COMP2270/6270 – Theory of Computation
Seventh week

School of Electrical Engineering & Computing
The University of Newcastle

Note: Some exercises belong to Chapters 11 and 12 of Ref [1]

Exercise 1) Give an example of a language that is not regular but that it can be generated by a context-free grammar. Can you give another three examples? Justify your answers.

> Example of three languages which are context free but not regular are: A^nB^n, WW^R, and Balanced Parenthesis.

Exercise 2) Let *G* be the ambiguous expression grammar given below (Example 11.14 of Ref. [1]). Show at least three different parse trees that can be generated from *G* for the string `id+id*id*id`.

G = {{E, id, +, *, (,)}, { id, +, *, (,) }, R, E }, where:

R = { E → E + E
E → E * E
E → (E)
E → id
}

Exercise 3) Consider the expression grammar G′ given below (Example 11.19 of Ref. [1]).

G′ = {{E, T, F, id, +, *, (,)}, { id, +, *, (,) }, R, E }, where:

R = { E → E + T
E → T
T → T * F
T → F
F → (E)
F → id
}

a) Trace a derivation of the string id+id*id*id in *G*□.

> $E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow \texttt{id} + T \Rightarrow \texttt{id} + T * F \Rightarrow \texttt{id} + T * F * F \Rightarrow \texttt{id} + F * F * F \Rightarrow$
> $\texttt{id} + \texttt{id} * F * F \Rightarrow \texttt{id} + \texttt{id} * \texttt{id} * F \Rightarrow \texttt{id} + \texttt{id} * \texttt{id} * \texttt{id}$

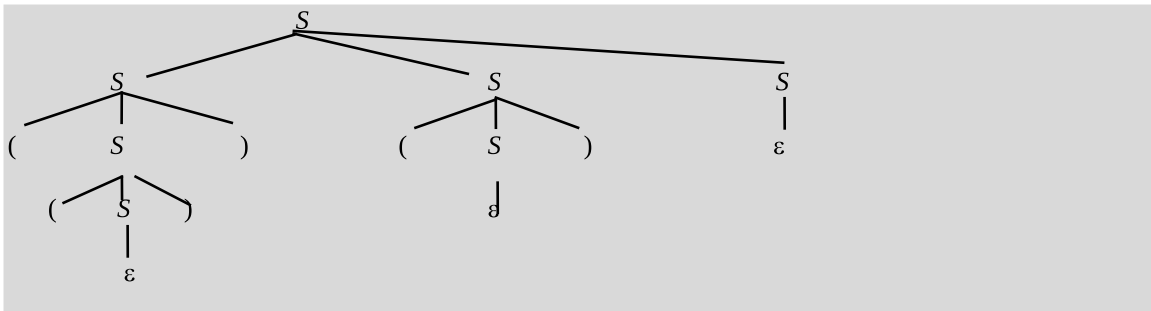
b) Add exponentiation (∗∗) and unary minus (-) to *G′*, assigning the highest precedence to unary minus, followed by exponentiation, multiplication, and addition, in that order.

> $R = \{ E \to E + T$
> $E \to T$
> $T \to T * F$
> $T \to F$
> $F \to F ** X$
> $F \to X$
> $X \to -X$
> $X \to Y$
> $Y \to (E)$
> $Y \to \texttt{id}$ }.

Exercise 4) Let *G* be the grammar given below (Example 11.12 of Ref [1]). Show a third parse tree that *G* can produce for the string (())().

G = {{S,), (}, {), (}, R, S }, where:

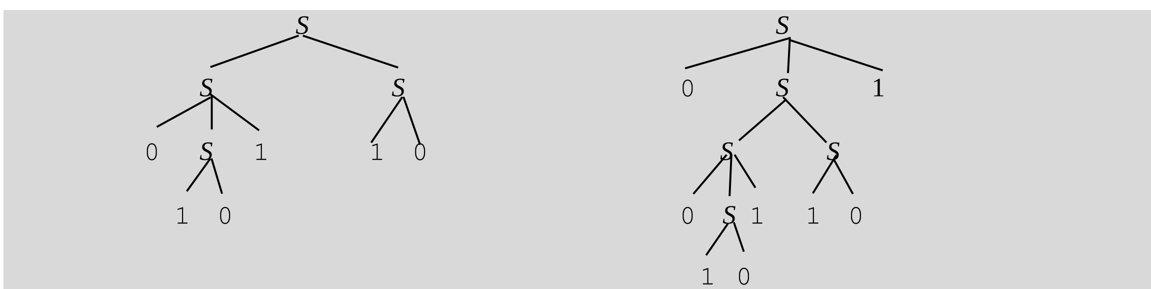
R = { S → (S)
S → SS
S → ε
}



Exercise 5) Consider the following grammar *G*: $S \rightarrow 0S1 \mid SS \mid 10$

Show a parse tree produced by *G* for each of the following strings:

a) `010110`
b) `00101101`



Exercise 6) Convert each of the following grammars to Chomsky Normal Form:

a) $S \rightarrow ABC$
$A \rightarrow \mathtt{a}C \mid D$
$B \rightarrow \mathtt{b}B \mid \varepsilon \mid A$
$C \rightarrow A\mathtt{c} \mid \varepsilon \mid C\mathtt{c}$
$D \rightarrow \mathtt{aa}$

$S \rightarrow AS_1$
$S_1 \rightarrow BC$
$S \rightarrow AC$
$S \rightarrow AB$
$S \rightarrow X_aC \mid \texttt{a} \mid X_aX_a$
$A \rightarrow X_aC$
$A \rightarrow \texttt{a}$
$A \rightarrow X_aX_a$
$B \rightarrow X_bB$
$B \rightarrow \texttt{b}$
$B \rightarrow X_aC \mid \texttt{a} \mid X_aX_a$

$C \rightarrow AX_c$
$C \rightarrow CX_c$
$C \rightarrow \texttt{c}$
$D \rightarrow X_aX_a$
$X_a \rightarrow \texttt{a}$
$X_b \rightarrow \texttt{b}$
$X_c \rightarrow \texttt{c}$

b) $S \rightarrow aTVa$
$T \rightarrow aTa \mid bTb \mid \varepsilon \mid V$
$V \rightarrow cVc \mid \varepsilon$

S → aTVa

T → aTa | bTb | ε | V

V → cVc | ε

removeEps:

Set of *Nullable* Variables *N* : {T,V}

New rules in bold, the $T \rightarrow \varepsilon$ and $V \rightarrow \varepsilon$ rules have been removed.

S → aTVa | **aTa** | **aVa** | **aa**

T → aTa | bTb | V | **aa** | **bb**

V → cVc | cc

At this point all epsilon transitions have been removed.

removeUnits:

New rules in bold, the $T \rightarrow V$ rule have been removed.

S → aTVa | aTa | aVa | aa

T → aTa | bTb | aa | bb | **cVc** | **cc**

V → cV c | cc

At this point all rules with a RHS of length one are in Chomsky Normal Form (i.e. they are a single terminal symbol).

removeMixed:

New and modified rules in bold. An important note is that rules such as T → aa must still be changed even though they are technically not "mixed", any rule with a RHS longer than 1 symbol and which contains a terminal must be modified by replacing the terminal with its new nonterminal equivalent.

S → **ATVA** | **ATA** | **AVA** | **AA**

T → **ATA** | **BTB** | **AA** | **BB** | **CVC** | **CC**

V → **CVC** | **CC**

A → a

B → b

C → c

At this point all rules who have a RHS of length one or two are in CNF.

removeLong:

$S \rightarrow \mathbf{AS_1} \mid \mathbf{AS_2} \mid \mathbf{AS_3} \mid AA$

$S_1 \rightarrow \mathbf{TS_3}$

$S_2 \rightarrow \mathbf{TA}$

$S_3 \rightarrow \mathbf{VA}$

$T \rightarrow \mathbf{AS_2} \mid \mathbf{BT_1} \mid AA \mid BB \mid \mathbf{CT_2} \mid CC$

$T_1 \rightarrow \mathbf{TB}$

$T_2 \rightarrow \mathbf{VC}$

$V \rightarrow \mathbf{CT_2} \mid CC$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

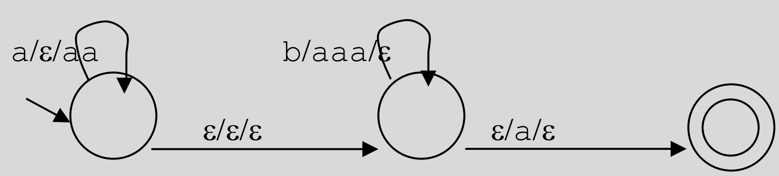
At this point the grammar is now in CNF and still describes the same language, though it will not necessarily generate the same parse tree for each string.

Exercise 7) Build a PDA to accept each of the following languages *L*:

BalDelim = {*w* : where *w* is a string of delimeters: (,), [,], {, }, that are properly balanced}.

M = ({1}, {(,), [,], {, }}, {(, [, {}, Δ, 1, {1}), where Δ =
{ ((1, (, ε), (1, ()),
((1, [, ε), (1, [)),
((1, {, ε), (1, {)),
((1,), (), (1, ε)),
((1,], [), (1, ε)),
((1, }, {), (1, ε)) }

a) $\{a^i b^j : 2i = 3j + 1\}$.



b) $\{w \in \{a, b\}^* : \#_a(w) = 2 \cdot \#_b(w)\}$.

The idea is that we only need one state. The stack will do all the work. It will count whatever it is ahead on. Since one a matches two b's, each a will push an a (if the machine is counting a's) and each b (if the machine is counting a's) will pop two of them. If, on the other hand, the machine is counting b's, each b will push two b's and each a will pop one. The only tricky case arises with inputs like aba. *M* will start out counting a's and so it will push one onto the stack. Then comes a b. It wants to pop two a's, but there's only one. So it will pop that one and then switch to counting b's by pushing a single b by using the last transition listed in Δ. The final a will then pop that b. *M* is highly nondeterministic. But there will be an accepting path iff the input string *w* is in *L*.

$M = (\{1\}, \{a, b\}, \{a, b\}, \Delta, 1, \{1\})$, where $\Delta =$
{ ((1, a, ε), (1, a)),
((1, a, b), (1, ε)),
((1, b, ε), (1, bb)),
((1, b, aa), (1, ε)),
((1, b, a), (1, b)) }

c) $\{a^n b^m : m \le n \le 2m\}$.

$M = (\{1, 2\}, \{a, b\}, \{a\}, \Delta, 1, \{1, 2\})$, where $\Delta =$
{ ((1, a, ε), (1, a)),
((1, ε, ε), (2, ε)),
((2, b, a), (2, ε)),
((2, b, aa), (2, ε)) }.

d) $\{w \in \{a, b\}^* : w = w^R\}$.

This language includes all the even-length palindromes of Example 12.5, plus the odd-length palindromes. So a PDA to accept it has a start state we'll call 1. There is a transition, from 1, labelled ε/ε/ε, to a copy of the PDA of Example 12.5. There is also a similarly labelled transition from 1 to a machine that is identical to the machine of Example 12.5 except that the transition from state *s* to state *f* has the following two labels: a/ε/ε and b/ε/ε. If an input string has a middle character, that character will drive the new machine through that transition. A PDA to accept this language can also be derived from the machine in Ex 12.5 by adding the a/ε/ε and b/ε/ε transitions in parallel to the ε/ε/ε transition. In both cases the automata is non-deterministic.

REFERENCES
[1] Elaine Rich, Automata Computatibility and Complexity: Theory and Applications, Pearson, Prentice Hall, 2008.