Introduction to Formal Language Theory — day 3

Context-free languages

Wiebke Petersen & Kata Balogh

(Heinrich-Heine-Universität Düsseldorf)

NASSLLI 2014
University of Maryland, College Park

Outline

Equivalence relation

Definition

Let M be a set. A binary Relation $R \subseteq M \times M$ on M is an ***equivalence relation*** *if*

1. *R is reflexive ($\forall x \in M : xRx$)*
2. *R is symmetric (if xRy, then yRx)*
3. *R is transitive (if xRy and yRz, then xRz)*

*An equivalence relation R on M, parts M into disjoint subsets (**equivalence classes**) M_i (with $i \in I$), where*

1. *for all $i \in I$ and $x, y \in M_i$, the relation xRy holds and*
2. *for all $i, j \in I$ with $i \neq j$ and $x \in M_i$ and $y \in M_j$, the relation xRy does not hold.*

*If $x \in M$, $[x]_R$ determines the equivalence class, that contains x. The number of equivalence classes $|\{[x]_R : x \in M\}|$ is the **index** of the equivalence relation.*

Indistinguishability relation

Definition

Let L be a language over the alphabet Σ. We define the ***indistinguishability relation*** *R_L over Σ^* as follows:*

> *xR_Ly holds iff for all $z \in \Sigma^*$ either xz and yz are both in language L or xz and yz are both not in language L.*

If two strings x and y are in relation R_L, we call them ***indistinguishable*** *with respect to language L.*

Example: a and aa are indistinguishable with respect to the language a^* but they are not indistinguishable with respect to the language $\{a^n b^n\}$.

Lemma

The indistinguishability relation is an equivalence relation.

Myhill-Nerode theorem

Proposition

A language $L \subseteq \Sigma^*$ is regular iff the index of the indistinguishability relation R_L is finite.

Proposition (Corollary)

A language $L \subseteq \Sigma^*$ is not regular iff the number of chains in Σ^*, such that they are pairwise distinguishable with respect to L, is infinite.

Example:

1. The index of R_L for $L(a(a|b)^*c)$ is 4, thus L is regular. $([\epsilon], [a], [ac], [b])$
2. The index of R_L for $L(a^i b^k : i \geq k)$ is infinite, thus L is not regular. ($[a^i]$ for $i \geq 0$ are all different)

Proof of the Myhill-Nerode theorem (I) (if regular, then finite index)

Let M be a deterministic FSA that accepts the language L. Define a further equivalence relation R_M over Σ^* as follows: xR_My iff the automaton M is in the same state $((\epsilon, q_0, x) \vdash^* (x, q, \epsilon), (\epsilon, q_0, y) \vdash^* (y, q, \epsilon))$ after processing the strings x and y.

- Every equivalence class of R_M is associated with a state in M.
- Since the number of states is finite, the index of R_M has to be finite as well.
- If xR_My holds, then xzR_Myz holds for any $z \in \Sigma^*$.
- If we assume that xR_My holds and $z \in \Sigma^*$, then xz will be accepted by automaton M iff yz is also accepted by the automaton.
- Therefore $xz \in L$ holds iff $yz \in L$.
- Therefore from xR_My follows xR_Ly.
- Thus every equivalence class of R_M is a subset of the equivalence class of R_L.
- Since the index of R_M is finite, the index of R_L has to be finite as well.

Proof of the Myhill-Nerode theorem (II) (if finite index, then regular)

Let L be a regular language. Thus the index of R_L is finite.
Let $[x_1]_{R_L}, [x_2]_{R_L}, \dots [x_n]_{R_L}$ be the n equivalence classes of R_L. Then we can define a detFSA M_L that accepts L as follows:
$M_L = (Q, \Sigma, \delta, S, F)$ with:

- $Q = \{[x_1]_{R_L}, [x_2]_{R_L}, \dots [x_n]_{R_L}\}$,
- $S = [\epsilon]_{R_L} \quad (= [x_i]_{R_L}$ if $\epsilon \in [x_i]_{R_L})$,
- $F = \{[x_i]_{R_L} | x_i \in L\}$,
- $\delta([x]_{R_L}, a) = [xa]_{R_L}$.

A detFSA with n (= index of R_L) states is called a **minimal detFSA** for the language L. Every detFSA with n states that accepts the language L, can be derived from M_L by renaming the states.

detFSA minimization (algorithm)

Given a detFSA M (where all states are accessible from initial state).

1. Create a table with all pairs of states $q_i \neq q_j$.
2. Mark all pairs (q_i, q_j) with $q_i \in F$ and $q_j \notin F$ (or the other way around).
3. Check for every unmarked pair (q_i, q_j) and every symbol $a \in \Sigma$ whether $(\delta(q_i, a), \delta(q_j, a))$ is marked or not. If it is marked, also mark (q_i, q_j).
4. Repeat step 3 as long as you can add new marks.
5. Merge all unmarked pairs to one state.

Pumping lemma for regular languages

Lemma (Pumping-Lemma)

If L is a regular language over Σ, then there exists $n \in \mathbb{N}$ such that every word $z \in L$ with $|z| \geq n$ can be written as $z = uvw$ such that

- $|v| \geq 1$
- $|uv| \leq n$
- $uv^iw \in L$ *for any* $i \geq 0$.

proof sketch:

- Any regular language is accepted by a deterministic FSA with a finite number n of states.
- While reading in z with $|z| \geq n$ the detFSA passes at least one state q_j twice.

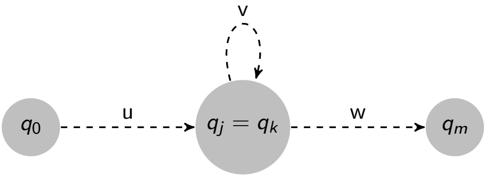
Pumping lemma for regular languages (cont.)

Lemma (Pumping-Lemma)

If L is a regular language over Σ, then there exists $n \in \mathbb{N}$ such that every word $z \in L$ with $|z| \geq n$ can be written as $w = uvw$ such that

- $|v| \geq 1$
- $|uv| \leq n$
- $uv^i w \in L$ *for any* $i \geq 0$.

proof sketch:



Let q_j be the first state that is passed twice, then $|u| < n$ and $|uv| \leq n$

$L = \{a^m b^m : m \geq 0\}$ is not regular

- Suppose L is regular and n is the natural number associated with L by the pumping lemma. Let $z = a^n b^n$ and write $z = uvw$ with $|uv| \leq n$ and $|v| \geq 1$.
- $|uv| \leq n$ implies that u and v can only consist of a's.
- The pumping lemma implies that $uv^i w \in L$ for any $i \geq 0$, but $uvvw$ has more a's as uvw (remember $|v| \neq \epsilon$).
- Thus either uvw or $uvvw$ is not an element of L.
- Contradiction to the assumption that L is regular.

Closure properties of regular languages

A language class is closed under an operation if its application to arbitrary languages of this class

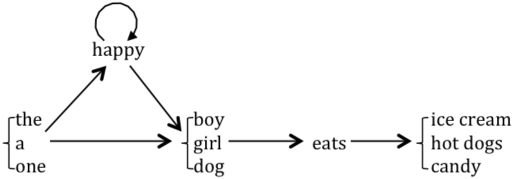
	Type3	Type2	Type1	Type0
union	+ ✓	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+ ✓	+	+	+
Kleene's star	+ ✓	+	+	+
intersection with a regular language	+	+	+	+

complement: construct complementary DFSA

intersection: implied by *de Morgan*

Are natural languages regular?

- Pinker: Finite State Model (Markov Model / word chain device)
  - a model whereby a sentence is produced one word at a time
  - each successive word limits the choice of the next word



- it was considered plausible until the 1950's
- problems for modeling natural languages, e.g.:
  - long-distance dependencies and sentence embedding
  - the FSM cannot handle hierarchical / tree-like structures
  - structural ambiguity
  - recursion (embedding)
- Chomsky (*Syntactic structures*, 1957): English is not a regular language.

Long distance dependencies and embedding

- *if ..., then ...* / *either ..., or ...* structures
  - (a) Either John$_i$ is sick or he$_i$ is depressed.
  - (b) Either Mary$_i$ knows [that John$_j$ thinks that he$_j$ is sick] or she$_i$ is depressed.
- arbitrary (sentence) embedding possible, e.g.

  The cheese [that the mouse stole]$_S$ was expensive.
  The cheese [that the mouse [that the cat caught]$_S$ stole]$_S$ was expensive.
  The cheese [that the mouse [that the cat [that the dog chased]$_S$ caught]$_S$ stole]$_S$ was expensive.
  The cheese [that the mouse [that the cat [that the dog [that Peter bought]$_S$ chased]$_S$ caught]$_S$ stole]$_S$ was expensive.

Constituency

- words can be hierachically grouped to bigger units ⇒ phrases / constituents
  - $[_{NP}$ Felix$]$ $[_{VP}$ slept$]$.
  - $[_{NP}$ A cat$]$ $[_{VP}$ slept$]$.
  - $[_{NP}$ A small cat$]$ $[_{VP}$ slept$]$.
  - $[_{NP}$ A small grey cat$]$ $[_{VP}$ slept$]$.
  - $[_{NP}$ Rose$]$ $[_{VP}$ $[_{V}$ admires$]$ $[_{NP}$ Felix$]$ $]$.
  - $[_{NP}$ Rose$]$ $[_{VP}$ $[_{V}$ admires$]$ $[_{NP}$ an actor$]$ $]$.
  - $[_{NP}$ Rose$]$ $[_{VP}$ $[_{V}$ admires$]$ $[_{NP}$ an actor $[_{S}$ who likes Felix$]$ $]$ $]$.

Structural ambiguity

- one sentence with two (or more) different syntactic analyses
- two (or more) different phrase structure trees
- e.g. *Sherlock saw the man with the binoculars.*
  - $[_S$ $[_{NP}$ Sherlock$]$ $[_{VP}$ $[_V$ saw$]$ $[_{NP}$ the man$]$ $[_{PP}$ with the binoculars$]$ $]$.
  - $[_S$ $[_{NP}$ Sherlock$]$ $[_{VP}$ $[_V$ saw$]$ $[_{NP}$ the man $[_{PP}$ with the binoculars$]$ $]$ $]$.
- other different ambiguities:
  - lexical ambiguity; e.g. *The fisherman went to the bank.*
  - scope ambiguity; e.g. *Every student read a book.*

Natural languages are not regular

- see, e.g., the example of nested dependency:
  - a woman met another woman
  - a woman whom another woman hired hired another woman
  - a woman whom another woman whom another woman hired hired met another woman
  - ... etc.
- formal proof using closure under intersection and the pumping lemma for regular languages
- recall: $L1_{REG} \cap L2_{REG} = L_{REG}$
- we cannot directly apply the Pumping Lemma to English
- but we can use a common strategy: intersection and homomorphism
- homomorphism f: $f(\text{a woman}) = w$, $f(\text{whom another woman}) = x$, $f(\text{hired}) = y$, $f(\text{met another woman}) = z$
  - wx^*y^*z is a regular language; and
  - f(English) $\cap\ wx^*y^*z = wx^ny^nz$
- we can apply the Pumping Lemma to wx^ny^nz
- $\Rightarrow x^ny^n$ is not regular $\Rightarrow$ English is not regular

Context-free language

Definition

A grammar (N, T, S, R) is ***context-free*** *if all production rules in R are of the form:*

$$A \to \beta \text{ with } A \in N \text{ and } \beta \in (N \cup T)^* \setminus \{\epsilon\}$$

Additionally, the rule $S \to \epsilon$ is allowed if S does not occur in any rule's right-hand side. A language generated by a context-free grammar is said to be context-free.

Proposition

The set of context-free languages is a strict superset of the set of regular languages.

Proof: Each regular language is per definition context-free. $L(a^n b^n)$ is context-free but not regular ($S \to aSb, S \to \epsilon$).

Note: $S \to \epsilon$ is only allowed if S does not occur in any rule's right-hand side, however the problem can always be eliminated ($S \to \epsilon, S \to T, T \to aTb, T \to ab$)

Examples of context-free languages

- $L_1 = \{ww^R : w \in \{a, b\}^*\}$
  - generated by the grammar $G_1 = (\{S\}, \{a, b\}, S, R)$ with
    R = {S → ϵ, S → aSa, S → bSb }
- $L_2 = \{a^i b^j : i \geq j\}$
- $L_3 = \{w \in \{a, b\}^* : \text{more } a's \text{ than } b's\}$
- $L_4 = \{w \in \{a, b\}^* : \text{number of } a's \text{ equals number of } b's\}$
  - generated by the grammar $G_4 = (\{a, b\}, \{S, A, B\}, S, R)$ with
    R = {S → aB, S → bA, A → aS, B → bS, A → bAA, B → aBB,
    A → a, B → b }

Ambiguous grammars and ambiguous languages

Definition

Given a context-free grammar G: A derivation which always replaces the leftmost nonterminal symbol is called ***left-derivation***

Definition

A context-free grammar G is ***ambiguous*** *iff there exists a $w \in L(G)$ with more than one left-derivation, $S \rightarrow^* w$.*

Definition

A context-free language L is ***ambiguous*** *iff each context-free grammar G with $L(G) = L$ is ambiguous.*

Recall: there is a one-to-one correspondence between left-derivations and derivation trees.

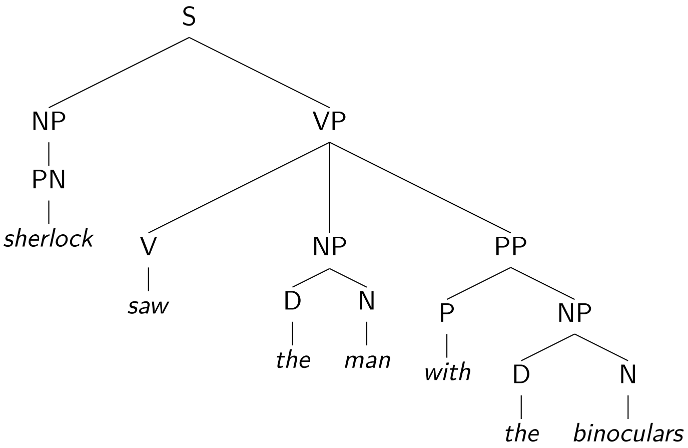
Example of an ambiguous grammar

- $G = (N, T, S, R)$ with
  N $=$ {D, N, NP, P, PP, PN, V, VP, S},
  T $=$ {*the, man, binoculars, sherlock, with, saw*},

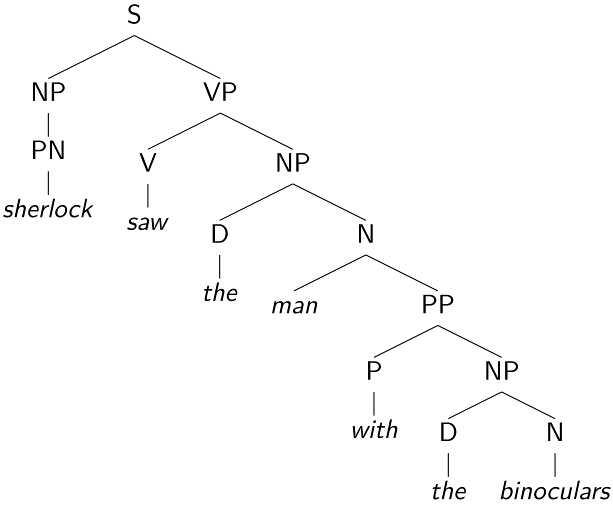
$$R = \left\{ \begin{array}{l} \mathrm{S} \to \mathrm{NP\ VP}, \mathrm{VP} \to \mathrm{V\ NP}, \mathrm{VP} \to \mathrm{V\ NP\ PP}, \\ \mathrm{NP} \to \mathrm{PN}, \mathrm{NP} \to \mathrm{D\ N}, \mathrm{NP} \to \mathrm{D\ N\ PP}, \mathrm{PP} \to \mathrm{P\ NP}, \\ \mathrm{V} \to \mathit{saw}, \mathrm{PN} \to \mathit{sherlock}, \mathrm{N} \to \mathit{man}, \\ \mathrm{N} \to \mathit{binoculars}, \mathrm{D} \to \mathit{the}, \mathrm{P} \to \mathit{with} \end{array} \right\}$$

- left-derivations:
  - S $\Rightarrow$ NP VP $\Rightarrow$ PN VP $\Rightarrow$ Sherlock VP $\Rightarrow$ Sherlock V NP $\Rightarrow$ Sherlock saw NP $\Rightarrow$ ...
  - S $\Rightarrow$ NP VP $\Rightarrow$ PN VP $\Rightarrow$ Sherlock VP $\Rightarrow$ Sherlock V NP PP $\Rightarrow$ Sherlock saw NP PP $\Rightarrow$ ...

Example of an ambiguous grammar

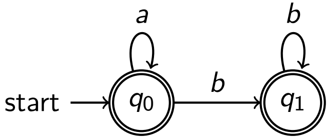


Example of an ambiguous grammar



Push-down automaton

- we saw, that the regular language a^*b^* can be accepted by an FSA

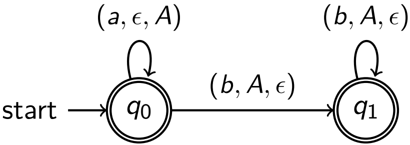


- take now the language a^nb^n $\rightarrow$ we cannot create an FSA that accepts a^nb^n, since the 'loops' do not 'remember' how many a's are read at a given moment $\Rightarrow$ we need some kind of "memory"
- **Push-down Automaton** (PDA)
  - a PDA is essentially an FSA augmented with an *auxiliary tape* or *stack* on which it can read, write, and erase symbols
  - 'last in – first out' (LIFO) system
  - the stack can be seen as a kind of "memory"
- context-free languages are accepted by Push-down Automata

Example PDAs

- a PDA for language $a^n b^n$
- a PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ with

  $Q = \{q_0, q_1\}$ (set of states)
  $\Sigma = \{a, b\}$ (input alphabet)
  $\Gamma = \{A\}$ (stack alphabet)
  q_0 (initial state)
  $F = \{q_0, q_1\}$ (set of final states)
  $\delta = \{(q_0, a, \epsilon) \rightarrow (q_0, A), (q_0, b, A) \rightarrow (q_1, \epsilon), (q_1, b, A) \rightarrow (q_1, \epsilon)\}$



Push-down automaton

Definition

*A **nondeterministic push-down automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ with:*

1. *a finite, nonempty set of **states** Q*
2. *an alphabet Σ with $Q \cap \Sigma = \emptyset$*
3. *a stack alphabet Γ with $\Sigma \cap \Gamma = \emptyset$*
4. *a **transition** relation $\delta : (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*)$*
5. *an **initial state** $q_0 \in Q$ and*
6. *a set of **final states** $F \subseteq Q$.*

- nondeterministic and deterministic PDAs are **not** equivalent!

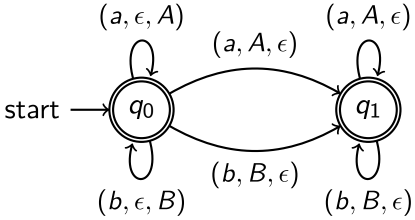
Push-down automaton

- transition rules of the form $(q_i, x, \alpha) \to (q_k, \beta)$
- the transitions include not only change of state but also operations on the stack: *pop and push*
- transition rules from state q_1 to state q_2, while reading a, and
  - pop A from the stack: $(q_1, a, A) \to (q_2, \epsilon)$
  - push B to the stack: $(q_1, a, \epsilon) \to (q_2, B)$
  - pop A and push B: $(q_1, a, A) \to (q_2, B)$
  - no stack operation: $(q_1, a, \epsilon) \to (q_2, \epsilon)$
- according to the definition A and B can also be strings over Γ
- a PDA accepts an input string iff
  - the entire input string has been read
  - the PDA is in a final (accepting) state
  - the stack is empty

Example PDAs

- a PDA for language ww^R
- a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ with

  $Q = \{q_0, q_1\}$ (set of states)
  $\Sigma = \{a, b\}$ (input alphabet)
  $\Gamma = \{A, B\}$ (stack alphabet)
  q_0 (initial state)
  $F = \{q_0, q_1\}$ (set of final states)
  $\delta = \{(q_0, a, \epsilon) \to (q_0, A), (q_0, b, A) \to (q_1, \epsilon), (q_1, b, A) \to (q_1, \epsilon)\}$



Chomsky Normal Form

Definition

A grammar is in ***Chomsky Normal Form (CNF)*** *if all production rules are of the form*

1. $A \to a$
2. $A \to BC$

with $A, B, C \in T$ *and* $a \in \Sigma$ *(and if necessary* $S \to \epsilon$ *in which case* S *may not occur in any right-hand side of a rule).*

Proposition

Each context-free language is generated by a grammar in CNF.

Proposition

No node in a derivation tree of a grammar in CNF has more than two daughters.

Chomsky Normal Form

- Each context-free language is generated by a grammar in CNF.
- Given a context-free grammar G with $\epsilon \notin L(G)$

3 steps

1. Eliminate complex terminal rules.
2. Eliminate chain rules.
3. Eliminate $A \rightarrow B_1 B_2 \ldots B_n$ $(n > 2)$ rules.

CNF: eliminate complex terminal rules

Aim: Terminals only occur in rules of type $A \to a$

1. Introduce a new non-terminal X_a for each terminal a occurring in a complex terminal rule.
2. Replace a by X_a in all complex terminal rules.
3. For each X_a add a rule $X_a \to a$.

$$\begin{aligned} S &\to ABA|B \\ A &\to aA|C|a \\ B &\to bB|b \\ C &\to A \end{aligned} \quad \Rightarrow \quad \begin{aligned} S &\to ABA|B \\ A &\to X_aA|C|a \\ B &\to X_bB|b \\ C &\to A \\ X_a &\to a \\ X_b &\to b \end{aligned}$$

CNF: eliminate chain rules

Aim: No rules of the form $A \rightarrow B$

- For each circle $A_1 \rightarrow A_2, \ldots, A_{k-1} \rightarrow A_k$, $A_k \rightarrow A_1$ replace in all rules each A_i by a new non-terminal A' and delete all $A' \rightarrow A'$-rules.
- Remove stepwise all rules $A \rightarrow B$ and add for each $B \rightarrow \beta$ a rule $A \rightarrow \beta$

$$\begin{aligned} S &\rightarrow ABA|B \\ A &\rightarrow X_aA|C|a \\ B &\rightarrow X_bB|b \\ C &\rightarrow A \\ X_a &\rightarrow a \\ X_b &\rightarrow b \end{aligned} \quad \Rightarrow \quad \begin{aligned} S &\rightarrow A'BA'|X_bB|b \\ A' &\rightarrow X_aA'|a \\ B &\rightarrow X_bB|b \\ X_a &\rightarrow a \\ X_b &\rightarrow b \end{aligned}$$

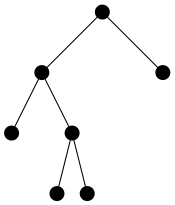
CNF: $A \to B_1 B_2 \dots B_n$ $(n > 2)$

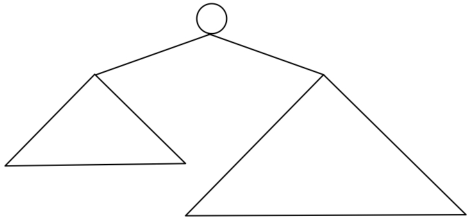
Aim: not more than two non-terminals in one rule's right-hand side

- For each rule of the form $A \to B_1 B_2 \dots B_n$ introduce a new non-terminal $X_{B_2 \dots B_n}$.
- Remove the rule and add two new rules:
  - $A \to B_1 X_{B_2 \dots B_n}$
  - $X_{B_2 \dots B_n} \to B_2 \dots B_n$

$$
\begin{aligned}
S &\to A'BA' | X_b B | b \\
A' &\to X_a A' | a \\
B &\to X_b B | b \\
X_a &\to a \\
X_b &\to b
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
S &\to A' X_{BA'} | X_b B | b \\
A' &\to X_a A' | a \\
B &\to X_b B | b \\
X_a &\to a \\
X_b &\to b \\
X_{BA'} &\to BA'
\end{aligned}
$$

binary trees





Proposition

If T is an arbitrary binary tree with at least 2^k leafs, then height$(T) \geq k$.

Proof by induction on k. The proposition is true for $k = 0$. Given the proposition is true for some fixed k, let T be a tree with $\geq 2^{k+1}$ leafs. T has two subtrees of which at least one has 2^k leafs. Thus the height of T is $\geq 2^{k+1}$.

Corollary

If a context-free grammar is in CNF, then the height of a derivation tree of a word of length $\geq 2^k$, then height(T) is greater than k (note that the last derivation step is always a unary one).

Pumping lemma for context-free languages

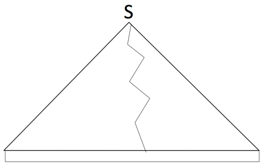
Lemma (Pumping Lemma)

For each context-free language L there exists a $n \in \mathbb{N}$ such that for any $z \in L$: if $|z| \geq n$, then z may be written as $z = uvwxy$ with

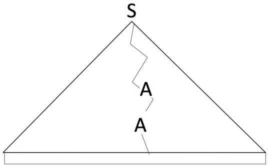
- $u, v, w, x, y \in T^*$,
- $|vwx| \leq p$,
- $vx \neq \epsilon$ *and*
- $uv^iwx^iy \in L$ *for any* $i \geq 0$.

Pumping lemma: proof sketch

Let $k = |N|$ and $n = 2^k$. Be $z \in L$ with $|z| \geq n$.

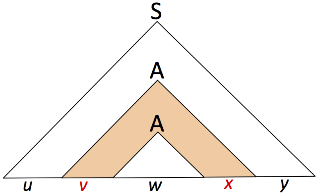


Because of $|z| \geq 2^k$ there exists a path in the binary part of the derivation tree of z of length $\geq k$.



At least one non-terminal symbol occurs twice on the path.
Starting from the bottom of the path, let A be the first non-terminal occurring twice.

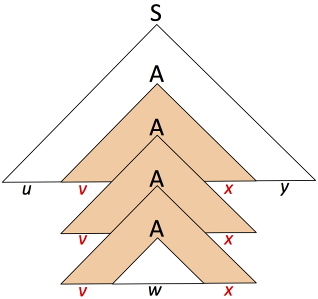
Pumping Lemma: proof sketch

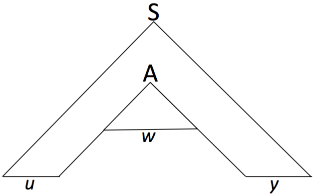


$|vwx| \leq n$ (A is chosen such that no non-terminal occurs twice in the trees spanned by the upper of the two A's)

$vx \neq \epsilon$ (a binary rule $A \to BC$ must have been applied to the upper A).

Pumping Lemma: proof sketch





$uv^i wx^i y \in L$ for any $i \geq 0$.

Pumping Lemma: application

The language $L(a^k b^m c^k d^m)$ is not context-free

- Assume that $L(a^k b^m c^k d^m)$ is context-free then there is a $n \in \mathbb{N}$ as specified by the Pumping Lemma.
- Choose $z = a^n b^n c^n d^n$, and $z = uvwxy$ in accordance with the Pumping Lemma.
- Because of $vwx \leq n$ the string vwx consists either of only a's, of a and b's, only of b's, of b and c's, only of c's,. . . .
- It follows that the pumped word uv^2wx^2y cannot be in L.
- That contradicts the assumption that L is context-free.

Closure properties of context-free languages

	Type3	Type2	Type1	Type0
union	+	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+	+	+	+
Kleene's star	+	+	+	+
intersection with a regular language	+	+	+	+

union: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \to S_1, S \to S_2\}$

intersection: $L_1 = \{a^n b^n a^k\}$, $L_2 = \{a^n b^k a^k\}$, but $L_1 \cap L_2 = \{a^n b^n a^n\}$

complement: *de Morgan*

concatenation: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \to S_1 S_2\}$

Kleene's star: $G = (N_1 \cup \{S\}, T_1, S, P)$ with $P = P_1 \cup \{S \to S_1 S, S \to \epsilon\}$

decision problems

Given: grammars $G = (N, \Sigma, S, P)$, $G' = (N', \Sigma, S', P')$, and a word $w \in \Sigma^*$

word problem Is w derivable from G ?

emptiness problem Does G generate a nonempty language?

equivalence problem Do G and G' generate the same language $(L(G) = L(G'))$?

Results for the decision problems

	Type3	Type2	Type1	Type0
word problem	D	D	D	U
emptiness problem	D	D	U	U
equivalence problem	D	U	U	U

D: decidable; U: undecidable