Lemmas, Theorems and Definitions
Pumping Lemma through page 52

Author: José Kuiper, Noud de Kroon

May 24, 2015

1 Lemmas and Theorems

1.1 Chapter 1

1 Basic Definitons	
1.1	Definition of an alphabet: Let Σ be an alphabet. The set Σ^* of strings or finite words of Σ is defined as follows: • The empty string $\epsilon \in \Sigma^*$; • if $a \in \Sigma$ and $w \in \Sigma^*$ then $aw \in \Sigma^*$
1.2	Definition of length of a string: Given an alphabet Σ, the length $\|w\|$ for a string $w \in \Sigma^*$ is given by: (i) $\|\epsilon\| = 0$, (ii) $\|aw\| = \|w\| + 1$.
1.4	Definition of concatenations: Let Σ be an alphabet. The concatenation $wv \in \Sigma^*$ of strings $w, v \in \Sigma^*$ is given by: (i) $\epsilon v = v$, and (ii) $(aw)v = a(wv)$
def	A string v is called a prefix of a string w if $vu = w$ for some string u, notation $v \preccurlyeq w$. In the situation that $v \preccurlyeq w$ we occasionaly write $u = w/v$, so if $v \preccurlyeq w$ then we have $v(w/v) = w$.
1.5	Let Σ be an alphabet and $c \in \Sigma$. The count $\#_c(w)$ of a symbol $c \in \Sigma$ in a string $w \in \Sigma^*$ is given by: • $\#_c(\epsilon) = 0$; • $\#_c(cw) = \#_c(w) + 1$; • $\#_c(aw) = \#_c(w)$ if $a \neq c$
1.6	Definition of a language: Let Σ be an alphabet. A subset $L \subseteq \Sigma^*$ is called a language over Σ.
1.8a	Definition of language concatenation: Let $L_1, L_2 \subseteq \Sigma^*$ be two languages over an alphabet Σ. The concatenation $L_1 \cdot L_2$ of L_1 and L_2 is given by: $L_1 \cdot L_2 = \{w_1 w_2 \| w_1 \in L_1, w_2 \in L_2\}$

1.8b	Definition of Kleene-closure: Let $L \subseteq \Sigma$ be a language over an alphabet Σ. The Kleene-closure L^* of L is given by: $L^*= \{w_1 \cdots w_n \mid n \geq 0, w_1, \cdots, w_n \in L\}$

1.2 Chapter 2

2.1 DFA's	
2.1	Definition of a Deterministic Finite Automaton: A DFA is a tuple $D = (Q, \Sigma, \delta, q_0, , F)$ with Q a finite set of states, Σ a finite alphabet, $\delta : Q \times \Sigma \to Q$ the transition *function*, $q_0 \in Q$ the initial state, and $F \subseteq Q$ the set of final states.
Def	Definition of $\vdash_D$: $(q, w) \vdash_D (q', w')$ iff $w = aw'$ and $\delta(q, a) = q'$ for some $a \in \Sigma$
2.3	Lemma stating some basic properties of a DFA: Let D be a DFA, then: (a) For all states q, q', q'' and words w, w' it holds that: if $q, w \vdash_D{}^* (q', w')$ and $(q, w) \vdash_D{}^* (q'', w')$ then $q' = q''$ (b) For states q, q' and all words w, w', v it holds that: $(q, w) \vdash_D{}^* (q', w')$ iff $(q, wv) \vdash_D{}^* (q', w'v)$
2.4	Definition of the language defined by a DFA: Let $D = (Q, \Sigma, \epsilon, q_0, F)$ be a finite automaton. The language $L(D) \subseteq \Sigma^*$ accepted by D is defined by: $L(D) = \{w \in \Sigma^* \mid \exists q \in F : (q_0, w) \vdash_d {}^*(q, \epsilon)\}$
2.2 NFA's	
2.7	Definiton of an NFA: (Non deterministic finite automaton with silent steps). An NFA is a quintuple $N = (Q, \Sigma, \to_N, q_0, F)$ with Q a finite set of stats, Σ a finite alphabet, $\to_N \subseteq Q \times (\Sigma \cup \{\tau\}) \times Q$ the transion *relation*, $q_0 \in Q$ the initial state, and $F \subseteq Q$ the set of final states. Important to note that $\to_N$ is a relation and not a function, meaning there may be several transitions possible from a certain state for a given letter, or none at all for a certain letter.
def	Basic definition of the $\vdash_N$ yield relation: $(q, w) \vdash_N (q', w')$ iff $\exists a \in \Sigma : q \xrightarrow{a}_N q' \land w = aw'$ or $q \xrightarrow{\tau}_N q' \land w = w'$
2.8	Lemma stating a consistency property of NFA's for all words w, w', v and states q, q': $(q, w) \vdash_N {}^*(q', w')$ iff $(q, wv) \vdash_N {}^*(q', w'v)$

2.9	Definiton of the language defined by an NFA: Let $N = (Q, \Sigma, \rightarrow_N, q_0, F)$ be a finite automaton. The language $L(N)$ accepted by N is defined by: $$L(N) = \{w \in \Sigma^* \mid \exists q \in F : (q_0, w) \vdash_N {}^*(q, \epsilon)\}$$
2.12	Theorem: If a language $L \subseteq \Sigma^*$ is accepted by a DFA, then L is also accepted by some NFA. The proof of this is trivial, since any state or function rule of a DFA is also valid in an NFA.
2.13	Theorem: If a language $L \subseteq \Sigma^*$ is accepted by an NFA, then L is also accepted by a DFA. We construct this DFA from the NFA as follows:
2.27	Pumping Lemma for regular languages: This theorem can be used to prove that a language is not regular. Details: Let L be a regular language over an alphabet $\sum$. There exists a constant $m > 0$ such that each $w \in L$ with $\|w\| > m$ can be written as $w = xyz$ where x, y, $z \in \sum^*$, $y \neq \varepsilon$, $\|xy\| \leq m$, and for all $k > 0$: $xy^kz \in L$.
2.30	Let L be a regular language over an alphabet $\sum$ represented by an NFA N accepting L. Then it can be decided if $L = \emptyset$ or not.
2.31	With this theorem you can decide whether a string is in a language or not. Details: Let $L \subseteq \sum^*$ be a regular language over the alphabet $\sum$, represented by an NFA N accepting L, and let $w \in \sum^*$ be a string over $\sum$. Then it can be decided if $w \in L$ or not. How to prove this: Construct, using the algorithm given in the proof of Theorem 2.13, a DFA D such that] $\mathcal{L}(D) = \mathcal{L}(N)$. Simulate D starting from its initial state on input w, say $(q_0, w) \vdash^*_D (q', \varepsilon)$ for some state q' of D. Decide $w \in L$ if q' is a final state of D, decide $w \notin L$ otherwise.

1.3 Chapter 3

	3.1 Push-down automata
3.2	Definition of a push-down automaton: A push-down automaton (PDA) is a septuple $P = (Q, \sum, \Delta, \emptyset, \rightarrow, q_0, F)$ where 1. Q is a finite set of states, 2. $\sum$ is a finite input alphabet with $\tau \neq \sum$, 3. Δ is a finite data alphabet or stack alphabet, 4. $\emptyset \neq \Delta$ a special symbol denoting an empty stack, 5. $\rightarrow \subseteq Q \times \sum_\tau \times \Delta_\emptyset \times \Delta^* \times Q$, where $\sum_\tau = \sum \cup \tau$ and $\Delta_\emptyset = \Delta \cup \emptyset$, is a finite set of transitions or steps,

	6. $q_0 \in Q$ is the initial state, 7. $F \subseteq Q$ is the set of final states.
3.4	Definition of configuration of a PDA: Let $P = (Q, \sum, \Delta, \emptyset, \rightarrow, q_0, F)$ be a push-down automaton. A configuration or instantaneous description (ID) of P is a triple $(q, w, x) \in Q \times \sum^* \times \Delta^*$. The relation $\vdash_P \subseteq (Q \times \sum^* \times \Delta^*) \times (Q \times \sum^* \times \Delta^*)$ is given as follows. (i) $(q, aw, dy) \vdash_P (p, w, xy)$ if $q \xrightarrow{a[d/x]} p$ (ii) $(q, w, dy) \vdash_P (p, w, xy)$ if $q \xrightarrow{\tau[d/x]} p$ (iii) $(q, aw, \varepsilon) \vdash_P (p, w, x)$ if $q \xrightarrow{a[\emptyset/x]} p$ (iv) $(q, w, \varepsilon) \vdash_P (p, w, x)$ if $q \xrightarrow{\tau[\emptyset/x]} p$
3.6	Lemma for PDA properties, stating that suffixes for strings and stacks don't matter for configurations. Let $P = (Q, \sum, \Delta, \emptyset, \rightarrow, q_0, F)$ be a push-down automaton. (a) For $w, w', v \in \sum^*$, $q, q' \in Q$, $x, x', y \in \Delta^*$ with $x \neq \varepsilon$, it holds that $(q, wv, xy) \vdash_P (q', w'v, x'y) \Leftrightarrow (q, w, x) \vdash_P (q', w', x')$ (b) For $n \geq 0$, $w_i \in \sum^*$, $q_i \in Q$, $x_i \in \Delta^*$ with $x_i \neq \varepsilon$, for $1 \leq i < n$, $v \in \sum^*$ and $y \in \Delta^*$, it holds that $(q_0, w_0v, x_0y) \vdash_P (q_1, w_1v, x_1y) \vdash_P \ldots \vdash_P (q_n, w_nv, x_ny)$ $\Leftrightarrow$ $(q_0, w_0, x_0) \vdash_P (q_1, w_1, x_1) \vdash_P \ldots \vdash_P (q_n, w_n, x_n)$
3.7	Definition on when a language is accepted by a PDA: Let $P = (Q, \sum, \Delta, \emptyset, \rightarrow, q_0, F)$ be a push-down automaton. The language $\mathcal{L}(P)$, called the language *accepted* by the push-down automaton P, is given by: $\{ w \in \sum^* \mid \exists_q \in F\, \exists_x \in \Delta^*\colon (q_0, w, \varepsilon) \vdash^*{}_P (q, \varepsilon, x) \}$ Note that we require q to be a final state of P, i.e. $q \in F$, but x can be any stack content.

3.2 Context-free grammars

3.10 Definition of a context-free grammar:
A context-free grammar (CFG) is a four-tuple $G = (V, T, R, S)$ where
1. V is a non-empty finite set of variables or non-terminals,
2. T is a finite set of terminals,
3. $R \subseteq V \times (V \cup T)^*$ is a finite set of production rules, and
4. $S \in V$ is the start symbol.

Example: $V = \{S\}$, $T = \{a, b\}$ and R consists of $S \to ab$ and $S \to aSb$.

3.13 Definition on production, production sequence and language of a CFG.
Let $G = (V, T, R, S)$ be a context-free grammar.

- Let $A \to \alpha \in R$ be a production rule of G. Thus $A \in V$ and $\alpha \in (V \cup T)^*$. Let $\gamma = \beta_1 A \beta_2$ be a string in which A occurs. Put $\gamma' = \beta_1 \alpha \beta_2$. We say that from the string γ the production rule $A \to \alpha$ produces the string γ', notation $\gamma \Rightarrow_G \gamma'$.
- A production sequence or derivation is a sequence $(\gamma_i)_{i=0}^n$ such that $\gamma_{i-1} \Rightarrow_G \gamma_i$, for $1 \leq i \leq n$. Often we write
$$\gamma_0 \Rightarrow_G \gamma_1 \Rightarrow_G \cdots \Rightarrow_G \gamma_{n-1} \Rightarrow_G \gamma_n$$
The length of this production sequence is n. In case $\gamma = \gamma_0$ and $\gamma' = \gamma_n$ we also write $\gamma \Rightarrow^*{}_G \gamma'$.
- Let $A \in V$ be a variable of G. The language $\mathcal{L}_G(A)$ generated by G from A is given by
$$\mathcal{L}_G(A) = \{ w \in T^* \mid A \Rightarrow_G^* w \}$$
- The language (G), the language generated by the CFG G, consists of all strings of terminals that can be produced from the start symbol S, i.e.
$$\mathcal{L}(G) = \mathcal{L}_G(S)$$
- A language L is called context-free, if there exists a context-free grammar G such that $L = \mathcal{L}(G)$.

3.15 Lemma on splitting and combining production sequences. This technical lemma summarizes the context independence of the machinery introduced and is used in many situations. Let $G = (V, T, R, S)$ be a context-free grammar.

(a) Let $x, x', y, y' \in (V \cup T)^*$. If $x \Rightarrow_G^n x'$ and $y \Rightarrow_G^m y'$ then $xy \Rightarrow_G^{n+m} x'y'$.

(b) Let $k \geq 1$, $X_1, ..., X_k \in V \cup T$, $n_1, ..., n_k \geq 0$, and $x_1, ..., x_k \in (V \cup T)^*$. If $X_1 \Rightarrow_G^{n_1} x_1, ..., X_k \Rightarrow_G^{n_k} x_k$, then $X_1 ... X_k \Rightarrow_G^n x_1 ... x_k$ where $n = n_1 + ... + n_k$.

(c) Let $X_1, ..., X_k \in V \cup T$, and $x \in (V \cup T)^*$. If $X_1 ... X_k \Rightarrow_G^n x$ then exist $n_1, ..., n_k \geq 0$, and $x_1, ..., x_k \in V \cup T)^*$ such that $n = n_1 + ... + n_k$, $X_1 \Rightarrow_G^{n_1} x_1, ..., X_k \Rightarrow_G^{n_k} x_k$, and $x = x_1 ... x_k$.

3.17	Definition on left- and rightmost production and sequence. Let $G = (V,\, T,\, R,\, S)$ be a context-free grammar. A production $\gamma \Rightarrow_G \gamma'$ is called a leftmost production if γ' is obtained from γ by application of a production rule of G on the leftmost variable occuring in γ, i.e., $\gamma = wA\beta$, $A \to \alpha$ a rule of G, $\gamma' = w\alpha\beta$ for some $w \in T^*$, $A \in V$, and $\alpha,\, \beta \in (V \cup T)^*$. Notation, $\gamma \overset{l}{\Rightarrow}_G \gamma'$. A leftmost derivation of G is a production sequence $(\gamma_i)_{i=0}^n$ of G for which every production is leftmost. Notation, $\gamma \overset{l}{\Rightarrow}{}^*_G \gamma'$. Similarly, one can define the notion of a rightmost production and a rightmost derivation for a CFG G.
3.18	Theorem: If L is a regular language, then L is also context-free. How to prove: Let $D = (Q,\, \sum,\, \delta,\, q_0,\, F)$ be a deterministic automaton that accepts L. Define the grammar $G = (Q,\, \sum,\, R,\, q_0)$ where R is given by $R = \{q \to aq' \mid \delta(q,\, a) = q'\} \cup \{q \to \varepsilon \mid q \in F\}$ Then prove that $L \subseteq \mathcal{L}(G)$ as well as the other way around, with the help of a string $w \in L$, from which you may conclude that: $L = \mathcal{L}(G)$. Note: the reverse of this theorem does *not* hold!

3.3 Parse trees	
3.19	Definition of parse trees: Let $G = (V,\, T,\, R,\, S)$ be a context-free grammar. The collection $\mathcal{PT}_G$ of parse trees of G, a set of rooted node-labeled trees, is given as follows: • A single root tree $[X]$ with root labeled X is a parse tree for G if X is a variable or terminal of G. • A two-node tree $[A \to \varepsilon]$, with root labeled A and leaf labeled ε is a parse tree for G if $A \in V$ and $A \to \varepsilon$ is a production rule of G. • If $PT_1,\, PT_2,\, ...,\, PT_k$ are k parse trees for G with roots $X_1,\, X_2,\, ...,\, X_k$, respectively, and $A \to X_1X_2...X_k$ is a production rule of G, then the tree $[A \to PT_1,\, PT_2,\, ...,\, PT_k]$ with root labeled A and subtrees of the root $PT_1,\, PT_2,\, ...,\, PT_k$ is a parse tree for G. Definition of yield function: The yield function: $yield : \mathcal{PT}_G \to (V \cup T)^*$ with respect to G is given by: • $yield([X]) = X$ for $X \in V \cup T$, • $yield([A \to \varepsilon]) = \varepsilon$ for $A \in V$, • $yield([A \to PT_1,\, PT_2,\, ...,\, PT_k]) = yield(PT_1) * yield(PT_2) * ... * yield(PT_k)$ for $A \in V, PT_1,\, ...,\, PT_k \in \mathcal{PT}_G$.