Non-deterministic pushdown automata

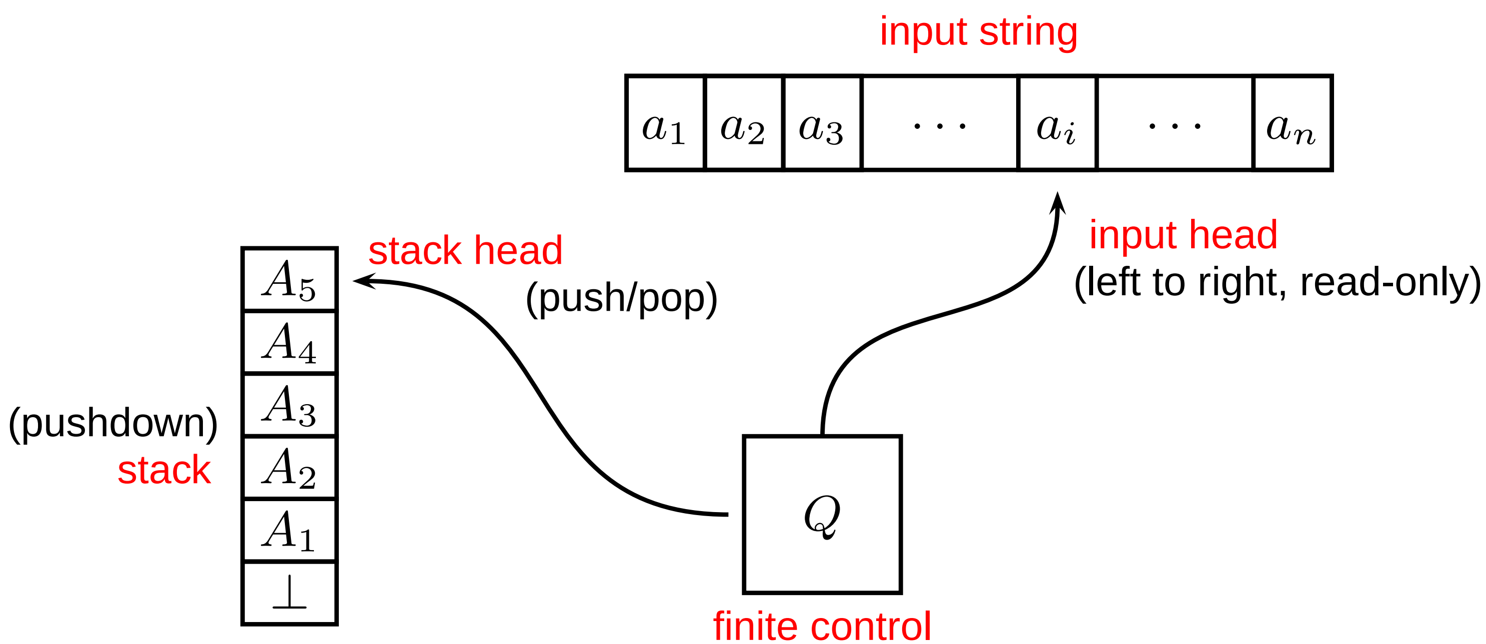
Regular languages are recognized by finite automata,

context free languages are recognized by non-deterministic pushdown automata.

Non-deterministic pushdown automata

(We mostly follow the definitions in Kozen's *Automata and Computability*.)

A (non-deterministic) pushdown automaton is like an NFA, except it has a *stack* (pushdown store) for recording a potentially unbounded amount of information, in a last-in-first-out (LIFO) fashion.



The workings of an NPDA

In each step, the NPDA pops the top symbol off the stack; based on (1) this symbol, (2) the input symbol currently reading, and (3) its current state, it can

1. push a sequence of symbols (possibly ϵ) onto the stack
2. move its read head one cell to the right, and
3. enter a new state

according to the transition rule δ of the machine.

We allow *ϵ-transition*: an NPDA can pop and push without reading the next input symbol or moving its read head.

Note: an NPDA can only access the top of stack symbol in each step.

Example: Balanced strings of parentheses

Intuitive description of an NPDA:

1. IF ¡input symbol is “[”¿

   THEN ¡push “[” onto the stack¿.

2. IF ¡input symbol is “]”¿ AND

   ¡top of stack is “[”¿ THEN ¡pop¿.

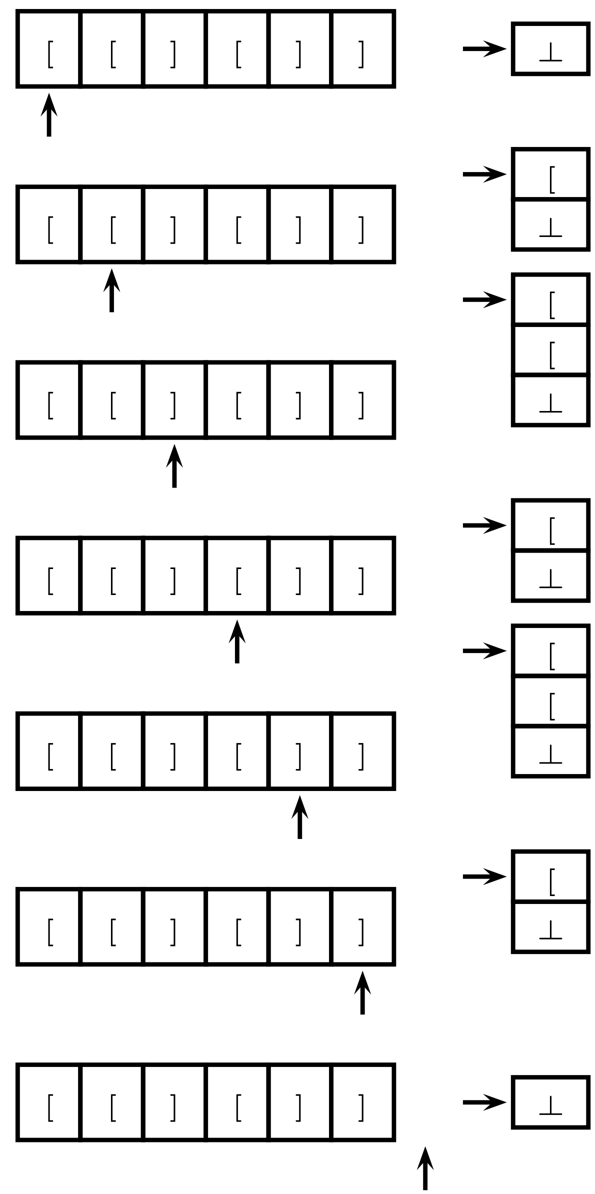
3. IF ¡all of input read¿ AND

   ¡top of stack is “⊥”¿ THEN ¡accept¿.

   (“⊥” is initial stack symbol.)

Example: input is “[[ ] [ ]]”

Think of an NPDA as (representing) an algorithm (for a decision problem) with memory access in the form of a stack.



Definition of an NPDA

A *non-deterministic pushdown automaton* (NPDA) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ where $Q, \Sigma, \Gamma, \delta$ and F are all finite sets, and

- Q is the set of states
- Σ is the *input alphabet*
- Γ is the *stack alphabet*
- $\delta : Q \times (\Sigma \cup \{ \epsilon \}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is the transition function
- $q_0 \in Q$ is the start state
- $\perp \in \Gamma$ is the initial stack symbol
- $F \subseteq Q$ is the set of accept states.

Note: An NPDA is strictly more powerful than a *deterministic* PDA. We shall not consider the latter specifically here.

Configuration

A configuration of M is an element of $Q \times \Sigma^* \times \Gamma^*$ describing (1) the current state, (2) the portion of the input yet unread (i.e. under and to the right of the input head) and (3) the current stack contents.

The *start configuration* is $(q_0, w, \perp)$. I.e. M always starts in the start state with its input head scanning the leftmost input symbol and the stack containing only $\perp$.

The *next-configuration relation* $\longrightarrow$ describes how M moves from one configuration to another in one step. Formally

- If $(q, \gamma) \in \delta(p, a, A)$ then for any $v \in \Sigma^*$ and $\beta \in \Gamma^*$,
  $$(p, av, A\beta) \;\longrightarrow\; (q, v, \gamma\beta)$$
  (The input symbol a has been "consumed"; A was popped and γ was pushed, and the new state is q.)
- If $(q, \gamma) \in \delta(p, \epsilon, A)$ then for any $v \in \Sigma^*$ and $\beta \in \Gamma^*$,
  $$(p, v, A\beta) \;\longrightarrow\; (q, v, \gamma\beta)$$
  (no input symbol has been "consumed".)

$L(M)$: The language accepted by NPDA M

We define the reflexive, transitive closure of $\longrightarrow$, written $\xrightarrow{*}$, as follows:

$$
\begin{aligned}
C \xrightarrow{0} D &\iff C = D \\
C \xrightarrow{n+1} D &\iff \exists E \,.\, C \xrightarrow{n} E \;\wedge\; E \longrightarrow D
\end{aligned}
$$

and define $C \xrightarrow{*} D$ just if $C \xrightarrow{n} D$ for some $n \geq 0$. I.e. $C \xrightarrow{*} D$ iff D follows from C in 0 or more steps of the relation $\longrightarrow$.

Formally we say that M *accepts an input x by final state* if for some $q \in F$ and $\gamma \in \Gamma^*$, we have $(q_0, x, \perp) \;\xrightarrow{*}\; (q, \epsilon, \gamma)$. Configurations of the form (q, ϵ, γ) where $q \in F$ and $\gamma \in \Gamma^*$ are called *accepting*.

The *language* of M, written $L(M)$, is defined to be the set of strings accepted by M.

There is another accepting convention:

M *accepts an input x by empty stack* if for some $q \in Q$,

$$(q_0, x, \perp) \xrightarrow{*} (q, \epsilon, \epsilon).$$

N.B. F is irrelevant in the definition of acceptance by empty stack.

The two accepting conventions are equivalent.

Example: An NPDA accepting $\{ ww^R : w \in \{ 0, 1 \}^* \}$

High-level description:

1. Push the input symbols onto the stack, one at a time.
2. Non-deterministically guess that the middle of the string has been reached at some point during 1, and then change into popping off the stack for each symbol read, checking to see if they (i.e. symbols just popped and just read) are the same.
3. If they are always the same symbols, and the stack empties at the same time as the input is finished, accept.

Example: An NPDA accepting $\{ ww^R : w \in \{ 0, 1 \}^* \}$

Implementation-level description:

$$(\{ q_1, q_0, q_2 \}, \underbrace{\{ 0, 1 \}}_{\Sigma}, \underbrace{\{ 0, 1, \perp \}}_{\Gamma}, \delta, q_0, \perp, \{ q_2 \})$$

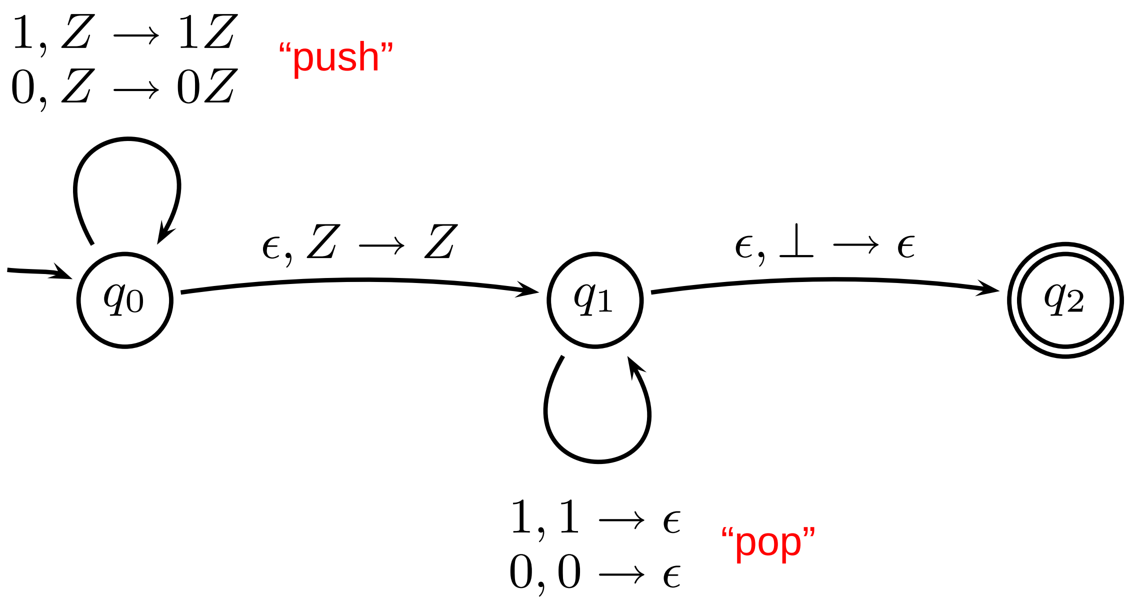
where

$$\delta : \begin{cases} (q_0, 0, Z) & \mapsto & \{ (q_0, 0Z) \} \\ (q_0, 1, Z) & \mapsto & \{ (q_0, 1Z) \} \\ (q_0, \epsilon, Z) & \mapsto & \{ (q_1, Z) \} \\ (q_1, 0, 0) & \mapsto & \{ (q_1, \epsilon) \} \\ (q_1, 1, 1) & \mapsto & \{ (q_1, \epsilon) \} \\ (q_1, \epsilon, \perp) & \mapsto & \{ (q_2, \epsilon) \} \end{cases}$$

where $Z = 0, 1$.

Example: An NPDA accepting $\{ ww^R : w \in \{0,1\}^* \}$

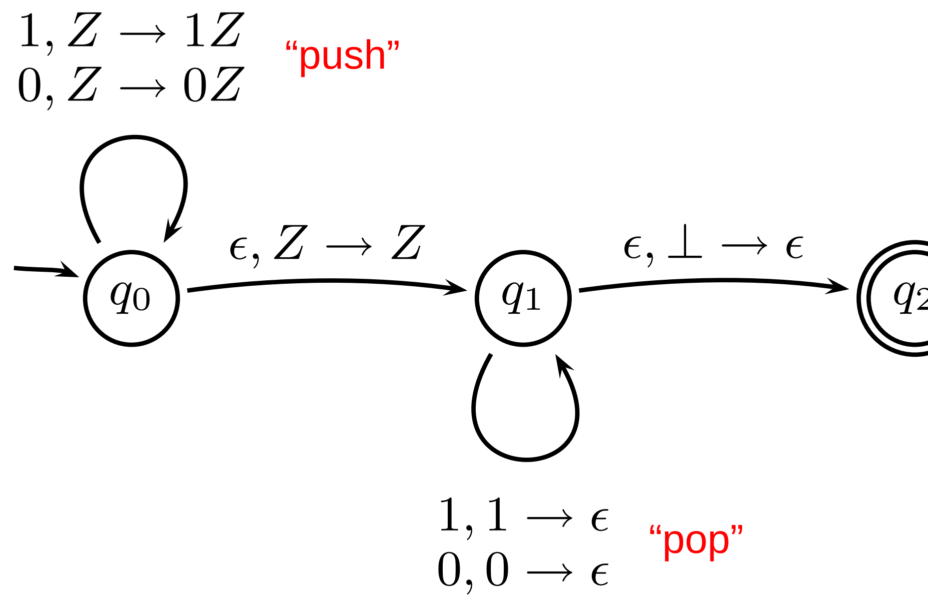
Transition graph:



Notation: In the transition graph, we represent the transition $(q', \gamma) \in \delta(q, a, Z)$ by an edge, labelled by "$a, Z \to \gamma$", that joins node q to q'.

Example: A run accepting the input 011110

$$
\begin{array}{rllr}
 & (q_0, & 011110, & \perp) \\
\longrightarrow & (q_0, & 11110, & 0\perp) \\
\longrightarrow & (q_0, & 1110, & 10\perp) \\
\longrightarrow & (q_0, & 110, & 110\perp) \\
\longrightarrow & (q_1, & 110, & 110\perp) \\
\longrightarrow & (q_1, & 10, & 10\perp) \\
\longrightarrow & (q_1, & 0, & 0\perp) \\
\longrightarrow & (q_1, & \epsilon, & \perp) \\
\longrightarrow & (q_2, & \epsilon, & \epsilon)
\end{array}
$$



Example: An NPDA accepting balanced strings of parentheses

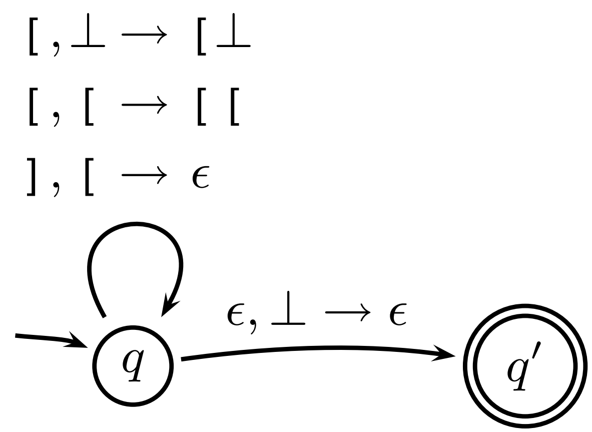
Implementation-level description:

$$(\{\, q, q' \,\}, \{\, [,] \,\}, \{\, \perp, [\,\}, \delta, q, \perp, \{\, q' \,\})$$

where

$$\delta \begin{cases} (q,\ [\ ,\perp) & \mapsto & \{\, (q,\ [\perp) \,\} \\ (q,\ [\ ,\ [\ ) & \mapsto & \{\, (q,\ [\ [\ ) \,\} \\ (q,\ ]\ ,\ [\ ) & \mapsto & \{\, (q,\ \epsilon) \,\} \\ (q,\ \epsilon\ ,\perp) & \mapsto & \{\, (q', \epsilon) \,\} \end{cases}$$

Transition diagram:



Equivalence between NPDAs and context-free languages

A major result in automata theory is:

Theorem. A language is context-free iff some NPDA accepts it.

Proof overview We are breaking down the proof into the following steps:

A Given a CFG G, there is an equivalent NPDA P_G.

B Given an NPDA N, there is an equivalent CFG G_N generating $L(N)$.

  1 Every NPDA can be simulated by an NPDA with one state

  2 Every NPDA with one state has an equivalent CFG.

Lemma. Given a CFG G, there is an equivalent NPDA P_G.

Proof idea: The stack alphabet of P_G consists of the terminal and variable symbols and $\perp$. We describe the action of P_G informally:

1. Place the start variable symbol on the stack.

2. Repeat forever: Pop top-of-stack x. Cases of:

   (a) x is a variable A: Nondeterministically select a rule for A and replace A by the string w (say) on the rhs of the rule (so that the leftmost symbol of w is at the top of stack).

   (b) x is a terminal a: Read the next input symbol and compare it with a. If they do not match, then exit (and reject this branch of the nondeterminism).

   (c) $x = \perp$: Enter the accept state.

Claim: P_G accepts $L(G)$.

NPDA that accepts CFL generated by $S \longrightarrow a\,S\,b\,S \mid b\,S\,a\,S \mid \epsilon$

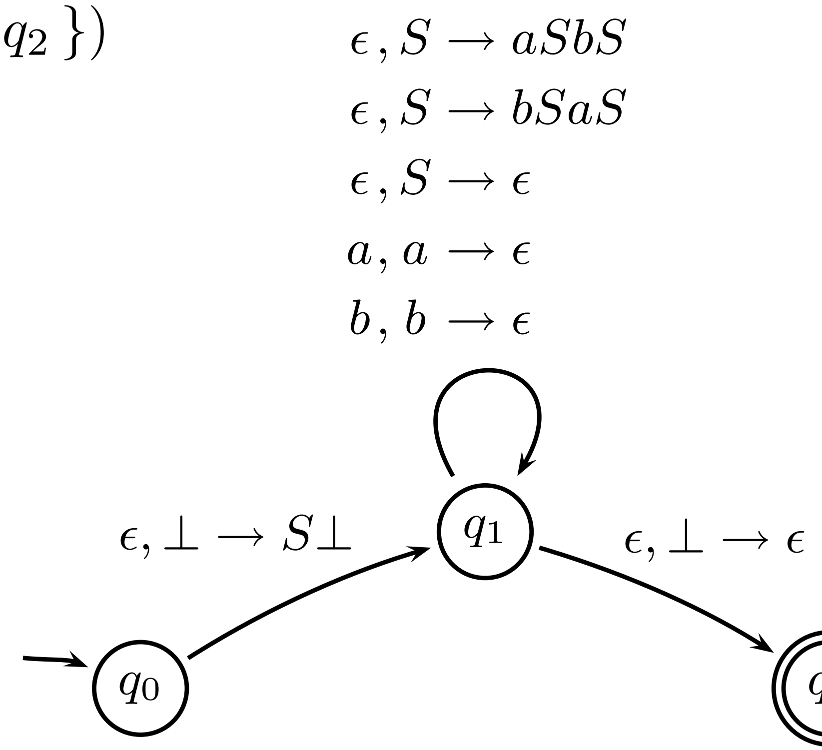
Implementation-level description:

$(\{\, q_0, q_1, q_2 \,\}, \{\, a, b \,\}, \{\, \perp, S, a, b \,\}, \delta, q_0, \perp, \{\, q_2 \,\})$

where δ is given by

$$
\begin{array}{rcl}
\delta(q_0, \epsilon, \perp) & = & \{(q_1, S\perp)\} \\
\delta(q_1, \epsilon, S) & = & \{(q_1, aSbS), \\
 & & \ (q_1, bSaS), \\
 & & \ (q_1, \epsilon)\} \\
\delta(q_1, a, a) & = & \{(q_1, \epsilon)\} \\
\delta(q_1, b, b) & = & \{(q_1, \epsilon)\} \\
\delta(q_1, \epsilon, \perp) & = & \{(q_2, \epsilon)\}
\end{array}
$$

Transition diagram:



Example: A run accepting the input $abab$

$$
\begin{array}{llrrl}
 & (q_0, & abab, & \perp) & \\
\longrightarrow & (q_1, & abab, & S\perp) & \\
\longrightarrow & (q_1, & abab, & aSbS\perp) & \text{(1)} \\
\longrightarrow & (q_1, & bab, & SbS\perp) & \\
\longrightarrow & (q_1, & bab, & bSaSbS\perp) & \text{(2)} \\
\longrightarrow & (q_1, & ab, & SaSbS\perp) & \\
\longrightarrow & (q_1, & ab, & aSbS\perp) & \text{(3)} \\
\longrightarrow & (q_1, & b, & SbS\perp) & \\
\longrightarrow & (q_1, & b, & bS\perp) & \text{(3)} \\
\longrightarrow & (q_1, & \epsilon, & S\perp) & \\
\longrightarrow & (q_1, & \epsilon, & \perp) & \text{(3)} \\
\longrightarrow & (q_2, & \epsilon, & \epsilon) &
\end{array}
$$

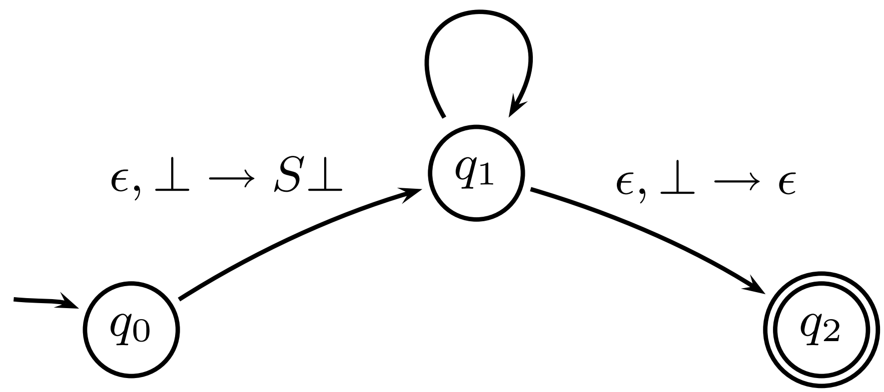
(1) $\epsilon\,, S \longrightarrow aSbS$

(2) $\epsilon\,, S \longrightarrow bSaS$

(3) $\epsilon\,, S \longrightarrow \epsilon$

$a\,, a \longrightarrow \epsilon$

$b\,, b \longrightarrow \epsilon$



Leftmost derivation: $S \rightarrow aSbS \rightarrow abSaSbS \rightarrow abaSbS \rightarrow ababS \rightarrow abab$

Simulating NPDAs by CFGs

We do this in two steps:

1. Every NPDA can be simulated by an NPDA with one state
2. Every NPDA with one state has an equivalent CFG.

For 2: Take a one-state NPDA $M = (\{q\}, \Sigma, \Gamma, \delta, q, \perp, \emptyset)$ that accepts by empty stack. Define

$$G_M \;=\; (\Gamma, \Sigma, P, \perp)$$

where P contains a rule

$$A \;\rightarrow\; c\, B_1 \;\cdots\; B_k$$

for every transition $(q, B_1 \;\cdots\; B_k) \in \delta(q, c, A)$ where $c \in \Sigma \cup \{\epsilon\}$. Then we have $L(M) = L(G_M)$.

Every NPDA can be simulated by a one-state NPDA

Idea: maintain all state information on the stack. W.l.o.g. we assume M is of the form $(Q, \Sigma, \Gamma, \delta, s, \perp, \{ t \})$, and M can empty its stack after entering final state t.

Set $\Gamma' = Q \times \Gamma \times Q$ (elements are written $\langle p\, A\, q \rangle$). We construct a new NPDA

$$M' \ = \ (\{ * \}, \Sigma, \Gamma', \delta', *, \langle s \perp t \rangle, \emptyset)$$

that accepts by empty stack. For each transition

$$(q_0, B_1 \ \cdots \ B_k) \in \delta(p, c, A)$$

where $c \in \Sigma \cup \{ \epsilon \}$, include in δ' the transition

$$(*, \langle q_0\, B_1\, q_1 \rangle \langle q_1\, B_2\, q_2 \rangle \ \cdots \langle q_{k-1}\, B_k\, q_k \rangle) \in \delta'(*, c, \langle p\, A\, q_k \rangle)$$

for all possible choices of $q_1, \cdots, q_k$.

Intuitively M' simulates M, guessing non-deterministically what state M will be in at certain future points in the computation, saving those guesses on the stack, and then verifying later that the guesses were correct.

Lemma. M' can scan a string x starting with only $\langle p\, A\, q \rangle$ on its stack and end up with an empty stack, if and only if M can scan x starting in stack p with only A on its stack and end up in state q with an empty stack. I.e. we have

$$(p, x, A) \xrightarrow{n}_M (q, \epsilon, \epsilon) \qquad \Longleftrightarrow \qquad (*, x, \langle p\, A\, q \rangle) \xrightarrow{n}_{M'} (*, \epsilon, \epsilon)$$

It then follows that $L(M) = L(M')$.

Regular Operations and Context-Free Languages

Theorem: Context-free languages are closed under the regular operations: union, concatenation and star.

Proof idea. Let $G_1 = (\Gamma_1, \Sigma, \mathcal{R}_1, S_1)$ and $G_2 = (\Gamma_2, \Sigma, \mathcal{R}_2, S_2)$ be context free grammars with $\Gamma_1 \cap \Gamma_2 = \emptyset$. Consider the following context free grammars

$$\begin{aligned}
G_{\text{union}} &= (\Gamma_1 \cup \Gamma_2 \cup \{ S \}, \Sigma, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{ S \to S_1, S \to S_2 \}, S) \\
G_{\text{concat}} &= (\Gamma_1 \cup \Gamma_2 \cup \{ S \}, \Sigma, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{ S \to S_1\, S_2 \}, S) \\
G_{\text{star}} &= (\Gamma_1 \cup \{ S \}, \Sigma, \mathcal{R}_1 \cup \{ S \to S_1\, S, S \to \epsilon \}, S)
\end{aligned}$$

where S is a fresh variable. Then

$$
\begin{aligned}
L(G_{\text{union}}) &= L(G_1) \cup L(G_2) \\
L(G_{\text{concat}}) &= L(G_1) \cdot L(G_2) \\
L(G_{\text{star}}) &= L(G_1)^*
\end{aligned}
$$

Intersection and context-free languages

1. Context-free languages are not closed under intersection.
2. The intersection of a context-free language with a regular set is context-free.

Proof Idea

1. Both $A = \{0^n\,1^n\,0^m : n, m \geq 0\}$ and $B = \{0^n\,1^m\,0^m : n, m \geq 0\}$ are context-free. But $A \cap B = \{0^n\,1^n\,0^n : n \geq 0\}$ is not context-free.
2. Let A be a regular set accepted by NFA $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and let B be a context-free language accepted by NPDA $N = (Q_2, \Sigma, \Gamma, \delta_2, q_2, \perp, F_2)$.
   Define the ”product”-automaton

$P = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), \perp, F_1 \times F_2)$ where δ is given by

$$\delta((r,t), a, A) = \{\, ((r', t'), \alpha) : r' \in \delta_1(r, a), (t', \alpha) \in \delta_2(t, a, A) \,\}$$

then P is a NPDA that accepts $A \cap B$.

Complementation and context-free languages

> Context-free languages are not closed under complementation.

Proof Idea The set $A = \{\, w\,w : w \in \{0,1\}\,\}$ is not context-free, but its complement $B = \{0,1\}^* \setminus A$ is context-free, as it is generated by the grammar $(\{\,S, A, B, C\,\}, \{\,0, 1\,\}, \mathcal{R}, S)$

whith rule set
$$\mathcal{R} = \{\begin{array}{lcl} S & \rightarrow & AB \mid BA \mid A \mid B\,, \\ A & \rightarrow & CAC \mid 0\,, \\ B & \rightarrow & CBC \mid 1\,, \\ C & \rightarrow & 0 \mid 1 \;\} \end{array}$$

This grammar generates

all strings of odd length starting with productions $S \rightarrow A$ or $S \rightarrow B$ or

strings of the form $x0yu1v$ or $u1vx0y$ where $x, y, u, v \in \{0,1\}^*$, $|x| = |y|$ and $|u| = |v|$. None of these strings can be of the form $w\,w$.