Chomsky Normal Form (C.N.F.)

Def A c.f. grammar is in Chomsky normal form if every rule is of the form:

- $S \to \varepsilon$ (S is the start variable)
- $A \to a$ (A is a variable, a is a terminal)
- $A \to BC$ (A, B, C are variables, B, C are not start variable).

We note the following theorem.

Theorem Every c.f. language is generated by a c.f. grammar in the Chomsky normal form.

Having the grammar in a "normal" (i.e. standardized) form often makes reasoning about the grammar more convenient.

E.g. If the grammar is in C.N.F., then a string of terminals of length n is generated in about $2n$ steps : n applications of the rule of the type $A \to BC$ and then n applications of the rule of the type $A \to a$, converting all variables to terminals.

We will prove the theorem by giving a procedure that takes a c.f. grammar and converts it into an equivalent grammar in C.N.F. We skip the formal proof and only illustrate through an example

Suppose the given grammar is :

$$
G: \quad \begin{aligned} S &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \varepsilon \end{aligned}
$$

The rules of type $B \to \varepsilon$ ($B \neq$ start var.) are called ε-rules and those of type $A \to B$ are called unit rules. We need to remove them.

We begin by introducing a new start variable S_0 and adding the rule $S_0 \to S$.

Step 1 : Add a new start variable

$$S_0 \to S$$
$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

Step 2 : Remove ε-rules

To remove a rule $B \to \varepsilon$, we delete it and replace every rule

$$A \to uBv$$

by $A \to uBv \mid uv$.

Thus removing the rule $B \to \varepsilon$ from above gives

$$S_0 \to S$$
$$S \to ASA \mid aB \mid a$$
$$A \to B \mid \varepsilon \mid S$$
$$B \to b$$

Now removing $A \to \varepsilon$ gives

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow aB \mid a \mid ASA \mid SA \mid AS \mid \cancel{S} \\
A &\rightarrow B \mid S \\
B &\rightarrow b.
\end{aligned}
$$

Whenever we have a rule $S \rightarrow S$, S is a variable, it is removed (safely).

Step 3 : Remove unit rules

To remove a rule $A \rightarrow B$, delete it and for every rule $B \rightarrow u$, add the rule $A \rightarrow u$.

Thus removing the rule $A \rightarrow B$ gives

$$
\begin{aligned}
S_0 &\rightarrow S \\
S &\rightarrow aB \mid a \mid ASA \mid SA \mid AS \\
A &\rightarrow b \mid S \\
B &\rightarrow b.
\end{aligned}
$$

Removing the rule $A \rightarrow S$ gives

$S_0 \to S$

$S \to aB \mid a \mid ASA \mid SA \mid AS$

$A \to b \mid aB \mid a \mid ASA \mid SA \mid AS$

$B \to b$.

Removing the rule $S_0 \to S$ gives

$S_0 \to aB \mid a \mid ASA \mid SA \mid AS$

$S \to aB \mid a \mid ASA \mid SA \mid AS$

$A \to b \mid aB \mid a \mid ASA \mid SA \mid AS$

$B \to b$

Step 4: Breaking rules and Dummy variables.

We can break a rule such as

$S \to ASA$

into two rules

$S \to A_1 A$

$A_1 \to AS$.

Also a rule such as $S \to aB$ can be replaced by

$S \to X_a B$

$X_a \to a$

where X_a is

a dummy variable that stands for the terminal $\underline{a}$. This gives

$$S_0 \rightarrow X_a B \mid a \mid A_1 A \mid SA \mid AS$$

$$A_1 \rightarrow AS$$

$$X_a \rightarrow a$$

$$S \rightarrow X_a B \mid a \mid A_1 A \mid SA \mid AS$$

$$A \rightarrow b \mid X_a B \mid a \mid A_1 A \mid SA \mid AS$$

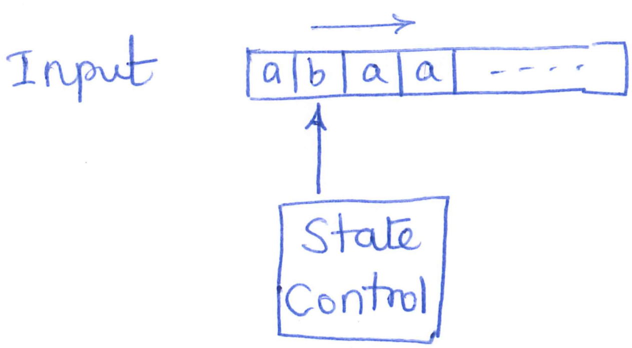
$$B \rightarrow b$$

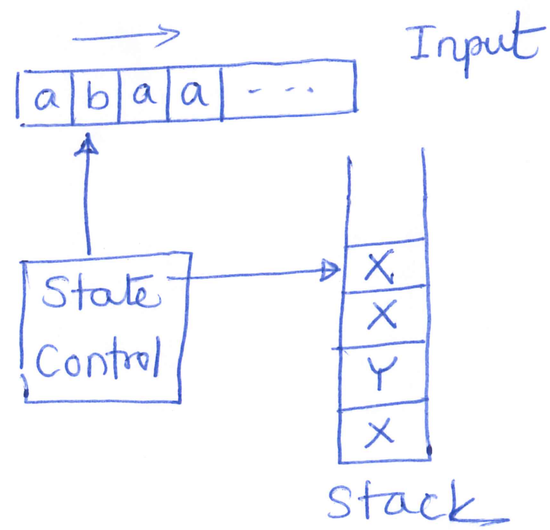
The grammar is now in Chomsky normal form.

Push-Down Automata (PDA)

We now study PDAs, a computational model that characterizes c.f. languages.

	Regular Languages	Contex Free lang.
Syntactic Characterization.	Regular expressions	Contex free grammars.
computational Characterization.	Finite Automata	Push-Down Automata.





- Read input symbol
- Enter (possibly) new state
- Move input pointer to the right (so input is 1-way, read-only).

- Read input symbol and top stack symbol
- Enter (possibly) new state.
- Replace or push or pop top stack symbol.
- Move input pointer to the right.

In both FA and PDA, accept if the input is exhausted and the m/c is in an accept state.

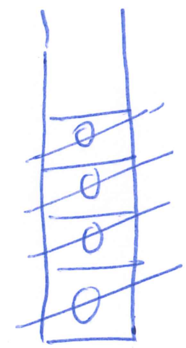
recognized

Example $L = \{ 0^n 1^n \mid n \geq 1 \}$ is accepted by a PDA.

Input. 00001111.

- Keep pushing 0's onto stack.
- After reading 1, pop a 0 from the stack and keep popping a 0 for every 1 read from the input.
- When the input is exhausted, accept if the stack is empty.

Stack.



Non-Deterministic PDA

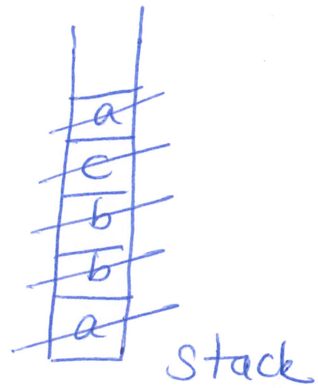
Example $L = \{ W \cdot W^{Reverse} \mid W \in \{a,b,c\}^* \}$

is recognized by a non-deterministic PDA.

The proposed PDA should

- push w onto the stack.
- After the string $w^{Reverse}$ "begins", pop the stack, matching the top stack symbol with the input symbol, in every step.

Input abbca; acbba



However, how does the PDA "know" when w ends and $w^{Reverse}$ begins?

Answer: Non-deterministically "guess" the midpoint of the string $ww^{Reverse}$.

The non-det PDA is then:

- Keep pusing input symbols onto the stack.
- Non-deterministically enter a new state.
- Keep matching input symbols to top stack symbols and popping. ▨

* Henceforth, PDA always means non-deterministic PDA. *

Formal Definition of a PDA

A PDA is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_1, F)$

where

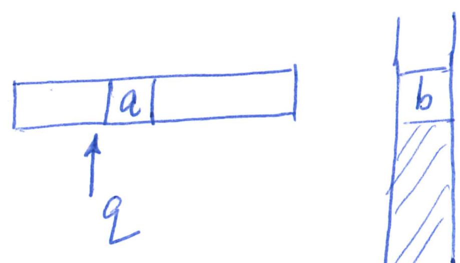
- Q is a finite set of states.
- Σ " input symbols.
- Γ " stack symbols.
- q_1 is the start state.
- $F \subseteq Q$ is the subset of accept states.
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function where

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}, \qquad \Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}.$$

The interpretation is that if

$\delta(q, a, b)$ contains (q', c) then

In state q, input a,
top of stack b,



the PDA changes state to q', replaces top stack symbol b with c.

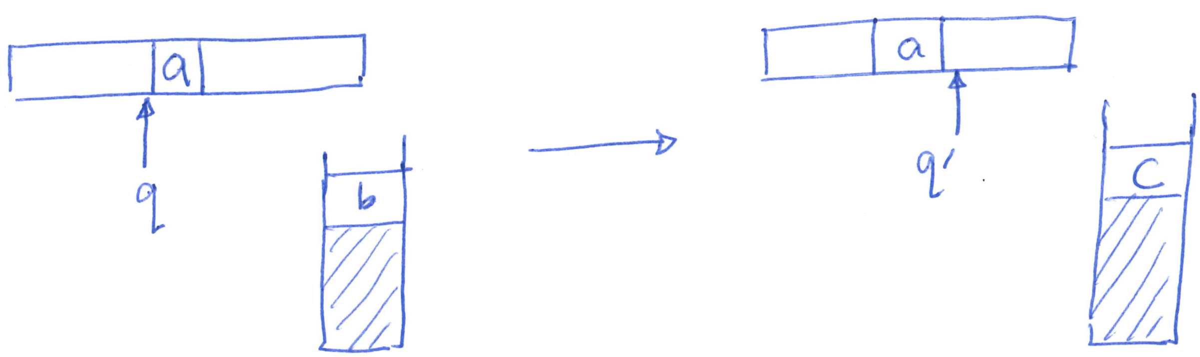
Note Formally $\delta(q, a, b)$ is a set of possible moves, since the PDA is non-deterministic. It may be that $\delta(q,a,b) = \phi$.

Any of a, b, c can be ε as allowed by the definition of the transition function. This gives rise to 8 possible types of moves, 4 by reading an input symbol and 4 without " (i.e. ε-move).

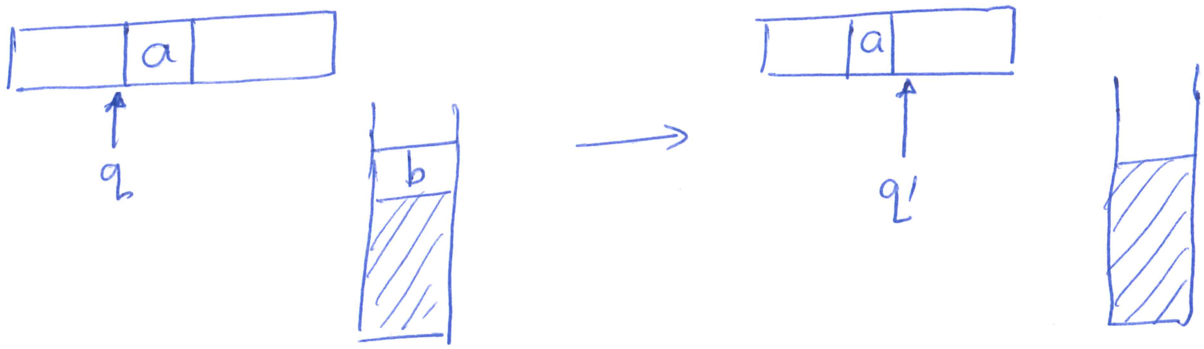
We will note down all these moves below. Below, a, b, c will denote input symbols and ε will be written explicitly.

4- Moves by Reading an Input Symbol

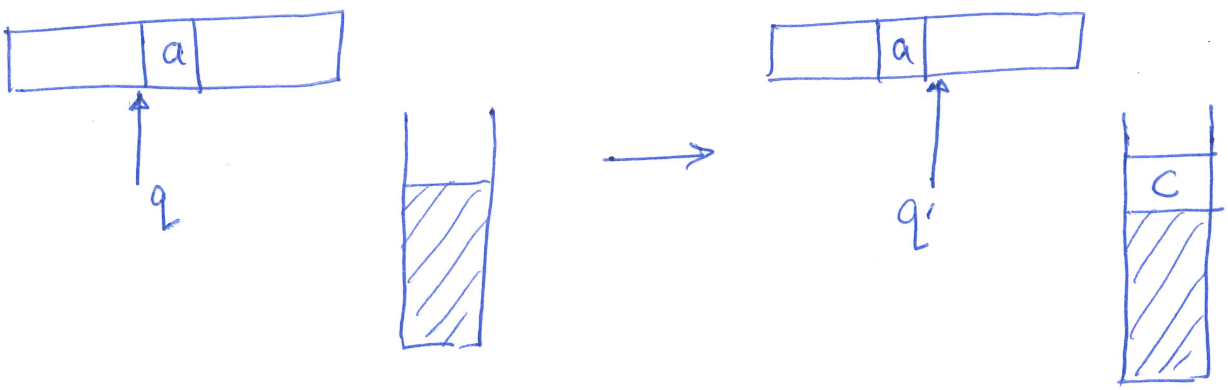
Replace $\delta(q, a, b)$ contains (q', c).



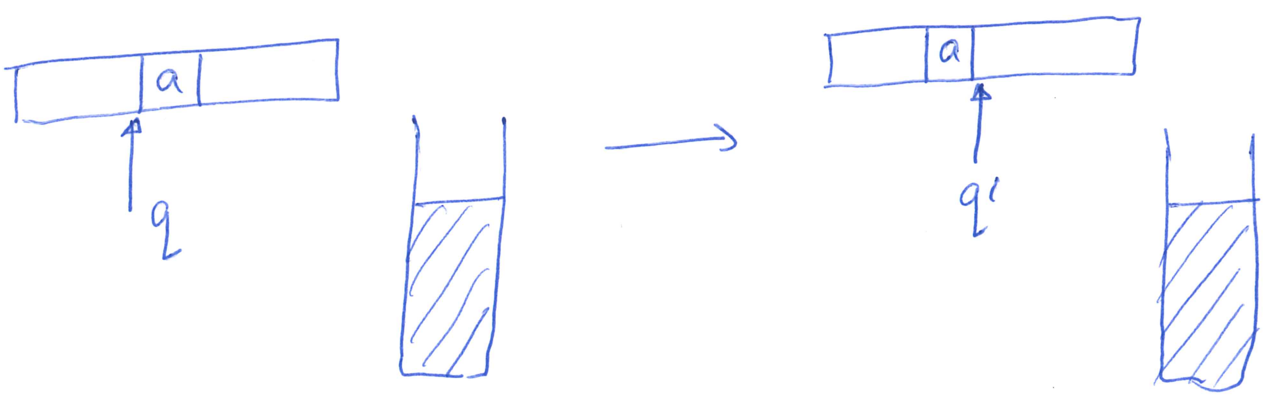
Pop $\delta(q, a, b)$ contains (q', ε).



Push $\delta(q, a, \varepsilon)$ contains (q', c)

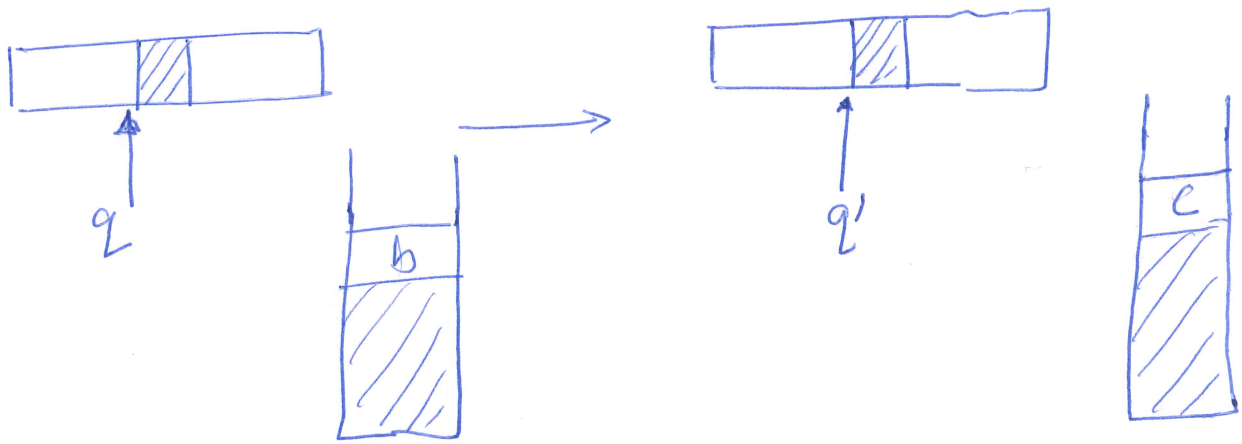


Stack Unchanged / Untouched $\delta(q, a, \varepsilon)$ contains (q', ε).

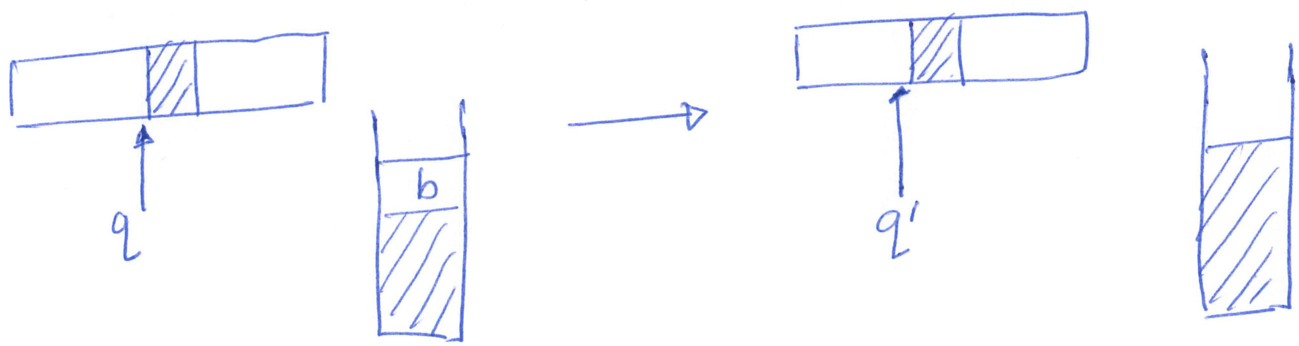


4 Moves Without Reading an Input Symbol

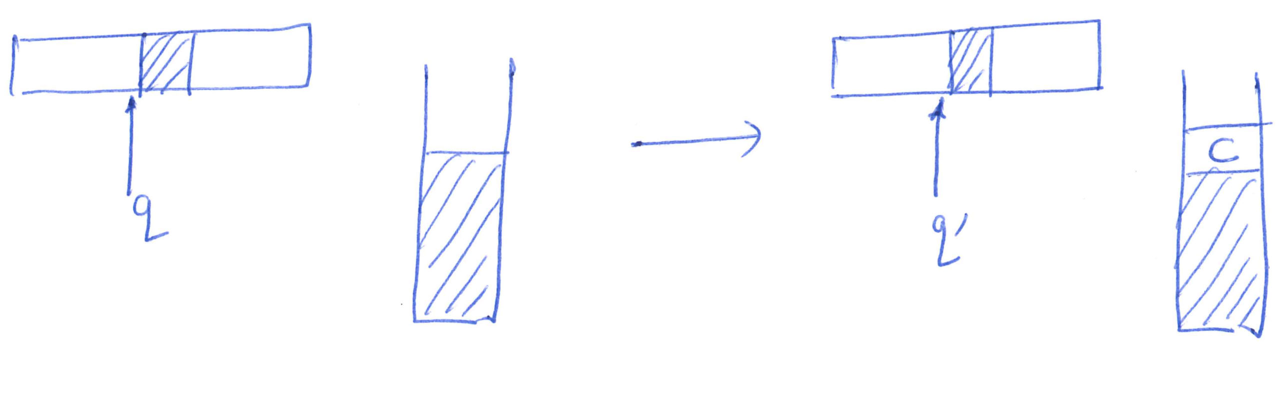
Replace $\delta(q, \varepsilon, b)$ contains (q', c).



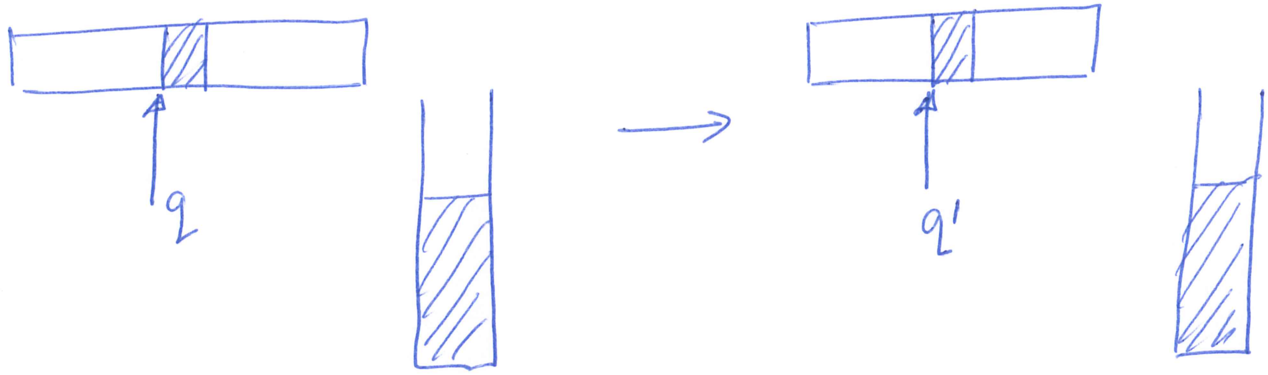
Pop $\delta(q, \varepsilon, b)$ contains (q', ε).



Push $\delta(q, \varepsilon, \varepsilon)$ contains (q', c).



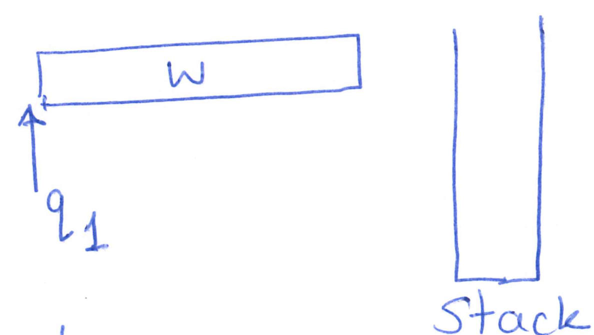
Untouched $\delta(q, \varepsilon, \varepsilon)$ contains (q', ε).



Computation of a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_1, F)$ on input $w \in \Sigma^*$:

- Starts in the initial configuration



  where State $= q_1$, Stack is empty.

- Makes moves according to transition f^n δ.
- Accepts if
  - input is exhausted and
  - the state is an accept state (i.e. in F).

Keeping in mind that the PDA is non-deterministic, the language recognized by the PDA M is:

$$L(M) = \{ w \in \Sigma^* \mid \text{There } \underline{\text{exists}} \text{ a computation of } M \text{ on } w \text{ that accepts.} \}$$

<u>Clarification</u> - The language $L(M)$ is defined only over the input alphabet Σ.

- Formally, input alphabet Σ and the stack alphabet Γ are disjoint. However one can argue as if $\Sigma \subseteq \Gamma$ since for every $a \in \Sigma$, one can have corresponding symbol $X_a \in \Gamma$ that stands for $a \in \Sigma$.

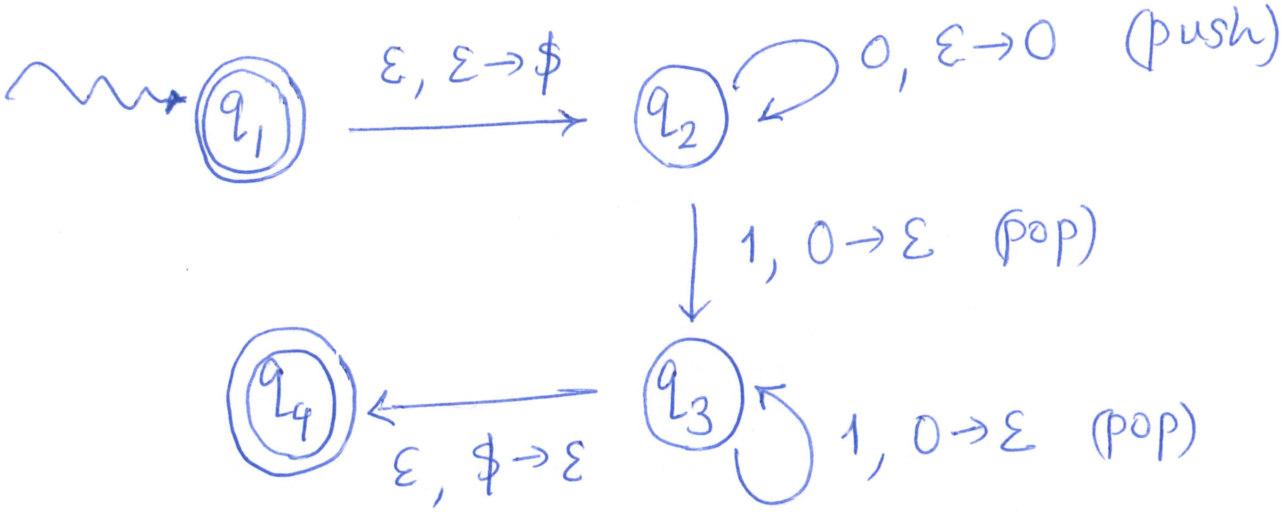
PDAs are represented by state diagrams.

$$q \xrightarrow{a,\, b \to c} q'$$

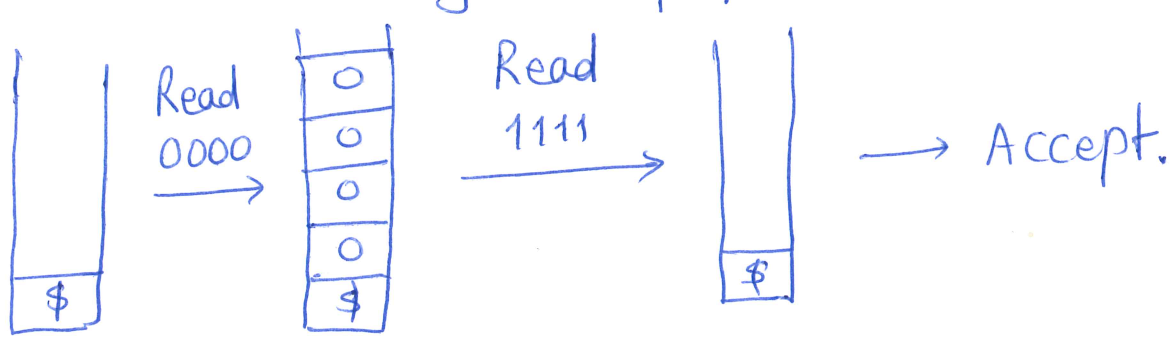
represents the move $\delta(q, a, b)$ contains (q', c).

Example State diagram for the PDA that recognizes the language

$$L = \{ 0^n 1^n \mid n \geq 0 \}.$$

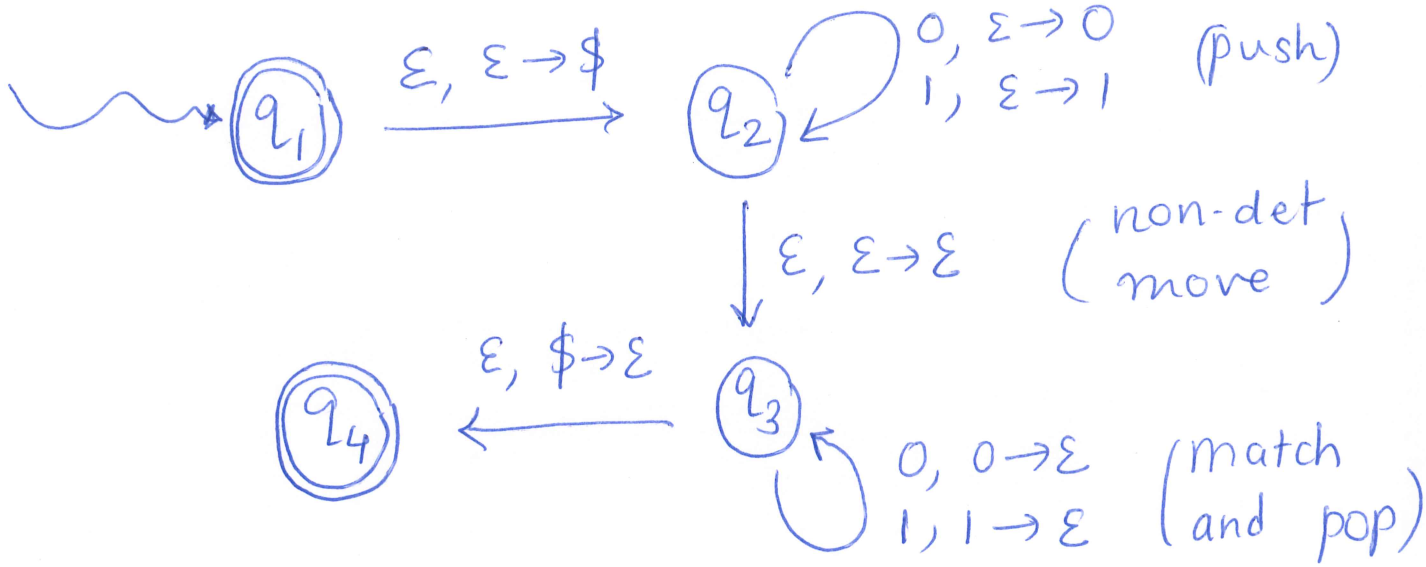


There is no formal mechanism to test whether the stack is empty, so the PDA begins by pushing $ symbol onto the stack. Later, whenever the top stack symbol is $, it effectively amounts to the stack being empty. On input 00001111



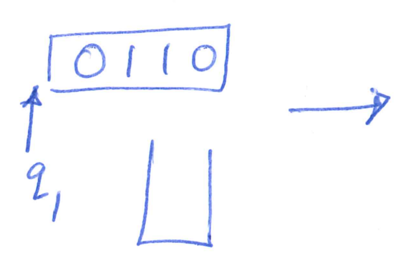
Example State diagram for the PDA that recognizes the language

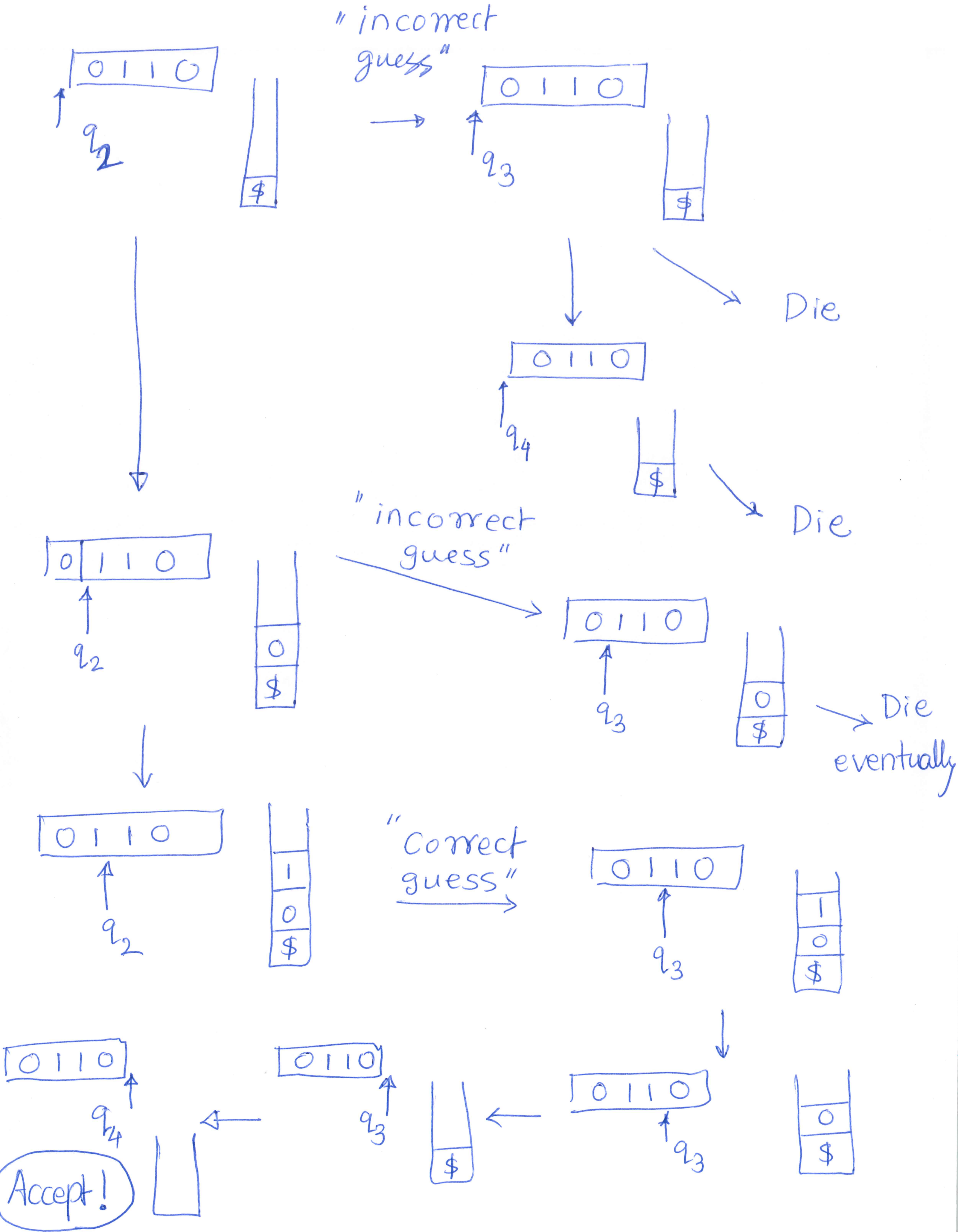
$$L = \{ w \cdot w^{Reverse} \mid w \in \{0,1\}^* \}.$$



The move $q_2 \xrightarrow{\varepsilon,\ \varepsilon \to \varepsilon} q_3$ is a non-deterministic move where the PDA "believes" that the midpoint of the string $w\, w^{Reverse}$ has been reached.

On input 0110, several possible computations are possible as: (depending on "guess" of the midpoint)




"incorrect guess"
0110
q_2
$
q_3
Die
q_4
Die
"incorrect guess"
0
Die eventually
"Correct guess"
1
Accept!