COMP2270/6270 – Theory of Computation
Twelfth Week

School of Electrical Engineering & Computing
The University of Newcastle

Exercise 1) Show each of the following grammars are ambiguous:

a) ({S,A,B,T,a,c}, {a,c}, R, S)
R = {S → AB, S → BA, A → aA, A → ac, B → Tc, T → aT, T → a}

Two parse trees can generate the string 'acac'. The nonterminals A and B can both produce 'ac' but S can generate AB or BA.

b) ({S,a,b}, {a,b}, R, S)
R = {S → ε, S → aSa, S → bSb, S → aSb, S → bSa, S → SS}

Any string can have infinite parse trees due to the S → SS and S → ε rules.

Exercise 2) What is meant by *inherent ambiguity* for context-free languages? Is it a concept that applies to grammars or languages?

A CF Language that is 'inherently ambiguous' does not have ANY non-ambiguous grammar that generates it. ***Inherent*** *ambiguity, as a concept, applies to languages not grammars, grammars can be ambiguous even if the language it describes is not inherently ambiguous.*

Exercise 3) Convert the following CFG to Chomsky Normal Form:

$S \rightarrow$ aSa
$S \rightarrow$ B
$B \rightarrow$ bbC
$B \rightarrow$ bb
$C \rightarrow \epsilon$
$C \rightarrow$ cC

RemoveEps:

$S \rightarrow aSa$
$S \rightarrow B$
$B \rightarrow bbC$
$B \rightarrow bb$
$C \rightarrow cC$
$C \rightarrow c$

RemoveUnits:

$S \rightarrow aSa$
$S \rightarrow bbC$
$S \rightarrow bb$

$$B \rightarrow bbC$$
$$B \rightarrow bb$$
$$C \rightarrow cC$$
$$C \rightarrow c$$

RemoveMixed:

$$S \rightarrow T_a S T_a$$
$$S \rightarrow T_b T_b C$$
$$S \rightarrow T_b T_b$$
$$B \rightarrow T_b T_b C$$
$$B \rightarrow T_b T_b$$
$$C \rightarrow T_c C$$
$$C \rightarrow c$$
$$T_a \rightarrow a$$
$$T_b \rightarrow b$$
$$T_c \rightarrow c$$

RemoveLong:

$$S \rightarrow T_a M_1$$
$$S \rightarrow T_b M_2$$
$$S \rightarrow T_b T_b$$
$$M_1 \rightarrow S T_a$$
$$M_2 \rightarrow T_b C$$
$$B \rightarrow T_b M_3$$
$$B \rightarrow T_b T_b$$
$$M_3 \rightarrow T_b C$$
$$C \rightarrow T_c C$$
$$C \rightarrow c$$
$$T_a \rightarrow a$$
$$T_b \rightarrow b$$
$$T_c \rightarrow c$$

Exercise 4) Show the PDA that cfgtoPDAtopdown will produce on G.

$$G = (\{S, A, B, a, b, c\}, \{a, b, c\}, R, S)$$

where:

$$R = \{S \rightarrow \mathrm{AB}, \mathrm{S} \rightarrow \mathrm{B}, \mathrm{A} \rightarrow \mathrm{aaA}, \mathrm{A} \rightarrow \mathrm{aa}, \mathrm{B} \rightarrow \mathrm{bbB}, \mathrm{B} \rightarrow \mathrm{b}\}$$

$$M = (\{p, q\}, \{a, b, c\}, \{S, A, B, a, b, c\}, \Delta, p, \{q\})$$

where

$$\Delta = \left\{ \begin{array}{c} ((p,\varepsilon,\varepsilon),(q,S)), \\ ((q,\varepsilon,S),(q,AB)), \\ ((q,\varepsilon,S),(q,B)), \\ ((q,\varepsilon,A),(q,aaA)), \\ ((q,\varepsilon,A),(q,aa)), \\ ((q,\varepsilon,B),(q,bbB)), \\ ((q,\varepsilon,B),(q,b)), \\ ((q,a,a),(q,\varepsilon)), \\ ((q,b,b),(q,\varepsilon)), \\ ((q,c,c),(q,\varepsilon)), \end{array} \right\}$$

With the initial transition to put S on the stack, the rule expansion transitions that replace LHS non terminals with the RHS of every rule in R, and the character matching transitions that compare the top of the stack with the next character in the string.

Exercise 5) Use closure properties to prove the following true or false (you can assume membership of other well-known languages (i.e. $a^n b^n \in CFL$).

a. $L = \{w \in \{a,b\}^* : \#_a(w) = \#_b(w)\}$ is regular.

If L was regular then $L' = L \cap a^*b^*$ *would also be regular (closure under intersection of the regular languages) but* $L' = \{a^n b^n\}$ *which we know is not regular, therefore one of the languages in the intersection must not be regular, we know* a^*b^* *is regular since we can generate it using a regular expression and therefore the only conclusion is that L is not regular. Therefore the statement is false.*

b. $L = \{a^i b^k : i \neq k\}$ is regular.

If L was regular then:

$$L' = \neg L = \{a^n b^n\} \cup \{\text{all strings with some } 'a's \text{ after } 'b's\}$$

would also be regular (closure under compliment of the regular languages) and therefore

$$L'' = L' \cap a^*b^* = \neg L \cap a^*b^*$$

would be regular (closure under intersection of the regular languages). Now

$$L'' = \{a^n b^n\}$$

which we know is not regular therefore one of the two languages that we intersected to form it must not be regular, we know (as before) that a^*b^* *is regular, therefore* $\neg L$ *must not be regular, since* $\neg L$ *is not regular, L cannot be regular. Therefore the statement is false.*

c. $L = \{a^n b^n : n \neq 10000\}$ is context-free.

L can be expressed as:

$$L = L_1 - L_2 = \{a^n b^n\} - \{a^{10000} b^{10000}\}$$

L_1 is a known context free language and L_2 is regular since it is finite (it only contains one string). Since the set of context free languages is closed under difference with regular languages we know that L is context free. Therefore the statement is true.

d. $L = \{w \in \{a,b,c\}^* : \#_a(w) = \#_b(w) = \#_c(w)\}$ is context-free.

Similar argument to Ex5a. If L were context free then

$$L' = L \cap a^*\mathrm{b}^*\mathrm{c}^*$$

would also be context free (closure of the context free languages under intersection with the regular languages). but

$$L' = \{a^n\mathrm{b}^n\mathrm{c}^n\}$$

which we know is not context free, therefore, since we know $\mathrm{a}^*\mathrm{b}^*\mathrm{c}^*$ *is regular, L is not context free. Therefore the statement is false.*

Exercise 6) What is the difference between a deterministic and a non-deterministic turing machine? Is one more powerful than the other (in terms of the languages they can accept or functions they can compute)? What about single tape and multiple tape TMs?

Exercise 7) What is the difference between a deciding and a semi-deciding turing machine?

Exercise 8) Encode the following TMs as strings as their string representations <M>.

a. $M = (\{s, q, h\}, \{a, b, c\}, \{\square, a, b, c\}, \delta, s, \{h\})$
Where δ is given by:

state	Symbol on tape	δ
s	□	$(q, \square, \rightarrow)$
s	a	$(s, b, \rightarrow)$
s	b	$(q, a, \leftarrow)$
s	c	$(q, b, \leftarrow)$
q	□	$(s, a, \rightarrow)$
q	a	$(q, b, \rightarrow)$
q	b	$(q, b, \leftarrow)$
q	c	$(h, a, \leftarrow)$

Encoding the states:
s = q00 q = q01 h = q10
(the start state MUST be q00, but the other states can be any enumeration)

Encoding the tape alphabet:
□ = a00 a = a01 b = a10 c = a11
(any other enumeration is also valid)

Finally the encoding <M> for M comes from δ.
<M> = (q00, a00, q01, a00, →), (q00, a01, q00, a10, →), (q00, a10, q01, a01, ←), (q00, a11, q01, a10, ←), (q01, a00, q00, a01, →), (q01, a10, q01, a10, →), (q01, a10, q01, a10, ←), (q01, a11, q10, a01, ←)

a. $M = (\{p, q, y, n\}, \{v, x\}, \{\square, v, x, z\}, \delta, p, \{y, n\})$
Where δ is given by:

State	Symbol on tape	δ
p	□	$(q, \square, \rightarrow)$
p	v	$(y, x, \rightarrow)$
p	x	$(y, v, \leftarrow)$
p	z	$(y, z, \leftarrow)$

q	□	$(n, z, \rightarrow)$
q	v	$(n, z, \rightarrow)$
q	x	$(p, z, \leftarrow)$
q	z	$(q, v, \leftarrow)$

Encoding the states:
p = q00 q = q01 y = y10 n = n11

Encoding the tape alphabet:
□ = a00 v = a01 x = a10 z = a11

Encoding M:
<M> = (q00, a00, q01, a00, →), (q00, a01, y10, a10, →), (q00, a10, y10, a01, ←), (q00, a11, y10, a10, ←), (q01, a00, n11, a11, →), (q01, a01, n11, a11, →), (q01, a10, q00, a11, ←), (q01, a11, q01, a01, ←)

Exercise 9) Describe in clear English a Turing Machine M to compute the following functions.

a. A single tape TM to compute addition of two unary numbers. Given the string $\langle x \rangle \# \langle y \rangle$ where $\langle x \rangle$ and $\langle y \rangle$ are the unary encoding of the values x and y the M should output halt with $\underline{\square}\langle z \rangle$ on the tape, where $\langle z \rangle$ is the unary encoding of $z = x + y$.

One option:
1. *Move right.*
2. *If a 1, then blank it, move right until a blank and write a 1, move left until blank, then repeat step 1*
3. *If a # blank it and halt.*

Another option:
Move right, if a blank, halt. If a 1 blank it out, move to the #, replace it with a 1, move left until blank and halt.

b. A multiple tape TM to compute **binary** addition in the same specification as above.

M moves right until encountering the '#' copying whatever it sees to the second tape. M then blanks the '#' and moves to the right of the string on the first tape. M now has each of the inputs on separate tapes and is to the right them on their respective tapes.

M now moves right on both tapes simultaneously, it stores the result of summing the binary digits over the top of the original digit on the first tape and stores the carry in internal state (remember a TM also has states). When it moves onto the next position it includes this internal carry value in the calculations. Whenever it encounters a blank on one tape it considers it a 0. When the TM encounters blanks on both tapes if it is in the internal carry state it writes a 1 moves left and halts, otherwise it just halts.

REFERENCES

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