ↓1    ↓1    ↓1
       (f1)  (f2)  (f3)  ...infinite accepting states
```

→ Pushdown Automata, with the use of stack, can capture all the states of this syntax



Basically:

- When see "a", push X on the stack
- When see "b", pop X out of stack

Grammar Flow Graph (GFG)

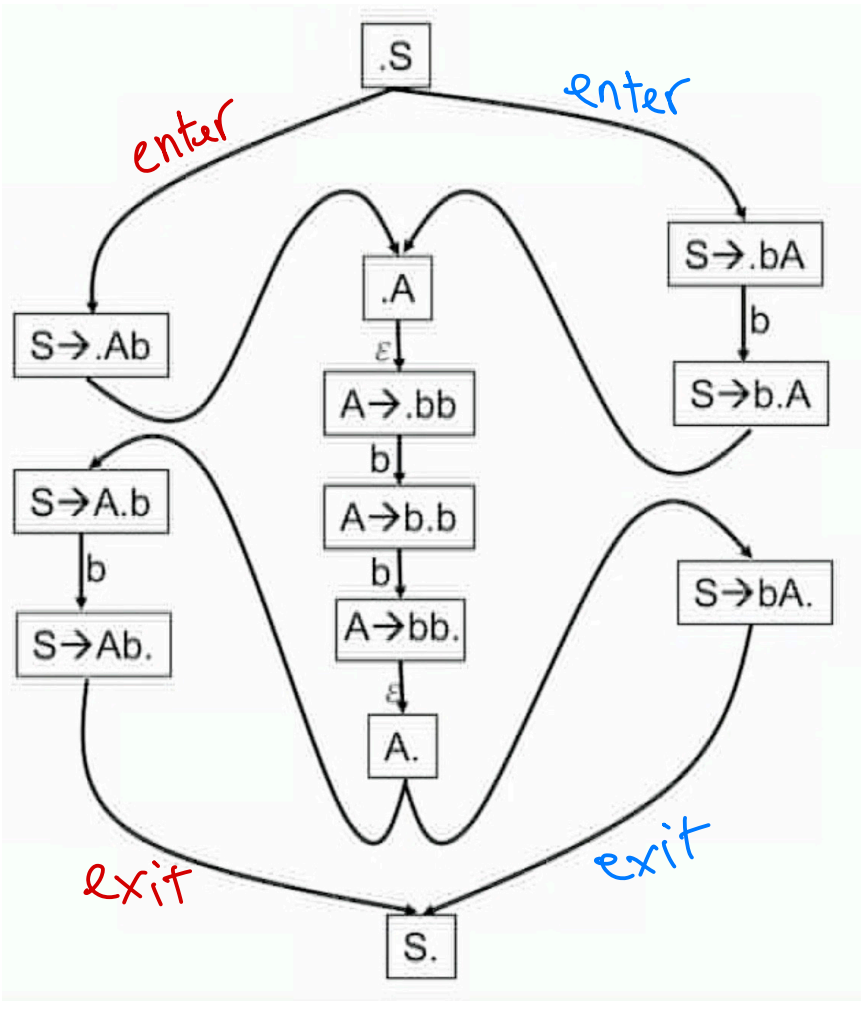
A graphical way to represent Pushdown Automata

Example: Given this grammar

$$S \rightarrow Ab \mid bA$$

$$A \rightarrow bb$$

The GFG would look something like this



Note:

- The "enter" and "exit" path need to match color. And the way to enforce it is to use a stack