Normal Forms (Chomsky, Greibach) for CFGs

(A) Eliminate useless symbols

Definition

A symbol $X \in V \cup T$ is called :

- **generating** if $X \Rightarrow^* z$ for some $z \in T^*$
- **reachable** if $S \Rightarrow^* \alpha X \beta$ for some $\alpha, \beta \in (V \cup T)^*$
- **useful** if it is both generating and reachable

Example

$S \rightarrow A\,B \mid a$; $B \rightarrow b$; $C \rightarrow cD \mid b$

A is non-generating ; C is non-reachable ;
D is both non-reachable and non-generating

Algorithm for eliminating useless symbols

Given a CFG $G = (V, T, R, S)$

*(1) Eliminate all **non-generating** symbols to end up in the CFG* **: $G1 = (V_1, T_1, R_1, S)$.**

Do this by the following inductive method :

Basis **:** *Elements of **T** are **generating** by definition of zero step derivation.*

Induction **:** *If for a production $A \to \alpha$ all the elements of α are **generating** or $\alpha = e$ then A is generating.*

If X is non-generating then remove all productions of the form $X \to \alpha$ and $C \to \alpha X \beta$

*(2) Eliminate all **non-reachable** symbols to end up in the CFG* **: $G2 = (V_2, T_2, R_2, S)$.**

Basis **: *S*** *is **reachable** by definition.*

Induction **:** *If within a production $A \to \alpha$, A is **reachable** then all the elements of α are **reachable***

If X is non-reachable then remove all productions of the form $X \to \alpha$

Fact *: After **first** removing all productions involving **nongenerating** variables on its LHS or RHS and **then** removing productions involving **unreachable** symbols (terminals and nonterminals) all remaining symbols are useful ; i.e. both **reachable** and **generating** !*

Consider the productions

$$S \rightarrow A\,B \mid a$$
$$B \rightarrow b$$

then all ***A,B , a*** *and* ***b*** *are* ***reachable.***

But at the next step of generability ***A*** *is* ***non-generating ,*** *hence the new grammar has the productions :*

$$S \rightarrow a$$
$$B \rightarrow b$$

But then ***B*** *is* ***non-reachable*** *which is missed out in the first step.*

Hence the ***correct*** *algorithmic method is : (1) Eliminate* ***non-generating*** *symbols and productions first and (2) Eliminate the* ***non-reachable*** *symbols out of the remaining symbols and productions*

Applied to the example above first eliminate the ***non-generating*** *variable* ***A*** *and the associated production* $S \rightarrow A\,B$ *and then eliminate the* ***non-reachable*** *symbol* ***B*** *and the associated production* $B \rightarrow b$

Theorem

*The CFG **G2** generated by the algorithm above has the property :*

***(1)** Every non- terminal and terminal variable of **G2** is **useful** in **G** ,*

*i.e. it is both **generating** and **reachable** in **G***

(2) $L_G = L_{G2}$

Proof *Exercise* **: Prove (1)**

*We prove **(2)** in two steps :* **(i)** $L_G \subseteq L_{G2}$ *and* **(ii)** $L_{G2} \subseteq L_G$

(i) $w \in L_G$ *and* $S \Rightarrow_G \ldots \Rightarrow_G \alpha_j \ldots \Rightarrow_G w$, *be a derivation of* w *in* **G** *then*

$S \Rightarrow_{G2} \ldots \Rightarrow_{G2} \alpha_j \ldots \Rightarrow_{G2} w$ *,since each* α_j *consists only of useful terms by definition*

*(ii) is trivially true since **G2** is a sub-grammar of **G***

(B) Eliminate e (epsilon) productions : $A \rightarrow e$

Definition

*A is called **nullable** if $A \Rightarrow^* e$*

Compute all nullable variables inductively

***Basis** : A is nullable if $A \rightarrow e$;*

***Induction** : If $B \rightarrow C_1C_2 \ldots C_n$ and each C_i is nullable*

*then **B** is nullable*

Algorithm to eliminate e-productions

Construct a new grammar $G' = (V,T,R',S)$ from $G = (V,T,R,S)$

*Productions in **R** are of the form $A \rightarrow B_1B_2 \ldots B_m$ where $k \leq m$ of the B_j*

*non-terminal variables are **nullable***

Include in $\boldsymbol{R'}$*,* $\boldsymbol{2^k}$ *productions where each nullable* $\boldsymbol{B_j}$ *is present or absent (except when* $\boldsymbol{m=k}$ *avoid the* $\boldsymbol{A \rightarrow e}$ *case that corresponds to absence of all terms) ; also* ***remove*** *all productions of the form* $\boldsymbol{A \rightarrow e}$

Theorem $\boldsymbol{L_{G'} = L_G - \{e\}}$

Proof : *For any production used in a derivation use the version where the eventually nullified variables are absent !*

(C) Eliminating unit productions : $A \rightarrow B$, $B \in V$

Definition

A production of the form $\boldsymbol{A \rightarrow B}$ *is called a* ***unit*** *production*

*Call **(A,B)** with $A,B \in V$ a **unit pair** if $A \Rightarrow^* B$ where only **unit productions** are used in the derivation*

- Algorithm *to determine unit pairs*

*Construct a **digraph D** where variables are the nodes and there is a directed edge from **A** to **B** iff there is a unit production $A \rightarrow B$.*

*Then **(A,B)** is a **unit pair** iff there is a path from **A** to **B** in **D** or **A=B**.*

- Algorithm *for computing unit production-free $G' = (V,T,R',S)$ from **G***

*(1) Compute all **unit pairs** of **G***

*(2) Include all **non-unit productions** of **R** in **R'** and in addition for each unit pair **(A,B)** add to **R'** the production $A \rightarrow \alpha$ if $B \rightarrow \alpha$ is a non-unit production in **R***

Theorem $L_{G'} = L_G$

Example *for elimination of* ***null*** *productions*

$S \rightarrow ABC \mid e \; ; \; A \rightarrow aAb \mid e \; ; B \rightarrow bCc \mid e \; ; C \rightarrow c \mid e$

S,A,B,C *are all nullable , hence productions of the new grammar* ***G'*** *are*

$S \rightarrow ABC \mid AB \mid AC \mid BC \mid A \mid B \mid C \; ; A \rightarrow aAb \mid ab \; ; B \rightarrow bCc \mid bc \; ; C \rightarrow c$

Example *for elimination of* ***unit*** *productions*

$S \rightarrow A \mid Ba \; ; \; A \rightarrow B \mid aC \; ; B \rightarrow C \mid bA \; ; C \rightarrow B \mid c$

Unit pairs are : ***(S,A) , (S,B) , (S,C), (A,B) ,(A,C) , (B,C) , (C,B)***

hence productions of the new grammar ***G'*** *are*

$S \rightarrow Ba \mid aC \mid bA \mid c \; ; \; A \rightarrow aC \mid bA \mid c \; ; B \rightarrow bA \mid c \; ; C \rightarrow c \mid bA$

Chomsky Normal Form (CNF)

2 kinds of productions are allowed and there are no useless symbols :

(1) $A \rightarrow BC$, $B,C \in V$

(2) $A \rightarrow a$, $a \in T$

Algorithm for computing the CNF

(i) eliminate ***(a)*** *epsilon productions ;* ***(b)*** *unit productions ;* ***(c)*** *useless symbols (first nongenerating then nonreachable)*

(ii)For every production of the form $W \rightarrow X_1 X_2 \ldots X_n$, *if* $X_i \in T$ *then replace* X_i *with a new variable* Λ_i *in this production and add the new production* $\Lambda_i \rightarrow X_i$

(iii) Replace every production of the type $A \rightarrow B_1 B_2 \ldots B_n$ *for* $n \geq 3$ *with the productions :* $A \rightarrow B_1 C_1$, $C_1 \rightarrow B_2 C_2$, … , $C_{n-2} \rightarrow B_{n-1} B_n$ *where* C_i , $i = 1, \ldots, n-2$ *are new variables.*

Example (Chomsky Normal Form) (Start symbol is E)

$E \rightarrow T \mid E+T$

$T \rightarrow F \mid T*F$

$F \rightarrow I \mid (E)$

$I \rightarrow 0 \mid 1J \mid x0 \mid x1J \mid$

$J \rightarrow 0J \mid 1J \mid e$

Eliminate null production $J \rightarrow e$

$E \rightarrow T \mid E+T$

$T \rightarrow F \mid T*F$

$F \rightarrow I \mid (E)$

$I \rightarrow 0 \mid 1 \mid 1J \mid x0 \mid x1 \mid x1J$

$J \rightarrow 0J \mid 1J \mid 0 \mid 1$

Eliminate unit pairs

$E \rightarrow T \rightarrow F \rightarrow I$

unit pairs (E,T),(E,F),(E,I),(T,F),(T,I),(F,I)

$E \rightarrow 0 \mid 1 \mid 1J \mid x0 \mid x1 \mid x1J \mid (E) \mid T*F \mid E+T$

$T \rightarrow 0 \mid 1 \mid 1J \mid x0 \mid x1 \mid x1J \mid (E) \mid T*F$

$F \rightarrow 0 \mid 1 \mid 1J \mid x0 \mid x1 \mid x1J \mid (E)$

$I \rightarrow 0 \mid 1 \mid 1J \mid x0 \mid x1 \mid x1J$ (*I is nonreachable*)

$J \rightarrow 0J \mid 1J \mid 0 \mid 1$

$E \rightarrow 0 \mid 1 \mid one\ J \mid X\ zero \mid X\ one \mid X\ one\ J \mid [E] \mid T\ mult\ F \mid E\ plus\ T$

$T \rightarrow 0 \mid 1 \mid one\ J \mid X\ zero \mid X\ one \mid X\ one\ J \mid [E] \mid T\ mult\ F$

$F \rightarrow 0 \mid 1 \mid one\ J \mid X\ zero \mid X\ one \mid X\ one\ J \mid [E]$

$J \rightarrow zero\ J \mid one\ J \mid 0 \mid 1$

$zero \rightarrow 0$ $\quad X \rightarrow x$ $\quad [\ \rightarrow ($ $\quad mult \rightarrow *$

$one \rightarrow 1$ $\quad]\ \rightarrow)$ $\quad plus \rightarrow +$

Example (Chomsky Normal Form , continued)

$E \rightarrow 0 \mid 1 \mid one\ J \mid X\ zero \mid A\ J \mid X\ one \mid B\ ] \mid CF \mid DT$

$T \rightarrow 0 \mid 1 \mid one\ J \mid X\ zero \mid A\ J \mid X\ one \mid B\ ] \mid CF$

$F \rightarrow 0 \mid 1 \mid one\ J \mid X\ zero \mid AJ \mid X\ one \mid B\ ]$

$J \rightarrow zero\ J \mid one\ J \mid 0 \mid 1$

$zero \rightarrow 0$

$one \rightarrow 1$

$X \rightarrow x$

$[\rightarrow ($

$] \rightarrow)$

$mult \rightarrow *$

$add \rightarrow +$

$A \rightarrow X\ one$

$B \rightarrow [\ E$

$C \rightarrow T\ mult$

$D \rightarrow E\ add$

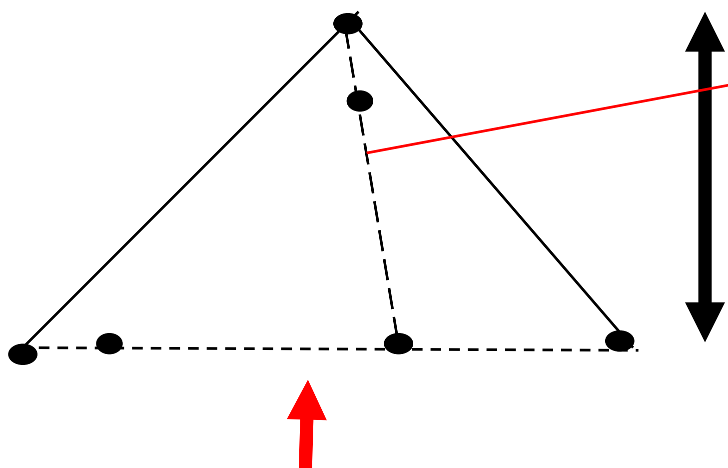
The Pumping Lemma for CFGs

The structure of a ***Parse Tree*** of a *CFG* (= *binary* tree if in *CNF*)

Let $\mathbf{m} := |V|$ *and choose a word* z *of length* $|z| = k \geq 2^{m+1}$

then if **d** *is the depth of the parse tree for* z *then* $2^{m+1} \leq |z| = k \leq 2^{d}$

hence $m+1 \leq d$ *; and thus at least one* ***variable*** *in* ***V*** *occurs repeated on the longest path !*



longest path length = d ⇒
d variable nodes on the path

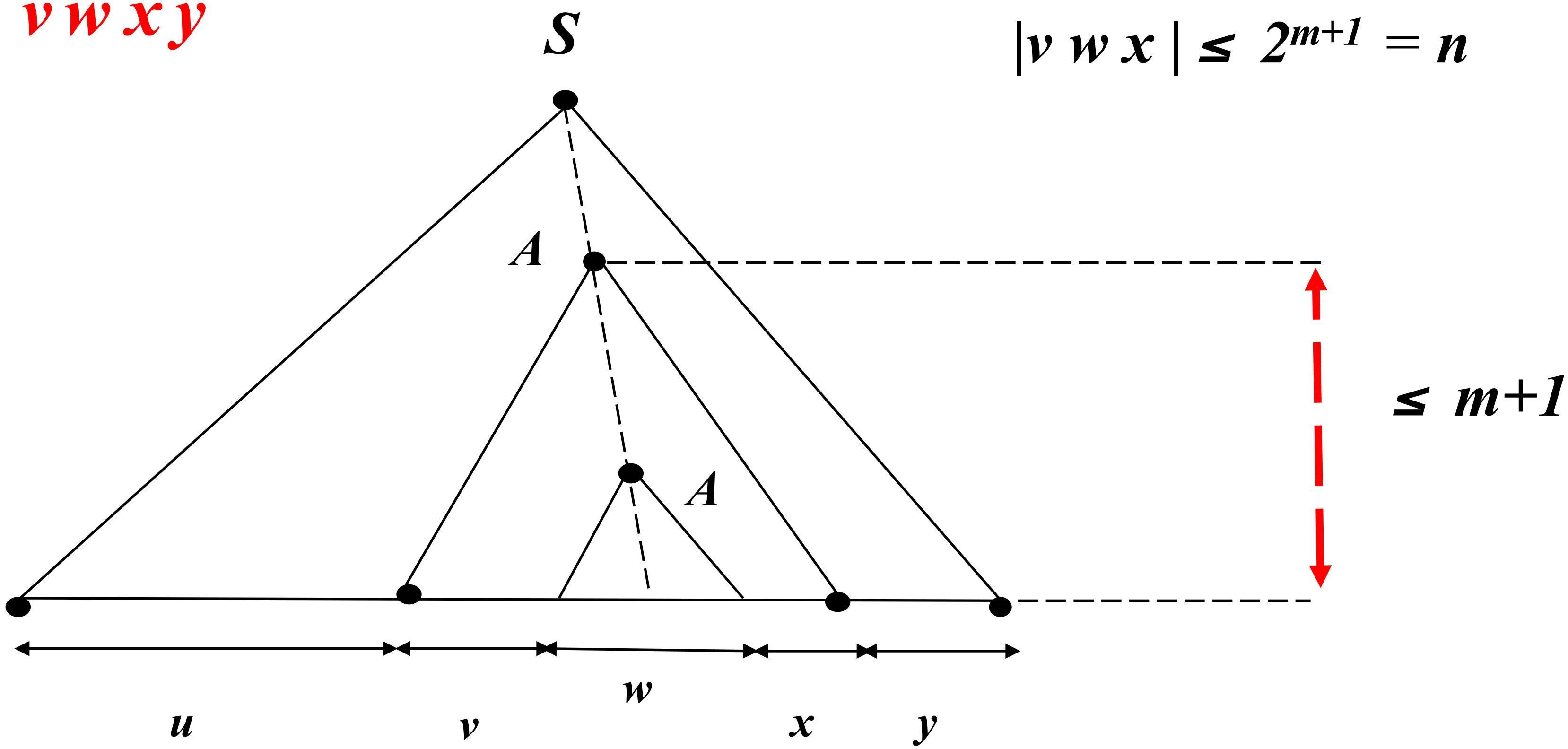
d *=depth of the tree = longest path from the root*

$$\boxed{n := 2^{m+1} \leq k \leq 2^{d} \Rightarrow m+1 \leq d}$$

k = # *leaves = length of the yield string* z

$z = u\,v\,w\,x\,y$

$|v\,w\,x| \le 2^{m+1} = n$



$A \Rightarrow^* v\,A\,x$ and $A \Rightarrow^* w$ hence $A \Rightarrow^* v^i w\,x^i$, $i = 0,1\ldots$

hence $S \Rightarrow^* uAy \Rightarrow^* u\,v^i w\,x^i y$,

$i = 0, 1, \ldots$ where $|vwx| \le n$ and $|v\,x| > 0$

Pumping Lemma for CFGs

*Let **L** be a CFL . Then there exists a constant **n** such that for any string*

$z \in L$ with $|z| \geq n$, z can be written as $z = u\ v\ w\ x\ y$ where :

(1) $|v\ w\ x| \leq n$

(2) $|v\ x| > 0$

(3) $u\ v^i\ w\ x^i\ y \in L$ *for all* $i \geq 0$

Applications of the Pumping Lemma

The following are examples of non-CF languages

1 - $L = \{ a^k b^k c^k \mid k \geq 1 \} \subseteq \{a, b, c\}^*$

2 - $L = \{ a^k b^m c^k d^m \mid k, m \geq 1 \} \subseteq \{a, b, c, d\}^*$

3 - $L = \{ t\,t \mid t \in \{a, b\}^* \}$

1 – Let $\boldsymbol{n}$ *be as in Pumping Lemma and choose* $z = \boldsymbol{a^n b^n c^n \in L}$. *Then by PL*

$\boldsymbol{a^n b^n c^n = uvwxy}$ *and we show that* $\boldsymbol{uwy \notin L}$, *a contradiction to PL.*

Since by PL $|\boldsymbol{vwx}| \leq n$ *either : (i)* $\boldsymbol{vwx = a^k}$ *or* $= \boldsymbol{b^k}$ *or* $= \boldsymbol{c^k}$ *where* $\boldsymbol{0 < k \leq n}$

or : (ii) $\boldsymbol{vwx = a^i b^j}$ *or* $= \boldsymbol{b^i c^j}$ *where* $\boldsymbol{0 < i+j \leq n}$

moreover again by PL , $\boldsymbol{p := |vx| > 0}$ *, hence :*

If (i) holds then $\boldsymbol{uwy} = \boldsymbol{a^{n-p} b^n c^n}$ *or* $= \boldsymbol{a^n b^{n-p} c^n}$ *or* $= \boldsymbol{a^n b^n c^{n-p}}$

If (ii) holds then $\boldsymbol{uwy = a^m b^k c^n}$ *or* $= \boldsymbol{a^n b^m c^k}$ *where* $\boldsymbol{m+k = 2n - p < 2n}$

for all cases $\boldsymbol{uwy \notin L}$ *and the result follows.*

*2 – Let **n** be as in Pumping Lemma (PL) and choose* $z = a^n b^n c^n d^n \in L$. *Then by PL* $a^n b^n c^n d^n = uvwxy$ *and we show that* $uwy \notin L$ *, a contradiction to PL.*

Since $|vwx| \leq n$ *, either **vwx** covers (i) one symbol among **a,b,c** and **d** or (ii) contains two adjacent symbols*

If (i) holds then $vwx = a^k$ *or* $= b^k$ *or* $= c^k$ *or* $= d^k$ *where* $0 < k \leq n$

If (ii) holds then $vwx = a^i b^j$ *or* $= b^i c^j$ *or* $= c^i d^j$ *where* $0 < i+j \leq n$

moreover by PL , $p := |vx| > 0$ *, hence :*

If (i) holds then $uwy = a^{n-p} b^n c^n d^n$ *or* $= a^n b^{n-p} c^n d^n$ *or* $= a^n b^n c^{n-p} d^n$ *or* $= a^n b^n c^n d^{n-p}$

If (ii) holds then $uwy = a^m b^k c^n d^n$ *or* $= a^n b^m c^k d^n$ *or* $= a^n b^n c^m d^k$ *where* $m,k \leq n$, $m+k = 2n - p <$.

In all cases $uwy \notin L$ *and the result follows.*

3 - Let n be as in Pumping Lemma (PL) and choose $z= a^nb^na^nb^n \in L$. Then by PL $a^nb^na^nb^n = uvwxy$ .We show that $uwy \notin L$, a contradiction to PL.

Since by PL $|vwx| \leq n$, either ; (i) $vwx = a^k$ or $vwx = b^k$; $0< k \leq n$, or :

(ii) $vwx = a^rb^q$ or ; $vwx = b^ra^q$; $0 <r+q \leq n$, and by PL $p := |vx| > 0$.

If (i) holds then $uwy =a^{n-p}b^na^nb^n$ or $=a^nb^na^{n-p}b^n$; or

$uwy =a^nb^{n-p}a^nb^n$ or $=a^nb^na^nb^{n-p}$ where p is as above hence clearly $uwy \notin L$.

If (ii) holds then $uwy = a^ib^ja^nb^n$; or $uwy = a^nb^ja^ib^n$;or $uwy = a^nb^na^ib^j$

with $i,j \leq n$ and $i+j =2n-p < 2n$ where again p is as above.

in all cases above $uwy \notin L$.

Properties of Context Free Languages

Theorem 1 *(Substitution)*

Let $\boldsymbol{L}$ *be a CFL over an alphabet* $\boldsymbol{\Sigma}$ *and for each* $\boldsymbol{a} \in \boldsymbol{\Sigma}$ *let* $\boldsymbol{L(a)}$ *be a CFL over an alphabet* $\boldsymbol{\Sigma_a}$. *Then the language :*

$$L_s := \{\, u \in \Sigma^* \mid u = l(a_1).l(a_2) \ldots l(a_n)\ ;\ a_1.a_2 \ldots a_n \in L\ ;\ l(a_k) \in L(a_k) \text{ for } k = 1, \ldots, n \,\}$$

is a CFL over the alphabet $\bigcup_{a \in \Sigma} \Sigma_a$

Proof : *Let* $\boldsymbol{G = (V, \Sigma, R, S)}$ *generate* $\boldsymbol{L}$ *and let each* $\boldsymbol{G_a = (V_a, \Sigma_a, R_a, S_a)}$ *generate* $\boldsymbol{L_a}$ *then the grammar* $\boldsymbol{G'}$ *that replaces each* **'a'** *in the productions of* $\boldsymbol{G}$ *by* $\boldsymbol{S_a}$ *and incorporate all the variables and productions of* $\boldsymbol{G_a}$*'s in* $\boldsymbol{G'}$ *generates* $\boldsymbol{L_s}$ …

Theorem 2

The (i) union, (ii) concatenation , (iii) Kleene ('') and positive ('+') closure and (iv) string reversal of CFLs are context-free languages.*

Proof : Use substitution theorem for ***(i)-(iii)*** *!*

(i) $L_1 \cup L_2 \rightarrow$ *Choose* $L = \{ a , b \}$ *and* $L_a = L_1$ *and* $L_b = L_2$

(ii) $L_1 L_2 \rightarrow$ *Choose* $L = \{ a\, b \}$ *and* $L_a = L_1$ *and* $L_b = L_2$

(iii) $M^* (M^+) \rightarrow$ *Choose* $L = \{ a \}^*$ $(L = \{ a \}^+)$ *and* $L_a = M$

(iv) *Construct* G_R *by reversing each production in* G **.**

Then each leftmost derivation of w *in* G *has a symmetric rightmost derivation in* G_R *that generates* w^R

Theorem 3

If $\boldsymbol{L}$ *is a CFL and* $\boldsymbol{R}$ *a regular language then* $\boldsymbol{L \cap R}$ *is a CFL*

Proof : Let the PDA $\boldsymbol{P}$ *accept* $\boldsymbol{L}$ *and let the DFA* $\boldsymbol{A}$ *accept* $\boldsymbol{R}$.

Then the product automaton $\boldsymbol{P \times A}$ *which is a PDA accepts* $\boldsymbol{L \cap R}$

What is a product automaton $\boldsymbol{P \times A}$ *?*

Let $\boldsymbol{\delta_P}$ *and* $\boldsymbol{\delta_A}$ *be the transition functions of* $\boldsymbol{P}$ *and* $\boldsymbol{A}$ *then :*

$\boldsymbol{((q',r'),\alpha) \in \delta_{P\times A}((q,r),a,X)}$ ***iff***

$\boldsymbol{(q',\alpha) \in \delta_P(q,a,X)}$ **and :** *(i) if* $\boldsymbol{a \neq e}$ *then* $\boldsymbol{\delta_A(r,a)=r'}$; *(ii) if* $\boldsymbol{a=e}$ *then* $\boldsymbol{r'=r}$

where $\boldsymbol{q,q'}$ *and* $\boldsymbol{r,r'}$ *are elements of the state sets* $\boldsymbol{Q}$ *of* $\boldsymbol{P}$ *and* $\boldsymbol{R}$ *of* $\boldsymbol{A}$ *respectively.*

Theorem 4

The intersection and complementation of CFLs are not necessarily context-free

$$\{ a^n b^n c^m \mid n, m \geq 0 \} \cap \{ a^m b^n c^n \mid n, m \geq 0 \} = \{ a^n b^n c^n \mid n \geq 0 \}$$

CFL's

not a CFL !

We prove (by contradiction) that ***complementation*** *does not necessarily preserve the 'context free'ness property using De Morgan's formula :*

$$A \cap B = (A^c \cup B^c)^c$$

Measuring Complexities

For $\boldsymbol{G = (V , \Sigma , R , S)}$ *measure of size is :*

$n := |V| + |\Sigma| + |R|.K$, *hence :* $O(|V| + |\Sigma| + |R|.K) = O(n)$

where $\boldsymbol{K}$ *is the maximum of length among all productions.*

For $\boldsymbol{P = (Q , \Sigma, \Gamma, \delta, q_0 , Z_0 , F)}$ *measure of size is :*

$n := |Q| + |\Sigma| + |\Gamma| + |\delta| K$, *hence :* $O(|Q| + |\Sigma| + |\Gamma| + |\delta| K) = O(n)$

where $\boldsymbol{K}$ *is the maximum of length among all transitions.*

Conversion from ***G*** *to* ***P*** *is* **:**

$$\textbf{Size of } P = O(\underbrace{|\Sigma|}_{\text{inputs}} + \underbrace{(|\Sigma| + |V|)}_{\text{stack}} + \underbrace{|\Sigma|}_{\text{input trans.}} + \underbrace{|R|.K}_{\text{stack transitions}}) = O(n)$$

Conversion from ***P*** *to* ***G*** *is as follows : for each transition*

$(p\, , Y_1\, Y_2 \ldots Y_k\,) \in \delta\,(q, a, X)$ *there are productions*

$[q\, X\, q_k] \rightarrow a[p\, Y_1\, q_1] \ldots [q_{k-1}\, Y_k\;\; q_k]$ *for all* $q_1 , \ldots , q_k$ *in* Q

which sum up to $O(n^K) = O(n^n)$ *where* $|Q| = O(n)$ *and*

$K := \max$ *{length of all transitions}* $= O(n)$ *!*

This is exponential in ***n*** *!*

But there is a solution :

Decompose each $(p, Y_1\, Y_2 \ldots Y_k\,) \in \delta\,(\,q\,,a\,, X\,)$ *as* ***k-1*** *transitions :*

$\delta\,(q, a, X) = \{(p_{k-1}\, , Y_{k-1}\, Y_k\,)\} \rightarrow$ *push* $Y_{k-1}\, Y_k$ *,*

$\delta\,(p_{k-1}\, , e, Y_{k-1}) = \{(p_{k-2}\, , Y_{k-2}\, Y_{k-1})\} \rightarrow$ *push* $Y_{k-2}\, Y_{k-1}$ *, ...*

$\delta(p_2\, , e\, , Y_2) = \{(p\, , Y_1\, Y_2)\}$ *then* $K = 2$ *and* $|\delta|.K \rightarrow |\delta|\, .O(K) = O(n)$ *,*

$|Q| \rightarrow |Q|+|\delta|\; O(K) = O(n)$ *hence total complexity is* $O(n^3)$

Complexity of conversion to Chomsky Normal Form

(1) Elimination of non-generating and non-reachable symbols ***O(n^3)***

(2) Eliminating the unit productions ***O(n^3) ;*** *effective size of new grammar* ***O(n) : Why ?***

(3)Elimination of the e-productions ((i) production size ≤ 2, (ii) eliminate ***O(n)***

(4) Replacement of terminals by variables : ***O(n)*** *; size of new grammar* ***O(n)***

(5) Breaking of bodies of size ***> 2*** *into* ***2 : O(n) ;*** *size of new grammar* ***O(n)***

Result : Computational complexity of CNF reduction is O(n^3)

Is $u = a_1\, a_2 \ldots a_n$ a member of L_G?

$X_{1\,n}$

$X_{1\,n-1}\; X_{2\,n}$

$X_{1\,n-2}\; X_{2\,n-1}\; X_{3\,n}$

$X_{p\,m}$

$X_{p\,m-1}\, X_{p+1\,m}$

$X_{p\,k}$ $X_{k+1\,m}$

$X_{12}\; X_{23} \ldots$ $X_{n-1\,n}$

$X_{11}\; X_{22} \ldots X_{p\,p}$ $X_{m\,m} \ldots X_{n-1\,n-1}\; X_{n\,n}$

$a_1\, a_2 \ldots\ldots$ a_n

*$X_{k\,m}$ is in row **m-k+1** and corresponds to substrings of **u** of length **m-k+1***

$$X_{k\,m} = \{A \in V \mid A \Rightarrow_G^* a_k\, a_{k+1} \ldots a_m\}$$

$O(n^3)$ computations

***$O(n^2)$** entries of table, each has **$O(n)$** pairs to compute, for each pair **$O(1)$** effort!*

Divide $X_{p\,m}$ as $X_{p\,k}$ and $X_{k+1\,m}$ and check for $A \rightarrow BC$ where $B \in X_{p\,k}$ and $C \in X_{k+1\,m}$

Example

$S \rightarrow aSb \mid e$

CNF :

$S \rightarrow AC \mid AB$, $C \rightarrow SB$, $A \rightarrow a$, $B \rightarrow b$

Is **aabb** in L_G ?

$X_{11} = X_{22} = \{A\}$; $X_{33} = X_{44} = \{B\}$ using $A \rightarrow a$, $B \rightarrow b$

$X_{12} = X_{34} = \varnothing$; (X_{22} , X_{33}) generated by $S \rightarrow AB$ hence $X_{23} = \{ S \}$

$X_{13} = \varnothing$; (X_{23} , X_{44}) generated by $C \rightarrow SB$ hence $X_{24} = \{ C \}$

(X_{11} , X_{24}) generated by $S \rightarrow AC$ hence $X_{14} = \{ S \}$

Hence $S \Rightarrow^* aabb$

Determinism in PDA and Parsing

Simple Example for top down look – ahead parser

$S \rightarrow a\,S\,b \mid e$ *leads to PDA below:*

$\delta\,(s\,,\,e\,,\,Z_0\,) = \{\,(s,\,SZ_0\,),\,(\,f\,,\,Z_0\,)\,\}$

$\delta\,(s,\,x\,,x\,) = \{\,(s\,,\,e\,)\}$ for $x = a$ and $x = b$

$\delta\,(s\,,\,e\,,\,S) = \{\,(\,s\,,\,aSb\,)\,,\,(s\,,\,e\,)\,\}$

Non-determinism !!

$\delta(s\,,\,e\,,\,Z_0\,) = \{(f\,,\,SZ_1\,)\}$ *accepts* ***e=empty string***

$\delta(f\,,\,a\,,\,S) = \{(q_a\,,\,S)\}$

$\delta(q_a\,,\,e\,,\,S) = \{(s\,,\,Sb)\}$

$\delta(s\,,\,a\,,\,S\,) = \{(q_a\,,\,S\,)\}$

Look-ahead for input a

$\delta(s\,,\,b\,,\,S\,) = \{(q_b,\,S\,)\}$

$\delta(q_b\,,\,e\,,\,S) = \{(q_b\,,\,e)\}$

$\delta\,(q_b\,,\,e\,,\,b) = \{(\,s\,,\,e\,)\}$

Look-ahead for input b

→ ***DPDA***

$\delta(s\,,\,b,\,b\,) = \{(s,\,e)\}$

$\delta(s\,,\,e\,,\,Z_1\,) = \{(f\,,\,Z_0\,)\}$

(1) *Top down parsing*

Given a grammar ***G*** *we elaborate on the productions before we apply a modified version of the PDA given in the proof of the theorem : from* ***G*** *to* ***PDA***

(i) *If* $A \rightarrow c\ \alpha_1 \mid \ldots \mid c\ \alpha_n$ *is a collection of productions of* ***G*** *where* ***c*** *is a terminal or a nonterminal then replace these productions by :*

$A \rightarrow cA'$ *and* $A' \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ *where* ***A'*** *is a new variable.*

The resulting grammar G_1 *yields the same language as* ***G***

(ii) *(Left recursion) If* $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_n$ *and* $A \rightarrow \beta_1 \mid \ldots \mid \beta_m$ *are productions where* $n,m > 0$ *and first element of each* β_i *is different from* ***A*** *then replace these productions by :*

$A \rightarrow \beta_1 B \mid \ldots \mid \beta_m B$ *and* $B \rightarrow \alpha_1 B \mid \ldots \mid \alpha_n B \mid e$.

The resulting grammar G_2 *yields the same language as* ***G***

These arguments lead us to the Greibach Normal Form (GNF):

Each production is of the type $A \rightarrow a\ \alpha$ *where* ***a*** *is a terminal.*

Apply case ***(i)*** *above and if necessary* ***GNF*** *repeatedly until for each production group* $A \rightarrow a_1\ \alpha_1 \mid \ldots \mid a_m\ \alpha_m$ *the terminals* a_j *are all* ***distinct ;***

and so the ***lookahead*** *technique of* ***top down parsing*** *can be applied via a* ***DPDA*** *in the manner demonstrated by the example* $L_G = \{a^n b^n\}$ *done in class*

Example

CFG is $G = (V, T, R, E)$ *where*

$V = \{E, T, F, I\}$; $T = \{+, *, (,), x, y, z\}$ (x, y, z *are either variables or function letters)*

R :

$E \rightarrow E + T \mid T$; $T \rightarrow T * F \mid F$; $F \rightarrow I \mid (E) \mid I(E)$; $I \rightarrow x \mid y \mid z$

PDA : $P = (\{q_0, s, f\}, T, V \cup T \cup \{Z_0\}, \delta, q_0, Z_0, \{s, f\})$

$\delta(q_0, e, Z_0) = \{(s, S Z_0)\}$

$\delta(s, t, t) = \{(s, e)\}$ *for all* $t \in T$

$\delta(s, e, E) = \{(s, E+T), (s, T)\}$

$\delta(s, e, T) = \{(s, T*F), (s, F)\}$

$\delta(s, e, F) = \{(s, I), (s, (E)), (s, I(E))\}$

$\delta(s, e, I) = \{(s, x), (s, y), (s, z)\}$

$\delta(s, e, Z_0) = \{(f, Z_0)\}$

(1) Fix left recursion :

replace $E \rightarrow E + T \mid T$ by $E \rightarrow TB$; $B \rightarrow +TB \mid e$

replace $T \rightarrow T * F \mid F$ by $T \rightarrow FC$; $C \rightarrow *FC \mid e$

(2) Fix common production start symbol :

replace $F \rightarrow I \mid (E) \mid I(E)$ by $F \rightarrow IA \mid (E)$ and $A \rightarrow (E) \mid e$

(3) Substitute until GNF-like structure prevails !! No need for **I** at the end

$E \rightarrow$ x-y-z $ACB \mid (E)CB$

$B \rightarrow +TB \mid e$

$T \rightarrow$ x-y-z $AC \mid (E)C$

$C \rightarrow *FC \mid e$

$F \rightarrow$ x-y-z $A \mid (E)$

$A \rightarrow (E) \mid e$

extra 7 states required

look-ahead works for this GNF **!**

The (DPDA ?) for the grammar G defined above !

$E \rightarrow$ x-y-z $ACB \mid (E)\,CB$

$B \rightarrow + TB \mid e$

$T \rightarrow$ x-y-z $AC \mid (E)\,C$

$C \rightarrow * F C \mid e$

$F \rightarrow$ x-y-z $A \mid (E)$

$A \rightarrow (E) \mid e$

$(q, e, Z_0) \rightarrow (s, EZ_0)$

$(s, + - * - (-) - x\text{-}y\text{-}z, V) \rightarrow (q_+ - q_* - q_(- q_) - q_x\text{-}q_y\text{-}q_z, V)$

$(q_+, e, C\text{-}A) \rightarrow (q_+, e)$

$(q_+, e, B) \rightarrow (s, TB)$

$(q_*, e, A\text{-}B) \rightarrow (q_*, e)$

$(q_*, e, C) \rightarrow (s, FC)$

$(q_(, e, C\text{-}B) \rightarrow (q_(, e)$

$(q_(, e, A\text{-}F\text{-}T\text{-}E) \rightarrow (s, E) - E) - E)C - E)CB)$

$(q_), e, C\text{-}A\text{-}B) \rightarrow (q_), e)$

$(q_x, e, E\text{-}T\text{-}F) \rightarrow (s, ACB - AC - A)$

$(q_y, e, E\text{-}T\text{-}F) \rightarrow (s, ACB - AC - A)$

$(q_z, e, E\text{-}T\text{-}F) \rightarrow (s, ACB - AC - A)$

$(s, input, input) \rightarrow (s, e)$

$(q_{input}, e, input) \rightarrow (s, e)$

$(s, e, A\text{-}C\text{-}B) \rightarrow (s, e)$

$(s, e, Z_0) \rightarrow (f, Z_0)$

V = any non-terminal variable

non-deterministic transitions

This transition is not used in the following example

Parse : x+(y*z(x) + x)

(s , x+(y*z(x) + x) , E Z_0) |-- (q_x ,+(y*z(x) + x) , E Z_0)

|-- (s ,+(y*z(x) + x) , ACB Z_0) |-- (q_+, (y*z(x) + x) , ACB Z_0)

|-- (q_+ , (y*z(x) + x) , CB Z_0) |-- (q_+ , (y*z(x) + x) , B Z_0)

|-- (s , (y*z(x) + x) ,TB Z_0) |-- ($q_($, y*z(x) + x) ,TB Z_0)

|-- (s , y*z(x) + x) , E)CB Z_0) |-- (q_y , *z(x) + x) , E)CB Z_0)

|-- (s ,*z(x) + x) , ACB) C B Z_0) |-- (q_* , z(x) + x) , ACB) C B Z_0)

|-- (q_* , z(x) + x) , CB) C B Z_0) |-- (s ,z(x) + x) , FCB) C B Z_0)

|-- (q_z , (x) + x) , FCB) C B Z_0) |-- (s ,(x) + x) , ACB) C B Z_0)

|-- ($q_($, x) + x) , ACB) C B Z_0) |-- (s , x) + x) , E)CB) C B Z_0)

|-- (q_x ,) + x) , E)CB) C B Z_0) |-- (s ,) + x) , ACB)CB) C B Z_0)

|-- ($q_)$, + x) , ACB)CB) C B Z_0) |-- ($q_)$, + x) , CB) C B) C B Z_0) |--

. . . |-- ($q_)$, + x) ,) C B) C B Z_0)

|-- (s, + x) , C B) C B Z_0) |-- (q_+ , x) , CB) C B Z_0)

|-- (q_+, x) , B) C B Z_0) |-- (s , x) , TB) C B Z_0) |-- (q_x ,) , TB) C B Z_0)

|-- (s ,) , ACB) C B Z_0) |-- ($q_)$, e , ACB)CB Z_0) . . . |-- (s , e , CBZ_0) |-- . . .

(s , e , Z_0) |-- (s , f , Z_0) **TOMBALA !!!!!!!**

E → x-y-z ACB | (E) CB

B → + TB | e

T → x-y-z AC | (E) C

C → * F C | e

F → x-y-z A | (E)

A → (E) | e

(2) *Bottom up Parsing*

First we generalize the new transition function as :

$\delta_N : Q \times (\Sigma \cup e) \times \Phi \rightarrow 2^{Q \times \Gamma^*}$ *where* Φ *is a finite subset of* Γ^*

This definition implies we can pop more than one symbol (or no symbol!) at the top of the stack and replace them as a function of what is popped.

It can easily be shown that this new ***PDA*** *is equivalent to a conventional version given below :*

For each $(p\,,\beta) \in \delta_N(q\,,a\,,X_1 X_2 \ldots X_n)$ *introduce distinct and new intermediate states* $r_1 ,\ldots, r_{n-1} \in Q$ *such that :*

$(r_1\,, e) \in \delta(q\,, e\,, X_1)\,, \ldots, (r_{n-1}\,, e) \in \delta(r_{n-2}\,, e\,, X_{n-1})\,, (p\,, \beta) \in \delta(r_{n-1}\,, a\,, X_n)$

where δ *depicts the transition function of the conventional equivalent system*

Now for a given grammar $\boldsymbol{G = (V, \Sigma, R, S)}$ *define a PDA with the following transition function :*

$(q, a) \in \delta_N(q, a, e)$, $\forall a \in \Sigma$ ***(input shifting)***

$(q, A) \in \delta_N(q, e, \beta)$ *whenever* $A \rightarrow \beta^R$ *in* $\boldsymbol{R}$ *of* $\boldsymbol{G}$ ***(stack reduction)***

$(f, e) \in \delta_N(q, e, S)$ *final acceptance move !*

Theorem

Let the PDA **P** *be defined as above for the grammar* **G** *then*

If $(q, x, \gamma) \vdash_P^* (q, e, S)$ ***then*** $S \Rightarrow_{rm}^* \gamma^R x$

conversely

If $S \Rightarrow_{rm}^* \gamma^R x$ ***and*** $\gamma = \{e\} \cup V.(V \cup T)^*$ ***then***

$(q, x, \gamma) \vdash_P^* (q, e, S)$

(Special case : $\gamma = e$ ***)***

Proof

(⇒) induction on PDA computation steps

(⇐) induction on grammar derivation steps

Example for Bottom Up Parsing

$E \rightarrow E+T \mid T+E$.... (1)
$E \rightarrow T$ .... (2)
$T \rightarrow T*F \mid F*T$...... (3)
$T \rightarrow F$ .... (4)
$F \rightarrow I$ ....(5)
$F \rightarrow I\ (\ E\ )$ ....(6)
$F \rightarrow (\ E\ )$ .... (7)
$I \rightarrow x$ .... (8)
$I \rightarrow y$ .... (9)
$I \rightarrow z$ .... (10)

Input	Stack	Rule
*x+(y*z(x) +x)*	*e*	*input* *8+5+4+2*
*+(y*z(x) +x)*	*E*	*input*
**z(x) +x)*	*y(+E*	*9+5+4*
**z(x) +x)*	*T(+E*	*input+10*
(x) +x)	*I*T(+E*	*input+8+5* *+4+2*
)+x)	*E(I*T(+E*	*input 6+3* *input*
)	*x+T(+E*	*8+5+4+2+1* *input*
e	*)E(+E*	*7+4+1*
e	*E*	

Consider the PDA $P= (Q,\Sigma,\Gamma,\delta ,Z_0 ,q_0 ,F)$ and the DFA $A = (Q_A ,\Sigma ,\delta_A ,q_{0A} ,F_A)$.
We define the product PDA $P\times A := (Q\times Q_A ,\Sigma ,\Gamma ,\delta_{P\times A} , Z_0 ,(q_0 , q_{0A}) ,F\times F_A)$
where the product transition function $\delta_{P\times A}$ is defined as :
$((q', r'), \gamma) \in \delta_{P\times A} ((q, r) ,a ,X)$ iff :
(i) $(q', \gamma) \in \delta(q, a, X) \wedge r'= \delta_A(r, a)$, if $a\in\Sigma , X\in\Gamma$;
(ii) $(q', \gamma) \in \delta(q, a, X) \wedge r'=r$, if $a=e$, $X\in\Gamma$ Prove that $L(P\times A) = L(P)\cap L(A)$

1- Show by induction on the length of string s

$((q, r), s , \alpha) \vdash^*_{(P\times A)} ((p, t),e , \theta)$ for $s \in\Sigma^*; \alpha, \theta \in\Gamma^*$ iff :
(i) $(q, s, \alpha) \vdash^*_P (p, e , \theta)$
(ii) $t = \delta_A E (r , s)$
where $\delta_A E$ is the extended transition function of A.

For $s=e$ let the top element of α be X, and use the **definition** above with $a=e$

$((q', r'), \gamma) \in \delta_{P\times A} ((q, r) ,e ,X)$ iff

$(q', \gamma) \in \delta(q, e, X) \wedge r'=r$

But $((q', r'), \gamma) \in \delta_{P\times A} ((q, r) ,e ,X)$ iff $((q, r), e, X) \vdash_{(P\times A)} ((q', r'),e ,\gamma)$ iff

$(q', \gamma) \in \delta(q, e, X) \wedge r'=r$ iff (i) $(q, e, X) \vdash_P (q', e ,\gamma)$
(ii) $r' = \delta_A E (r , e) =r$ which proves 1- for $a=e$

Now let $\boldsymbol{s = u.a}$ *where* $\boldsymbol{u \in \Sigma^*}$ *and* $\boldsymbol{a \in \Sigma}$

By induction hypothesis we have

$((q, r), u, \alpha) \mid\!\text{--}^*_{(P\times A)} ((v, w), e, \beta)$ *for* $u \in \Sigma^*; \alpha, \beta \in \Gamma^*$ *iff* : . . . . (*)

(i) $(q, u, \alpha) \mid\!\text{--}^*_{P} (v, e, \beta)$

(ii) $w = \delta_A E\,(r, u)$ . . . . (**)

where $\boldsymbol{\delta_A E}$ *is the extended transition function of* $\boldsymbol{A}$.

Now let $\boldsymbol{Y}$ *be top element of* $\boldsymbol{\beta}$ *that is,* $\boldsymbol{\beta = Y \beta'}$*, then again by* ***definition***

$((p, t), \eta) \in \delta_{P\times A}((v, w), a, Y)$ ***iff*** :

$(p, \eta) \in \delta(v, a, Y) \wedge t = \delta_A(w, a)$

But $((p, t), \eta) \in \delta_{P\times A}((v, w), a, Y)$ *iff* $((v, w), a, Y) \mid\!\text{--}_{P\times A)} ((p, t), e, \eta)$. . . (*)

and $(p, \eta) \in \delta(v, a, Y) \wedge t = \delta_A(w, a)$ *iff* $(v, a, Y) \mid\!\text{--}_{P} (p, e, \eta) \wedge t = \delta_A(w, a)$ …(**)

Hence combining two (*) and (**) formulas

$((q, r), u.a, \alpha) \mid\!\!-\!-^*_{(P\times A)} ((v, w), a, Y\beta') \mid\!\!-\!-_{(P\times A)} ((p, t), e, \eta\beta')$ *iff*

$(q, u.a, \alpha) \mid\!\!-\!-^*_P (v, a, Y\beta') \mid\!\!-\!-_P (p, e, \eta\beta') \wedge w = \delta_A E(r, u) \wedge t = \delta_A(w, a)$

$((q, r), s, \alpha) \mid\!\!-\!-^*_{(P\times A)} ((p, t), e, \theta)$ (where $\theta := \eta\beta'$) *iff*

$(q, s, \alpha) \mid\!\!-\!-^*_P (p, e, \theta) \wedge t = \delta_A E(r, s)$ ***which completes the proof***

Hence

$((q, r), s, \alpha) \mid\!\!-\!-^*_{(P\times A)} ((p, t), e, \theta)$ *for* $s \in \Sigma^*$; $\alpha, \theta \in \Gamma^*$ **iff** :

(i) $(q, s, \alpha) \mid\!\!-\!-^*_P (p, e, \theta)$

(ii) $t = \delta_A E(r, s)$

where $\delta_A E$ *is the extended transition function of* A.

We apply this result below

$((q_0, q_{0A}), s, Z_0) \mid\!\!-\!-^*_{(P\times A)} ((p, t), e, \theta) \wedge (p, t) \in F_P \times F_A$ ***iff*** $s \in L(P \times A)$ ***iff***

(i) $(q_0, s, Z_0) \mid\!\!-\!-^*_P (p, e, \theta) \wedge p \in F_P \wedge$ (ii) $t = \delta_A E(q_{0A}, s) \wedge t \in F_A$ ***iff***

(i) $s \in L(P) \wedge$ **(ii)** $s \in L(A)$ ***iff***

$s \in L(P) \cap L(A)$