Course 5

Pumping Lemma

- Not all languages are regular
- How to decide if a language is regular or not?

- Idea: pump symbols

Example: $L = \{0^n1^n \mid n >= 0\}$

Theorem: (Pumping lemma, Bar-Hillel)

Let **L** be a regular language. $\exists p \in N$, such that if $w \in L$ with $|w| > p$, then

$$w = xyz, \text{ where } 0 < |y| <= p$$

and

$$xy^iz \in L, \ \forall i \geq 0$$

Proof

L regular => $\exists$ M = (Q,$\boldsymbol{\Sigma}$,$\boldsymbol{\delta}$, q_0, F) such that L= L(M)

Let |Q| = p

$$\left.\begin{array}{l} \text{If } w \in L(M)\text{: } (q_0,w) \overset{*}{\vdash} (q_f , \boldsymbol{\varepsilon}) , q_f \in F \\ \qquad\text{and} \\ |w|>p \end{array}\right\} \begin{array}{l} \text{process at least p+1 symbols} \\ \\ \text{p states} \end{array}$$

$\Rightarrow\exists$ q_1 that appear in at least 2 configurations

$(q_0,xyz) \overset{*}{\vdash} (q1,yz) \overset{+}{\vdash} (q1,z) \overset{*}{\vdash} (q_f , \boldsymbol{\varepsilon})$, $q_f \in F$ => 0<=|y|<=p

Proof (cont)

$$\begin{aligned}
(q_0, xy^iz) &\overset{*}{\vdash} (q_1, y^iz) \\
&\overset{*}{\vdash} (q_1, y^{i-1}z) \\
&\overset{*}{\vdash} \dots \\
&\overset{*}{\vdash} (q_1, yz) \\
&\overset{*}{\vdash} (q_1, z) \\
&\overset{*}{\vdash} (q_f, \boldsymbol{\varepsilon}),\ q_f \in F
\end{aligned}$$

So, if $w=xyz \in L$ then $xy^iz \in L$, for all $i>0$

If $i=0$: $(q_0,xz) \overset{*}{\vdash} (q_1, z) \overset{*}{\vdash} (q_f , \boldsymbol{\varepsilon})$, $q_f \in F$

Example: $L = \{0^n1^n \mid n>= 0\}$

Suppose L is regular => $w= xyz = 0^n1^n$

Consider all possible decomposition =>

Case 1. $y = 0^k$

$xyz = 0^{n-k}0^k1^n; xy^iz = 0^{n-k}0^{ik}1^n \notin L$

Case 2. $y = 1^k$

$xyz = 0^n1^k1^{n-k}; xy^iz = 0^n1^{ik}1^{n-k} \notin L$

Case 3. $y = 0^k 1^l$

$xyz = 0^{n-k}0^k1^l1^{n-l}; xy^iz = 0^{n-k}(0^k1^l)^i1^{n-l} \notin L$

Case 4. $y = 0^k 1^K$

$xyz = 0^{n-k}0^k1^k1^{n-k}; xy^iz = 0^{n-k}0^k1^k0^k1^k...1^{n-l} \notin L$

=> L is not regular

Context free grammars (cfg)

Context free grammar (cfg)

- Procdutions of the form: A $\rightarrow\alpha$, A∈N, α∈(N∪$\boldsymbol{\Sigma}$)*

- More powerful

- Can model programming language:

  G = (N, $\boldsymbol{\Sigma}$,P,S) s.t. L(G) = programming language

Syntax tree

Definition: A syntax tree corresponding to a cfg G = (N, $\boldsymbol{\Sigma}$,P,S) is a tree obtained in the following way:

1. Root is the starting symbol S
2. Nodes ∊ N∪$\boldsymbol{\Sigma}$:
    1. Internal nodes ∊N
    2. Leaves ∊$\boldsymbol{\Sigma}$
3. For a node A the descendants in order from left to right are X_1, X_2, ..., X_n only if A → $X_1X_2 \ldots X_n$∊ P

Remarks:

a) Parse tree = syntax tree – result of parsing (syntatic analysis)

b) Derivation tree – condition 2.2 not satisfied

c) Abstract syntax tree (AST) ≠ syntax tree (semantic analysis)

Syntax tree (cont)

Property: In a cfg G = (N, $\boldsymbol{\Sigma}$,P,S), w ∊ L(G) <u>if and only if</u> there exists a syntax tree with frontier w.

Proof: HomeWork

Example: S-> aSbS | c; w = aacbcbc

Leftmost derivations	**Rightmost derivations**
S => aSbS => aaSbSbS => aacbSbS => aacbcbS => aacbcbc	S => aSbS => aSbc => aaSbSbc => aaSbcbc => aacbcbc

Definition: A cfg G = (N, $\boldsymbol{\Sigma}$,P,S) is ambigous if for a w ∊ L(G) there exists 2 distinct syntax tree with frontier w.

Example:

Parsing (syntax analysis) modeled with cfg:

cfg G = (N, $\boldsymbol{\Sigma}$,P,S):

- N – nonterminal: syntactical constructions: declaration, statement, expression, a.s.o.
- $\boldsymbol{\Sigma}$ – terminals; elements of the language: identifiers, constants, reserved words, operators, separators
- P – syntactical rules – expressed in BNF – simple transformation
- S – syntactical construct corresponding to program

THEN

Program syntactically correct <=> $w \in L(G)$

Equivalent transformation of cfg

- Unproductive symbols
- Inaccesible symbols

- ε - productions
- Single productions

1. Determine elements (symbols/ productions): Greedy alg
2. eliminate them: construct equivalent grammar

Unproductive symbols

Definition

A nonterminal A este ***unproductive*** in a cfg if does not generate any word: $\{w \mid A \Rightarrow^* w, w \in \Sigma^*\} = \emptyset$.

Algorithm 1: Elimination of unproductive symbols

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

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

// idea: build $N_0, N_1, \ldots$ recursively (until saturation)

```
step 1: $N_0 = \emptyset$; i:=1;
step 2: $N_i = N_{i-1} \cup \{A \mid A \to \alpha \in P, \alpha \in (N_{i-1} \cup \Sigma)^*\}$
step 3: if $N_i <> N_{i-1}$    then i:=i+1; goto step 2
                           else $N' = N_i$
step 4: if $S \notin N'$         then $L(G) = \emptyset$
                           else $P' = \{A \to \alpha \mid A \to \alpha \in P$ and $A \in N'\}$
```

Example

G = ({S,A,B,C,D}, {a,b,c}, P,S)

P: $S \to aA \mid aC$

$A \to AB$

$B \to b$

$C \to aC \mid CD$

$D \to b$

Inaccesible symbols

Definition

A symbol $X \in N \cup \Sigma$ is ***inaccesible*** in a cfg if X does not appear in any sentential form: $\forall\ S \Rightarrow^* \alpha,\ X \notin \alpha$

Algorithm 2: Elimination of inaccessible symbols

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

output: $G' = (N',\Sigma',P',S)$, $L(G) = L(G')$ and

$\forall X \in N \cup \Sigma\ \exists \alpha, \beta \in (N' \cup \Sigma')^*$ *s.t.* $S \Rightarrow^*_{G'} \alpha X \beta$.

```
step 1: $V_0 = \{S\}$; $i:=1$;
step 2: $V_i = V_{i-1} \cup \{X \mid \exists A \to \alpha X \beta \in P, A \in V_{i-1}\}$
step 3: if $V_i <> V_{i-1}$   then $i:=i+1$; goto step 2
                              else   $N' = N \cap V_i$
                                     $\Sigma' = \Sigma \cap V_i$
                                     $P' = \{A \to \alpha \mid A \to \alpha \in P, A \in N', \alpha \in (N \cup \Sigma)^*\}$
```

Example

G = ({S,A,B,C,D}, {a,b,c,d}, P,S)

P: $S \rightarrow aA \mid aC$

$A \rightarrow AB$

$B \rightarrow b$

$C \rightarrow aC \mid bCb$

$D \rightarrow bB \mid d$

ε-productions

Definition

A cfg $G=(N,\Sigma,P,S)$ is without ***ε-productions*** if

1. $P \not\ni A \to \varepsilon$ (ε-productions)

OR

2. $\exists\ S\to\varepsilon$ si $S \notin rhs(p), \forall p \in P$

Algorithm 3: Elimination of ε-productions

input: cfg $G = (N,\Sigma,P,S)$

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

```
step 1: construct $\overline{N} = \{A \mid A \in N, A \Rightarrow^+ \varepsilon\}$
        1.a.    $N_0 := \{A \mid A\to\varepsilon \in P\}$;
                i := 1;
        1.b. $N_i := N_{i-1} \cup \{A \mid A\to\alpha \in P, \alpha \in N^*_{i-1}\}$
        1.c. **if** $N_i <> N_{i-1}$ **then** i:=i+1; **goto** step 1.b
                        **else** $\overline{N} = N_i$

step 2: Let P' = set of productions built:
        2.a. **if** $A\to\alpha_0 B_1\alpha_1 B_2\alpha_2 \ldots B_k\alpha_k \in P$, k>=0
                **and** for i := 1,k $B_i \in \overline{N}$
                and $\alpha_j \notin \overline{N}$, j:=0,k
             **then** add to P' all prod of the form
                $A\to\alpha_0 X_1\alpha_1 X_2\alpha_2 \ldots X_k\alpha_k$
                where $X_i$ is $B_i$ or $\varepsilon$ (not $A\to\varepsilon$)
        2.b **if** $S \in N'$ **then** add S' to N' and $S'\to S \mid \varepsilon$ to P
                **else** N' := N; S' := S.
```

A->BC
B->ε
C->ε

Example

G = ({S,A,B}, {a,b},P,S)

P: $S \to aA \mid aAbB$

$A \to aA \mid B$

$B \to bB \mid \boldsymbol{\varepsilon}$

Single productions

Definition

O production of the form $A \to B$ is called single production or renaming rule.

Algorithm 4 : Elimination of single productions

Input: cfg G, without ε-productions

Output: G' s.t. L(G) = L(G')

```
For each $A \in N$ build the set $N_A = \{B \mid A \Rightarrow^* B\}$ :
1.a. $N_0 := \{A\}$, $i := 1$
1.b. $N_i := N_{i-1} \cup \{C \mid B \to C \in P$ si $B \in N_{i-1}\}$
1.c. if $N_i \neq N_{i-1}$ then $i := i+1$ goto 1.b.
                 else $N_A := N_i$
P': for all $A \in N$ do
         for all $B \in N_A$ do
          if $B \to \alpha \in P$ and not "single" then $A \to \alpha \in P'$
```

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

Example

G = ({E,T,F},{a,(,),+,*},P,E)

P: $E \rightarrow E+T \mid T$

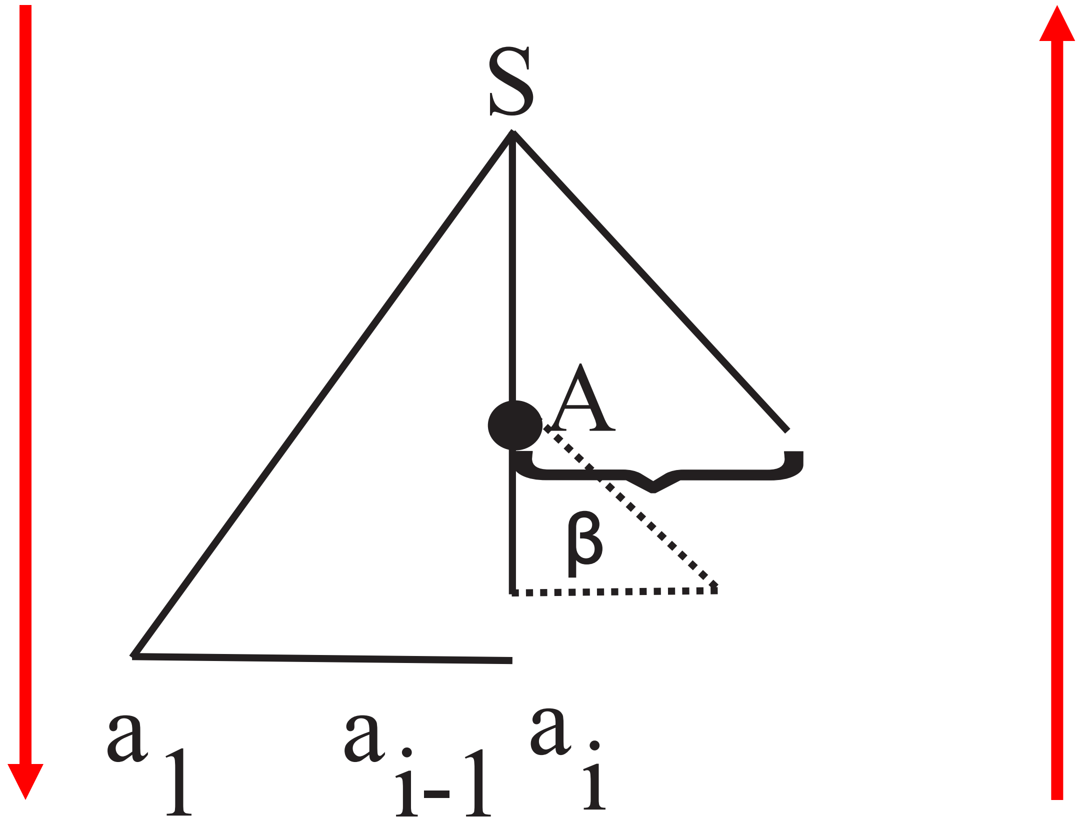
$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid a$

Parsing

- Cfg G = (N, **Σ**, P,S) check if w ∈ L(G)
- Construct parse tree

- How:
    1. Top-down vs. Bottom-up
    2. Recursive vs. linear


S
A
β
a_1
a_{i-1}
a_i