Course 5

Context free grammars (cfg)

Context free grammar (cfg)

- Procdutions of the form: A $\rightarrow \alpha$, A$\in$N, $\alpha \in$(N$\cup \Sigma$)*

- More powerful

- Can model programming language:

  G = (N, Σ,P,S) s.t. L(G) = programming language

Syntax tree

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

1. Root is the starting symbol S
2. Nodes $\in$ N$\cup\pmb{\Sigma}$:
   1. Internal nodes $\in$ N
   2. Leaves $\in\pmb{\Sigma}$
3. For a node A the descendants in order from left to right are X_1, X_2, ..., X_n only if A $\rightarrow$ X_1X_2... $X_n\in$ 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) $\neq$ syntax tree (semantic analysis)

Definition: A cfg G = (N, Σ,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, Σ,P,S):

- N – nonterminal: syntactical constructions: declaration, statement, expression, a.s.o.
- Σ – 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 ∈ 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: $\{w \mid A \Rightarrow^* w, w \in \Sigma^*\} = \varnothing$.

Algorithm 1: Elimination of unproductive symbols

input: G = (N,Σ,P,S)

output: G' = (N',Σ,P',S), L(G) = L(G')

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

step 1: $N_0 = \varnothing$; i:=1;

step 2: $N_i = N_{i-1}$ U $\{A \mid A \rightarrow \alpha \in P, \alpha \in (N_{i-1}$ U $\Sigma)^*\}$

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) = $\varnothing$

                         else P' = $\{A \rightarrow \alpha \mid A \rightarrow \alpha \in P$ and $A \in N'\}$

Example

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

P: S → aA | aC

       A → AB

       B → b

       C → aC | CD

       D → b

Inaccesible symbols

Algorithm 2: Elimination of inaccessible symbols

input: G = (N,Σ,P,S)

output: G' = (N',Σ',P',S), L(G) = L(G') and

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

step 1: V_0 = {S}; i:=1;

step 2: V_i = V_{i-1} $\cup$ {X| $\exists$ A $\rightarrow$ $\alpha X\beta$ $\in$ 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

                                             Σ' = Σ $\cap$ V_i

                                             P' = {A$\rightarrow\alpha$| A$\rightarrow\alpha$ $\in$ P, A $\in$ N', α $\in$(N $\cup\Sigma$)* }

Example

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

P: S $\rightarrow$ aA | aC

A $\rightarrow$ AB

B $\rightarrow$ b

C $\rightarrow$ aC | bCb

D $\rightarrow$ bB | d

ε -productions

Algorithm 3: Elimination of ε-productions

input: cfg G = (N,Σ,P,S)
output: cfg G' = (N',Σ,P',S')

step 1: construct $\overline{N}$ = {A| A $\in$ N, A=>$^+$ ε}

   1.a. N_0 := {A| A$\rightarrow\varepsilon$ $\in$ P};

        i := 1;

  1.b. N_i := N_{i-1} U {A| A$\rightarrow\alpha$ $\in$ P, α $\in$ N^*_{i-1}}

  1.c. **if** N_i <> N_{i-1} **then** i:=i+1; **goto** step 1.b

          else $\overline{N}$ = N_i

A->BC
B->ε
C->ε

step 2: Let P' = set of productions built:

  2.a. **if** A$\rightarrow\alpha_0 B_1\alpha_1 B_2\alpha_2$ . . . $B_k\alpha_k$ $\in$ 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$\rightarrow\alpha_0 X_1\alpha_1 X_2\alpha_2$ . . . $X_k\alpha_k$

      where X_i is B_i or ε (not A$\rightarrow\varepsilon$)

  2.b **if** S $\in$N' **then** add S' to N' and S'$\rightarrow$ S|ε to P

      else N' := N; S' := S.

Example

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

P: S → aA | aAbB

         A → aA | B

         B → bB | **ε**

Single productions

Definition
O production of the form A→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∈N build the set N_A ={B| A$\Rightarrow^*$B} :

1.a. N_0:={A}, i:=1

1.b. N_i:= N_{i-1} ∪ {C | B→C ∈ P si B ∈ 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∈N **do**

       for all B∈N_A **do**

          if B→α ∈ P **and not** "single" **then** A→α ∈ P'　　　　　　　G' =(N,Σ,P',S)

Example

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

P: E → E+T | T

        T → T*F | F

        F → (E) | a

Parsing

- Cfg $G = (N, \mathbf{\Sigma}, P, S)$ check if $w \in L(G)$

- Construct parse tree

- How:
    1. Top-down vs. Bottom-up
    2. Recursive vs. linear
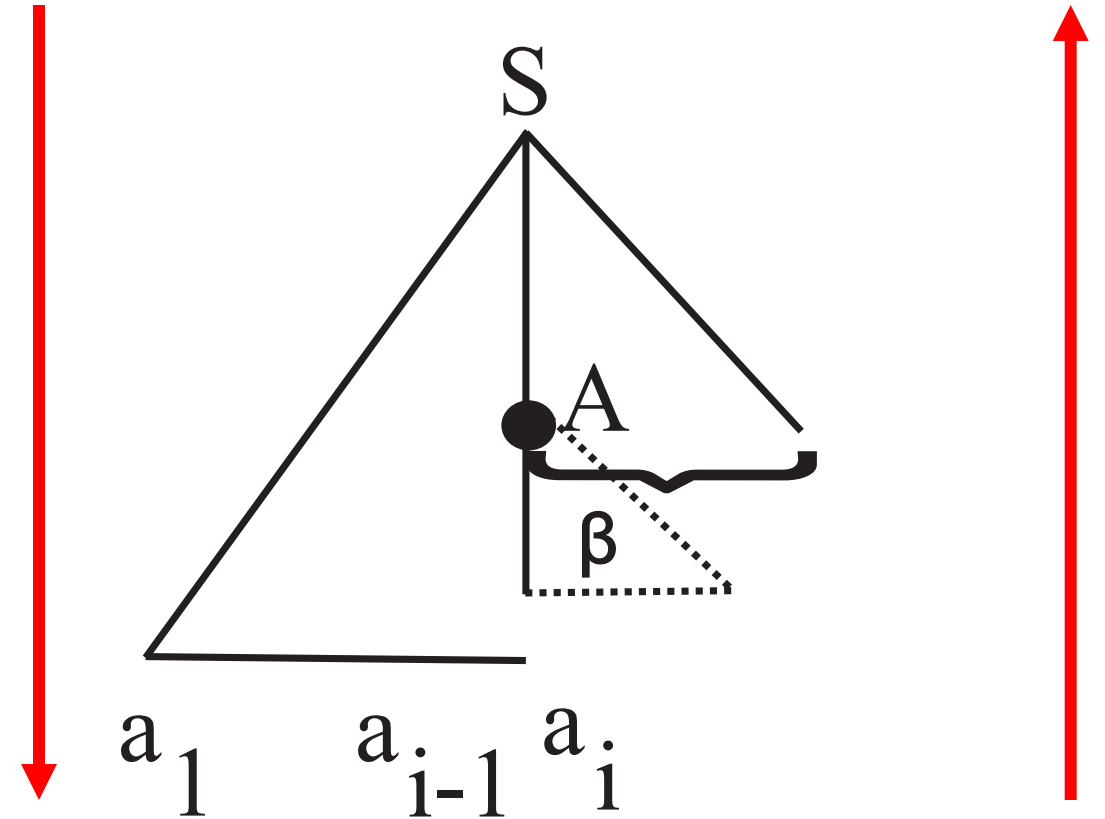
	Descendent	**Ascendent**
Recursive	Descendent recursive parser	Ascendent recursive parser
Linear	LL(k): LL(1)	LR(k): LR(0), SLR, LR(1), LALR

Result – parse tree -representation

- Arbitrary tree – child sybling representation

- Sequence of derivations S => α_1 => α_2 =>... => α_n = w

- String of production – index associated to prod – which prod is used at each derivation step: 1,4,3,...

index	Info	Parent	Right sibling
1	S	0	0
2	a	1	0
3	S	1	2
4	b	1	3
5	S	1	4
6	c	3	0
7	c	5	0