2 Context-Free Languages

2.1 Context-Free Grammars

A *context-free grammar* (CFG) is a structure $G = (N, \Sigma, P, S)$ where

* N is a finite set, the *non-terminals*,
* Σ is a finite set disjoint from N, the *terminals*,
* $P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of *productions*,
* $S \in N$ is the *start symbol*.

Productions are denoted as follows.

* A production $(A, w) \in P$ is written $A \to w$.
* Several productions $A \to w_1, \ldots, A \to w_n$ are written $A \to w_1 \mid \ldots \mid w_n$.
* The right-hand side may be empty: an ε-production is written $A \to \varepsilon$.

Example: $G = (N, \Sigma, P, E)$ for arithmetic expressions

Example: $G' = (N, \Sigma, P, S)$

Consider a CFG $G = (N, \Sigma, P, S)$.

* Let $u, v, w \in (N \cup \Sigma)^*$ and $A \to w \in P$.
* Then uAv yields uwv in one step, by replacing A with w; this is denoted $uAv \Rightarrow_G^1 uwv$.
* Alternatively, uwv is *derivable* from uAv in one step.
* Each of u, v, w may be ε.
* A can be replaced with w irrespective of the context u, v in which A occurs.

Example:

Derivability is a relation on $(N \cup \Sigma)^*$.

* Let $x_i \in (N \cup \Sigma)^*$ for each $i \in \mathbb{N}$.
* x_n is derivable from x_0 in n steps if $x_i \Rightarrow_G^1 x_{i+1}$ for each $0 \leq i < n$; this is denoted $x_0 \Rightarrow_G^n x_n$.
* $x \Rightarrow_G^0 y$ if and only if $x = y$.
* y is derivable from x if it is derivable in any number of steps.
* Alternatively, x *generates* y or x yields y.
* $x \Rightarrow_G^* y$ if $x \Rightarrow_G^n y$ for some $n \in \mathbb{N}$.
* The relation $\Rightarrow_G^*$ is the *reflexive-transitive closure* of the relation $\Rightarrow_G^1$.

Example:

2.1.1 Context-Free Languages

A context-free grammar generates a context-free language (CFL).

* A *sentential form* is any $x \in (N \cup \Sigma)^*$ derivable from the start symbol S, that is, $S \Rightarrow_G^* x$.
* A *sentence* is a sentential form that consists only of terminal symbols: $x \in \Sigma^*$.
* $L(G) = \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$ is the *language generated* by G.
* $A \subseteq \Sigma^*$ is *context-free* if $A = L(G)$ for some CFG G.

$A = \{a^n b^n \mid n \in \mathbb{N}\}$ is context-free:

Further examples of CFLs are:

* $\{a^i b^j c^k \mid (i = j \text{ or } j = k) \text{ and } i, j, k \geq 1\}$ is generated by

$$\begin{array}{lcl} S & \rightarrow & TC \mid AU \\ T & \rightarrow & aTb \mid ab \\ U & \rightarrow & bUc \mid bc \\ A & \rightarrow & a \mid aA \\ C & \rightarrow & c \mid cC \end{array}$$

The language of balanced parentheses is generated by

A sentence may have several derivations.

* Consider the CFG induced by the production

$$E \quad \rightarrow \quad E + E \mid E * E \mid n$$

* Some derivations of the sentence $n + n * n$ are:

* $\Rightarrow$ is short for $\Rightarrow^1$, which is short for $\Rightarrow^1_G$ if G is understood.
* The first two are *leftmost derivations*, the next two are *rightmost* derivations.
* In each step of a leftmost derivation, the leftmost non-terminal is replaced.
* In each step of a rightmost derivation, the rightmost non-terminal is replaced.
* A sentence is *ambiguous* if it has more than one leftmost derivation.
* A CFG is *ambiguous* if it generates an ambiguous sentence.
* A CFL is *inherently ambiguous* if every CFG generating it is ambiguous.
* $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$ is inherently ambiguous.

The above CFG for expressions has the following rightmost derivation:

$$\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow E + T * F \\
&\Rightarrow E + T * n \\
&\Rightarrow E + F * n \\
&\Rightarrow E + n * n \\
&\Rightarrow T + n * n \\
&\Rightarrow F + n * n \\
&\Rightarrow n + n * n
\end{aligned}$$

2.1.2 Regular Grammars

A *regular grammar* is a CFG $G = (N, \Sigma, P, S)$ where for each $A \to w \in P$,

* $w = \varepsilon$, or
* $w \in \Sigma N$.
* The right-hand side of each production is either ε or a terminal followed by a non-terminal.

Example:

Every regular language is generated by a regular grammar. Proof:

* Consider the DFA $M = (Q, \Sigma, \delta, q_0, F)$.
* Construct the regular grammar $G = (Q, \Sigma, P, q_0)$ with the following productions P:

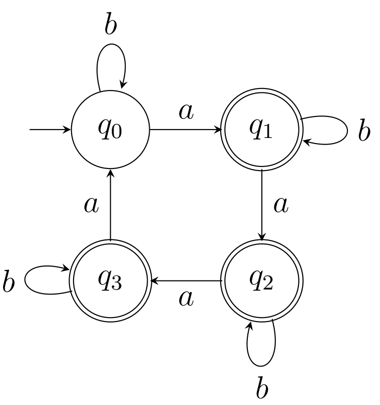
$$\begin{array}{lcll} q_i & \to & aq_j & \text{if } \delta(q_i, a) = q_j \\ q_i & \to & \varepsilon & \text{if } q_i \in F \end{array}$$

* Then for each $w \in \Sigma^*$,

$$\begin{array}{ll} & w = a_1 a_2 \ldots a_{n-1} a_n \in L(M) \\ \Leftrightarrow & \hat{\delta}(q_0, w) \in F \\ \Leftrightarrow & \delta(q_{i-1}, a_i) = q_i \text{ for each } 1 \leq i \leq n \text{ and } q_n \in F \\ \Leftrightarrow & q_{i-1} \to a_i q_i \in P \text{ for each } 1 \leq i \leq n \text{ and } q_n \to \varepsilon \in P \\ \Leftrightarrow & q_0 \Rightarrow a_1 q_1 \Rightarrow a_1 a_2 q_2 \Rightarrow \ldots \Rightarrow a_1 a_2 \ldots a_{n-1} a_n q_n \Rightarrow a_1 a_2 \ldots a_{n-1} a_n \\ \Leftrightarrow & w = a_1 a_2 \ldots a_{n-1} a_n \in L(G). \end{array}$$

* Hence $L(M) = L(G)$.

Example:



Every regular grammar generates a regular language. Proof:

* Consider the regular grammar $G = (N, \Sigma, P, S)$.
* Construct NFA $M = (N, \Sigma, \delta, S, F)$.
* The accept states are $F = \{V \mid (V \to \varepsilon) \in P\}$.
* The transitions are $\delta(V, a) = \{W \mid (V \to aW) \in P\}$.
* Then for each $w \in \Sigma^*$,

$$\begin{array}{ll} & w = a_1 a_2 \ldots a_{n-1} a_n \in L(G) \\ \Leftrightarrow & S = V_0 \Rightarrow a_1 V_1 \Rightarrow a_1 a_2 V_2 \Rightarrow \ldots \Rightarrow a_1 a_2 \ldots a_{n-1} a_n V_n \Rightarrow a_1 a_2 \ldots a_{n-1} a_n \\ \Leftrightarrow & V_{i-1} \to a_i V_i \in P \text{ for each } 1 \leq i \leq n \text{ and } V_n \to \varepsilon \in P \\ \Leftrightarrow & V_i \in \delta(V_{i-1}, a_i) \text{ for each } 1 \leq i \leq n \text{ and } V_n \in F \\ \Leftrightarrow & \hat{\delta}(S, w) \cap F \neq \emptyset \\ \Leftrightarrow & w = a_1 a_2 \ldots a_{n-1} a_n \in L(M). \end{array}$$

* Hence $L(G) = L(M)$.

Every regular language A has the following equivalent characterisations:

* A is accepted by a DFA.
* A is accepted by an NFA.
* A is accepted by an NFA with ε-transitions.
* A is generated by a regular expression.
* A is generated by a regular grammar.

It follows that:

Alternative characterisations of regular grammars require for each $A \to w \in P$:

* $w \in \Sigma N \cup \{\varepsilon\}$,
* $w \in N\Sigma \cup \{\varepsilon\}$,
* $w \in \Sigma N \cup \Sigma \cup \{\varepsilon\}$,
* $w \in N\Sigma \cup \Sigma \cup \{\varepsilon\}$,
* $w \in \Sigma N \cup N \cup \{\varepsilon\}$,
* $w \in N\Sigma \cup N \cup \{\varepsilon\}$,
* $w \in \Sigma^* N \cup \Sigma^*$, or
* $w \in N\Sigma^* \cup \Sigma^*$.

In general, the following restrictions do not yield regular languages:

* $w \in \Sigma N \cup N\Sigma \cup \{\varepsilon\}$,
* $w \in NN \cup \Sigma$.

2.1.3 Chomsky Normal Form

A CFG $G = (N, \Sigma, P, S)$ is in *Chomsky normal form* if every production has the form

$*$ $A \to BC$, where $B, C \in N$, or

$*$ $A \to a$, where $a \in \Sigma$.

$*$ The right-hand side of each production is either two non-terminals or a terminal.

For every CFG G with $\varepsilon \notin L(G)$ there is a CFG G' in Chomsky normal form with $L(G) = L(G')$.

1. Eliminate ε-productions of the form $A \to \varepsilon$.
2. Eliminate unit-productions of the form $A \to B$.
3. Eliminate non-generating non-terminals.
4. Eliminate non-reachable non-terminals.
5. Eliminate terminals from right-hand sides of length at least 2.
6. Eliminate right-hand sides of length at least 3.

1. If $A \to uBv \in P$ for $u, v \in (\Sigma \cup N)^*$ and $B \to \varepsilon \in P$, add $A \to uv$ to P.

$*$ Repeat this step while there are changes.

$*$ Afterwards, remove all ε-productions of the form $A \to \varepsilon$.

2. If $A \to B \in P$ and $B \to w \in P$ for $w \in (\Sigma \cup N)^*$, add $A \to w$ to P.

$*$ Repeat this step while there are changes.

$*$ Afterwards, remove all unit-productions of the form $A \to B$.

3. A non-terminal A is *generating* if $A \Rightarrow^* w$ for some $w \in \Sigma^*$.

$*$ If $A \to w \in P$ and w contains only terminals, A is generating.

$*$ If $A \to w \in P$ and w contains only terminals or generating non-terminals, A is generating.

$*$ Remove each non-generating non-terminal with all productions containing it.

4. A non-terminal A is *reachable* if $S \Rightarrow^* uAv$ for some $u, v \in (\Sigma \cup N)^*$.

$*$ S is reachable.

$*$ If $A \to w \in P$ and A is reachable, every non-terminal in w is reachable.

$*$ Remove each non-reachable non-terminal with all productions containing it.

5. Consider every terminal a in a right-hand side of length at least 2.

$*$ Add a new non-terminal A and the production $A \to a$.

$*$ Replace every occurrence of a in a right-hand side of length at least 2 with A.

6. Consider every production $A \to B_1 B_2 \dots B_n$ with $n \geq 3$.

$*$ Add a new non-terminal C and replace this production with $A \to B_1 C$ and $C \to B_2 B_3 \dots B_n$.

$*$ Repeat this step while there are changes.

The above steps must be performed

The resulting grammar is in Chomsky normal form.

Example:

$$\begin{array}{rcl} S & \rightarrow & TbT \\ T & \rightarrow & aU \\ T & \rightarrow & U \\ T & \rightarrow & V \\ U & \rightarrow & \varepsilon \\ V & \rightarrow & b \end{array}$$

Grammars in Chomsky normal form cannot generate ε. If ε is needed:

* Add a new non-terminal S' and the production $S' \rightarrow \varepsilon$.
* For each production $S \rightarrow w$ of the start symbol S, add $S' \rightarrow w$.
* Make S' the new start symbol.

2.2 The Cocke-Younger-Kasami Algorithm

Given a string $w \in \Sigma^*$ and a CFL A, is $w \in A$?

* This is the test for *membership* in a CFL.
* It is similar to parsing, but no syntax tree has to be constructed.
* $A = L(G) = \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$ for a CFG G.
* Hence $w \in A$ if and only if $S \Rightarrow_G^* w$.
* Checking all derivations does not work, since there might be infinitely many.
* Assume that G is in Chomsky normal form.
* Every non-terminal produces at least one terminal.
* It suffices to consider derivations that introduce up to $|w|$ non-terminals.
* This gives an upper bound on the length of derivations that need to be checked.
* The number of derivations might still be exponential in the length of w.

A technique to improve the running time is *dynamic programming.*

* It applies to problems whose solution can be reduced to similar, but smaller subproblems.
* The solution to a problem must be obtained from solutions to subproblems.
* This gives a recursive algorithm.
* The total number of possible subproblems must be small.
* Storing the solutions to subproblems in a table avoids repeated calculations in the recursion.
* Dynamic programming is typically applied to optimisation problems.
* A different example is the Cocke-Younger-Kasami (CYK) algorithm.

The CYK algorithm solves the membership problem $w \in L(G)$.

* Assume that G is in Chomsky normal form, for example:

$$\begin{array}{lcl} S & \rightarrow & BB \mid AS \mid a \\ A & \rightarrow & BC \\ B & \rightarrow & BS \mid b \\ C & \rightarrow & a \end{array}$$

* Let n be the length of w: for example, $n = 6$ for $w = bbabab$.
* Mark the positions that separate symbols in w:

$$\begin{array}{l} |\,b\,|\,b\,|\,a\,|\,b\,|\,a\,|\,b\,| \\ 0\;\;1\;\;2\;\;3\;\;4\;\;5\;\;6 \end{array}$$

* Let w_{ij} be the substring of w between positions i and j: for example, $w_{25} = aba$ and $w_{06} = w$.
* $N_{ij} = \{A \in N \mid A \Rightarrow^* w_{ij}\}$ contains the non-terminals that generate w_{ij}.
* The CYK algorithm calculates N_{ij} for each $0 \leq i < j \leq n$.
* Then $w \in L(G)$ if and only if $S \in N_{0n}$.

Problem is solved by generalisation:

The CYK algorithm fills a table with N_{ij} in column i, row j.

* The calculation proceeds in increasing order of substring length.

First come the substrings of length 1, that is, $j = i + 1$.

* The 1-symbol substring $w_{i,i+1}$ can be generated from A if $A \rightarrow w_{i,i+1} \in P$.
* For each production $A \rightarrow a$ where $a = w_{i,i+1}$, add A to the entry at column i, row j.
* These entries form the main diagonal of the table.

Then come the substrings of length 2, that is, $j = i + 2$.

* The 2-symbol substring $w_{i,i+2}$ is broken up into two 1-symbol substrings $w_{i,i+1}$ and $w_{i+1,i+2}$.
* If $B \in N_{i,i+1}$ and $C \in N_{i+1,i+2}$ and $A \to BC \in P$, then add A to $N_{i,i+2}$.
* These entries form the diagonal below the main diagonal.

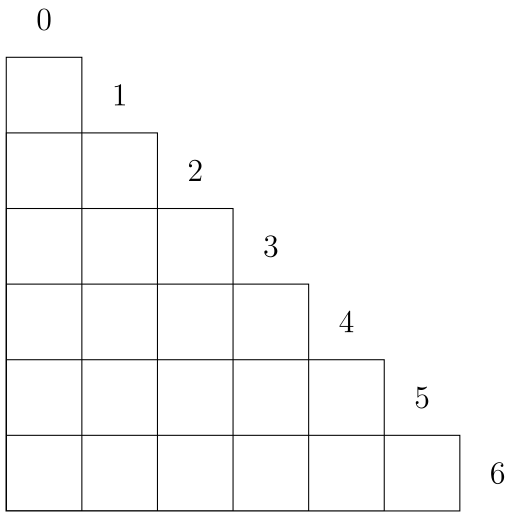
The 3-symbol substring $w_{i,i+3}$ can be broken up in two ways.

* Consider how to generate $w_{i,i+3}$ using a production $A \to BC$, that is, $A \Rightarrow BC \Rightarrow^* w_{i,i+3}$.
* This follows from $B \Rightarrow^* w_{i,i+1}$ and $C \Rightarrow^* w_{i+1,i+3}$ or from $B \Rightarrow^* w_{i,i+2}$ and $C \Rightarrow^* w_{i+2,i+3}$.
* If $B \in N_{i,i+1}$ and $C \in N_{i+1,i+3}$ and $A \to BC \in P$, then add A to $N_{i,i+3}$.
* If $B \in N_{i,i+2}$ and $C \in N_{i+2,i+3}$ and $A \to BC \in P$, then add A to $N_{i,i+3}$.

The k-symbol substring $w_{i,i+k}$ can be broken up in $k - 1$ ways.

* If $B \in N_{i,i+l}$ and $C \in N_{i+l,i+k}$ for any $1 \leq l < k$ and $A \to BC \in P$, then add A to $N_{i,i+k}$.

Example:



Implementation:

* The CYK algorithm can be implemented as follows:

for $i := 0$ **to** $n - 1$ **do**
  $N_{i,i+1} := \{A \mid (A \to w_{i,i+1}) \in P\}$
for $k := 2$ **to** n **do**
  for $i := 0$ **to** $n - k$ **do**
    $N_{i,i+k} := \emptyset$
    for $j := i + 1$ **to** $i + k - 1$ **do**
      $N_{i,i+k} := N_{i,i+k} \cup \{A \mid (A \to BC) \in P \text{ and } B \in N_{i,j} \text{ and } C \in N_{j,i+k}\}$

* The running time is $O(n^3)$.
* The underlying recursion is:

$$N_{i,i+k} = \begin{cases} \{A \mid (A \to w_{i,i+1}) \in P\} & \text{if } k = 1 \\ \bigcup_{i+1 \leq j \leq i+k-1} \{A \mid (A \to BC) \in P \text{ and } B \in N_{i,j} \text{ and } C \in N_{j,i+k}\} & \text{if } 2 \leq k \leq n \end{cases}$$

2.3 Pushdown Automata

A pushdown automaton is an extension of a finite automaton.

* A stack is added, in which stack symbols can be stored.
* Transitions may depend on symbols at the top of the stack.
* Transitions may change the symbols at the top of the stack.
* The stack size is unlimited.

A *pushdown automaton* (PDA) is a structure $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

* Q, Σ, q_0 and F are as in an NFA or a DFA,
* Γ is a finite set, the *stack alphabet*,
* $\varepsilon \notin \Sigma \cup \Gamma$,
* $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \to \mathcal{P}(Q \times \Gamma^*)$ is the transition relation.

The transition relation has the following properties.

* As in NFAs with ε-transitions, there are two sources of non-determinism.
* δ may have ε-transitions.
* δ yields a set of successor states.
* Extending NFAs, the transitions refer to the top symbols of the stack; hence Γ^* is added.

The language $A = \{a^n b^n \mid n \in \mathbb{N}\}$ is accepted by the following PDA.

* $L(M) = A$ for $M = (\{q_0, q_1\}, \{a, b\}, \{0\}, \delta, q_0, \{q_1\})$ where δ returns $\emptyset$ except for

$$\begin{aligned}
\delta(q_0, a, \varepsilon) &= \{(q_0, 0)\} \\
\delta(q_0, \varepsilon, \varepsilon) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, b, 0) &= \{(q_1, \varepsilon)\}
\end{aligned}$$

* M is represented by the following transition diagram:

The label $a, \alpha/\beta$ shows the input symbol and the top of the stack before/after the transition.

* The first symbol of α and the first symbol of β are at the top of the stack.
* By convention the PDA stack is shown growing from right to left.

A transition $(q, \beta) \in \delta(p, a, \alpha)$ has the following meaning.

* It may be applied if the following three conditions hold.
* M is in state p.
* If $a \neq \varepsilon$, the next input symbol is a.
* The top $|\alpha|$ symbols of the stack are α.
* Then the following actions take place.
* M moves to state q.
* If $a \neq \varepsilon$, a is consumed from the input.
* The top $|\alpha|$ symbols of the stack are removed; the symbols in β are pushed on the stack.

A *configuration* is an element $(q, x, \alpha) \in Q \times \Sigma^* \times \Gamma^*$.

* q is the current state.
* x is the remaining input.
* α is the content of the stack.
* A configuration describes a snapshot of M during a computation.
* The start configuration for input x is (q_0, x, ε): the stack is empty initially.
* The next-configuration relation $\rightarrow$ describes a possible transition:

* The remaining input x and the content γ below the top of the stack are unchanged.

Configuration sequences:

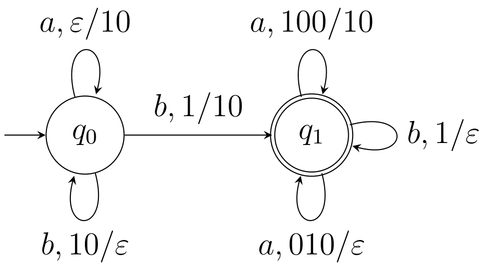
* $\rightarrow^*$ is the reflexive-transitive closure of $\rightarrow$.
* $c \rightarrow^* d$ means configuration d is reachable from configuration c in zero or more transitions.
* The *language accepted by M* is

$$L(M) = \{x \in \Sigma^* \mid (q_0, x, \varepsilon) \rightarrow^* (q, \varepsilon, \varepsilon) \text{ for some } q \in F\}.$$

* Acceptance requires the PDA to be in an accept state and to have an empty stack.

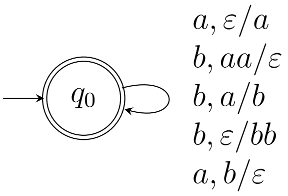
Further examples of PDAs are:

* The following PDA accepts $aababa$, but neither ab nor $aaababab$:



* The accepting sequence of configurations for input *aababa* is:

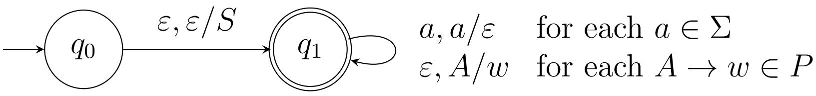
* The following PDA accepts the strings containing twice as many a-symbols as b-symbols:



2.3.1 Context-Free Languages and Pushdown Automata

Every CFL is accepted by a PDA. Proof sketch:

* Let $G = (N, \Sigma, P, S)$ be a CFG.
* Idea: simulate leftmost-derivation, keep track of sentential forms.
* Construct a PDA $M = (\{q_0, q_1\}, \Sigma, \Sigma \cup N, \delta, q_0, \{q_1\})$ with



* The transition $(q_1, S) \in \delta(q_0, \varepsilon, \varepsilon)$ pushes the start symbol S on the stack.
* A transition $(q_1, \varepsilon) \in \delta(q_1, a, a)$ *reads* terminal a.
* A transition $(q_1, w) \in \delta(q_1, \varepsilon, A)$ *expands* non-terminal A to w.
* The stack is the part of a sentential form that remains to be parsed in a leftmost derivation.
* An inductive proof shows that for each $A \in N$ and $x, y \in \Sigma^*$ and $z \in (\Sigma \cup N)^*$,

$$A \Rightarrow^* xz \text{ in a leftmost derivation if and only if } (q_1, xy, A) \to^* (q_1, y, z).$$

* In particular, $S \Rightarrow^* x$ if and only if $(q_1, x, S) \to^* (q_1, \varepsilon, \varepsilon)$, if and only if $(q_0, x, \varepsilon) \to^* (q_1, \varepsilon, \varepsilon)$.
* Thus $L(M) = L(G)$.

Example CFG with start symbol E:

$$
\begin{array}{lcl}
E & \to & T \mid E + T \\
T & \to & F \mid T * F \\
F & \to & (E) \mid n
\end{array}
$$

This results in the following PDA:

The accepting sequence of configurations and the leftmost derivation for input $n{+}n{*}n$ are:

$$
\begin{array}{rlrrl}
 & (q_0, & n{+}n{*}n, & \varepsilon) & \\
\to & (q_1, & n{+}n{*}n, & E) & E \\
\to & (q_1, & n{+}n{*}n, & E{+}T) & \Rightarrow E{+}T \\
\to & (q_1, & n{+}n{*}n, & T{+}T) & \Rightarrow T{+}T \\
\to & (q_1, & n{+}n{*}n, & F{+}T) & \Rightarrow F{+}T \\
\to & (q_1, & n{+}n{*}n, & n{+}T) & \Rightarrow n{+}T \\
\to & (q_1, & {+}n{*}n, & {+}T) & \\
\to & (q_1, & n{*}n, & T) &
\end{array}
$$

$$
\begin{array}{rlrrl}
 & (q_1, & n{*}n, & T) & \\
\to & (q_1, & n{*}n, & T{*}F) & \Rightarrow n{+}T{*}F \\
\to & (q_1, & n{*}n, & F{*}F) & \Rightarrow n{+}F{*}F \\
\to & (q_1, & n{*}n, & n{*}F) & \Rightarrow n{+}n{*}F \\
\to & (q_1, & {*}n, & {*}F) & \\
\to & (q_1, & n, & F) & \\
\to & (q_1, & n, & n) & \Rightarrow n{+}n{*}n \\
\to & (q_1, & \varepsilon, & \varepsilon) &
\end{array}
$$

Conversely, for every PDA M a CFG G can be constructed such that $L(G) = L(M)$.

* PDAs accept exactly the CFLs.
* Thus CFLs are described either by CFGs or by PDAs.
* This is similar to regular grammars and finite automata for regular languages.

Remarks on the use of PDAs for parsing:

* The PDA constructed for a CFG is non-deterministic.
* Efficient parsers use deterministic PDAs.
* Unlike finite automata, deterministic PDAs are less expressive than non-deterministic PDAs.
* Recursive-descent LL(k) parsers recognise only a small subset of the CFLs.
* The call-stack of the recursive functions corresponds to the stack of the PDA.
* Shift-reduce LR(k), LALR(k), SLR(k) parsers recognise more CFLs, but not all.

2.4 Decision Problems for Context-Free Languages

Problems for CFGs G, G' and string $x \in \Sigma^*$:

* Membership $x \in L(G)$:

* Emptiness $L(G) = \emptyset$:

* Finiteness $|L(G)| \in \mathbb{N}$:

* Universality $L(G) = \Sigma^*$ and intersection emptiness $L(G) \cap L(G') = \emptyset$:

* Equivalence $L(G) = L(G')$:

* Inclusion $L(G) \subseteq L(G')$:

Comparison of regular languages and CFL:

language	grammar	automaton	membership
regular language	regular grammar	DFA	$O(n)$
deterministic CFL	deterministic CFG	deterministic PDA	$O(n)$
CFL	CFG	PDA	$O(n^3)$

2.5 Non-Context-Free Languages

Pumping Lemma for CFLs: Let A be context-free. Then there is a number n such that each $z \in A$ with $|z| \geq n$ can be broken into five parts $z = uvwxy$ with

* $vx \neq \varepsilon$,
* $|vwx| \leq n$, and
* $uv^iwx^iy \in A$ for each $i \in \mathbb{N}$.

Proof idea:

* Let $A = L(G)$ for a CFG G in Chomsky normal form.
* A long string in A has a large syntax tree.
* A large syntax tree has a long path.
* On a long path, a non-terminal occurs twice.
* A copy of the tree below the first occurrence can be grafted at the second occurrence.
* This pumping can be repeated any number of times.

Use the pumping lemma to show that $A = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not context-free:

Another example of a non-CFL is $A = \{ww \mid w \in \{a, b\}^*\}$:

* Given $n \in \mathbb{N}$, set $z = a^n b^n a^n b^n \in A$.
* Let $z = uvwxy$ with $vx \neq \varepsilon$ and $|vwx| \leq n$.
* Set $i = 0$ and consider uwy.
* If vwx is a substring of $a^n b^n$, only the first or the second half of z is shortened.
* If vwx is a substring of $b^n a^n$, only b from the first half or a from the second half is removed.
* In either case, the two halves differ, so $uwy \notin A$.

2.5.1 Closure Properties of Context-Free Languages

CFLs are closed under union.

* Let $L_1 = L(G_1)$ and $L_2 = L(G_2)$ for CFGs $G_1 = (N_1, \Sigma, P_1, S_1)$ and $G_2 = (N_2, \Sigma, P_2, S_2)$.
* Assume that $N_1 \cap N_2 = \emptyset$ and let $S \notin N_1 \cup N_2$.
* Construct $G = (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$.
* Then $L(G) = L_1 \cup L_2$.
* Similar constructions show that CFLs are closed under concatenation and star.
* The intersection of a CFL and a regular language is a CFL.

CFLs are not closed under intersection.

* $L_1 = \{a^k b^n c^n \mid k, n \in \mathbb{N}\}$ is generated by the following CFG:

* $L_2 = \{a^n b^n c^k \mid k, n \in \mathbb{N}\}$ is generated by the following CFG:

* But $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not context-free.

CFLs are not closed under complement.

* $\overline{A} = \Sigma^* \setminus \{ww \mid w \in \{a, b\}^*\}$ is context-free.
* Consider the CFG with the productions:

$$\begin{array}{rcl} S & \to & AB \mid BA \mid A \mid B \\ A & \to & CAC \mid a \\ B & \to & CBC \mid b \\ C & \to & a \mid b \end{array}$$

* A generates all strings of odd length containing a.
* B generates all strings of odd length containing b.
* AB generates all strings of the form $uavxby$ with $u, v, x, y \in \Sigma^*$ and $|u| = |v|$ and $|x| = |y|$.
* BA generates all strings of the form $ubvxay$ with $u, v, x, y \in \Sigma^*$ and $|u| = |v|$ and $|x| = |y|$.
* Such a string z has a and b at distance $|z|/2$ from each other.
* The generated language contains all strings not of the form ww.