# COMP2270/6270 – Theory of Computation
## Ninth Week

## School of Electrical Engineering & Computing
## The University of Newcastle

*Note: Some exercises belong to Chapters 13 of Ref [1]*

**Exercise 1)** Consider the following (highly ambiguous) grammar $G$ for Boolean expressions (Don't get confused by the fact that the symbol $\rightarrow$ is both a metasymbol in each rule and a terminal symbol in the grammar $G$.):

$E \rightarrow \neg E$
$E \rightarrow E \vee E$
$E \rightarrow E \wedge E$
$E \rightarrow E \rightarrow E$
$E \rightarrow (E)$
$E \rightarrow \mathtt{id}$

a) Show the PDA that *cfgtoPDAbottomup* will build on input $G$.

$M = (\{p, q\}, \{\mathtt{id}, \neg, \vee, \wedge, \rightarrow, (, )\ \}, \{E, \mathtt{id}, \neg, \vee, \wedge, \rightarrow, (, )\}, \Delta, p, \{q\})$, where $\Delta =$
{ $((p, \mathtt{id}, \varepsilon), (p, \mathtt{id}))$,
$((p, \neg, \varepsilon), (p, \neg))$,
$((p, \vee, \varepsilon), (p, \vee))$,
$((p, \wedge, \varepsilon), (p, \wedge))$,
$((p, \rightarrow, \varepsilon), (p, \rightarrow))$,
$((p, (, \varepsilon), (p, ())$,
$((p, ), \varepsilon), (p, )))$,
$((p, \varepsilon, E\neg), (p, E))$,
$((p, \varepsilon, E \vee E), (p, E))$,
$((p, \varepsilon, E \wedge E), (p, E))$,
$((p, \varepsilon, E \rightarrow E), (p, E))$,
$((p, \varepsilon, )E(), (p, E))$,
$((p, \varepsilon, \mathtt{id}), (p, E))$,
$((p, \varepsilon, E), (q, \varepsilon))$ }

b) Trace the execution of one accepting path of your PDA on the input: $(\neg\mathtt{id} \rightarrow (\mathtt{id} \vee \mathtt{id}))$.

| state | w | stack | Rule type |
|---|---|---|---|
| p | ( ¬ id → ( id ∨ id ) ) | ε | Shift |
| p | ¬ id → ( id ∨ id ) ) | ( | Shift |
| p | id → ( id ∨ id ) ) | ¬ ( | Shift |
| p | → ( id ∨ id ) ) | id ¬ ( | Shift |

| | | | |
|---|---|---|---|
| p | → ( id ∨ id ) ) | E ¬ ( | Reduce |
| p | → ( id ∨ id ) ) | E ( | Reduce |
| p | ( id ∨ id ) ) | → E ( | Shift |
| p | id ∨ id ) ) | ( → E ( | Shift |
| p | ∨ id ) ) | id ( → E ( | Shift |
| p | ∨ id ) ) | E ( → E ( | Reduce |
| p | ) ) | ∨ E ( → E ( | Shift |
| p | ) ) | id ∨ E ( → E ( | Shift |
| p | ) ) | E ∨ E ( → E ( | Reduce |
| p | ) ) | E ( → E ( | Reduce |
| p | ) | ) E ( → E ( | Shift |
| p | ) | E → E ( | Reduce |
| p | ) | E ( | Reduce |
| p | ε | ) E ( | Shift |
| p | ε | E | Reduce |
| q | ε | ε | Accept |

Note that each pair of these rows corresponds to a "yields-in-one-step" statement, for example the following is for the first transition:

$(p,\ (\neg\, id \rightarrow (id \vee id)),\ \varepsilon) \vdash_M (p,\ \neg\, id \rightarrow (id \vee id)),\ (\ )$

The table has been given instead for ease of understanding however you should be aware of yielding notation.

c) Show the PDA that *cfgtoPDAtopdown* will build on input $G$.

$M = (\{p, q\}, \{\texttt{id}, \neg, \vee, \wedge, \rightarrow, (, ), \forall, \exists\}, \{E, \texttt{id}, \neg, \vee, \wedge, \rightarrow, (, ), \forall, \exists\}, \Delta, p, \{q\})$, where $\Delta =$

$\{$ $((p, \varepsilon, \varepsilon), (q, E))$

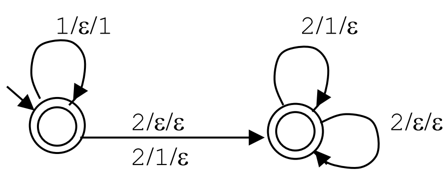
$((q, \varepsilon, E), (q, \neg E))$,
$((q, \varepsilon, E), (q, E \vee E))$,
$((q, \varepsilon, E), (q, E \wedge E))$,
$((q, \varepsilon, E), (q, E \rightarrow E))$,
$((q, \varepsilon, E), (q, (E)))$,
$((q, \varepsilon, E), (q, \texttt{id}))$,

$((q, \texttt{id}, \texttt{id}), (q, \varepsilon))$,
$((q, \neg, \neg), (q, \varepsilon))$,
$((q, \vee, \vee), (q, \varepsilon))$,
$((q, \rightarrow, \rightarrow), (q, \varepsilon))$,
$((q, (, (), (q, \varepsilon))$,
$((q, ), )), (q, \varepsilon))$ $\}$

d) Prove that $G$ is ambiguous.:

It suffices to show a single string to which $G$ assigns at least two parse trees. Consider $\texttt{id} \vee \texttt{id} \wedge \texttt{id}$:


**Exercise 2)** Consider the following PDA $M$:




a) Give a concise description of $L(M)$.

$\{1^n 2^m : 0 \leq n \leq m\}$

b) Show a context-free grammar that generates $L(M)$.

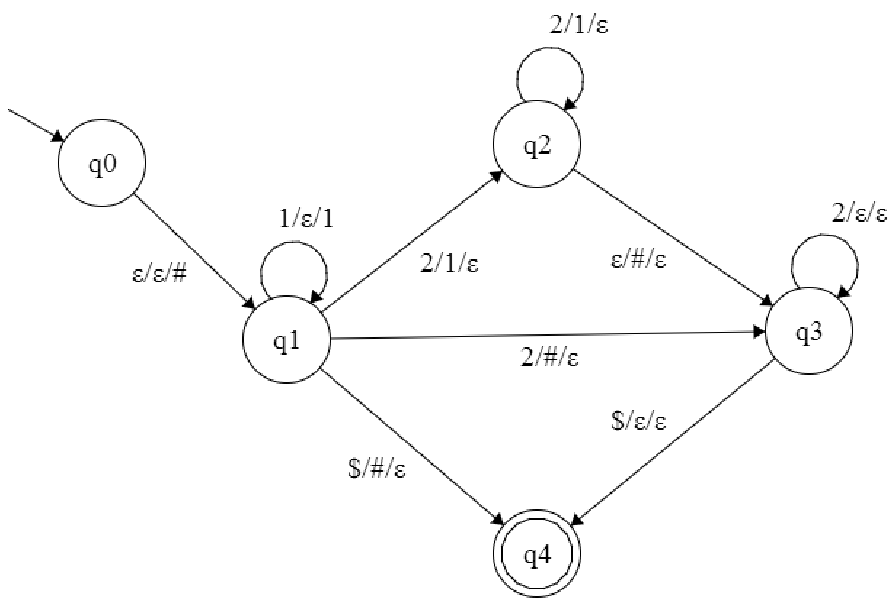
$S \rightarrow 1\ S\ 2$
$S \rightarrow S\ 2$
$S \rightarrow \varepsilon$

c) Is *M* deterministic? Justify your answer.

No. Whenever there is a `1` on the stack and the input symbol is `2`, the two transitions from the start state to the other state compete with each other.

d) Is *L*(*M*) deterministic context-free? Justify your answer.

Yes. There exists a deterministic PDA that accepts *L*(*M*)$. It works similarly to the way *M* works except that, before it begins reading input, it pushes a marker # onto the bottom of the stack. Then it only takes the two transitions that don't pop a `1` if the stack contains no `1`'s.



**Exercise 3)** For each of the following languages *L*, state whether *L* is regular, context-free but not regular, or not context-free and prove your answer.

a) $\{a^i b^j c^k : i, j, k \geq 0 \text{ and } j > i + k\}$.

Context-free, not regular. A grammar for *L* is:

$S \rightarrow TBX$
$T \rightarrow aTb \mid \varepsilon$
$X \rightarrow bXc \mid \varepsilon$
$B \rightarrow bB \mid b$

Not regular, by pumping. Let $w = a^k b^{k+1}$. Set *q* to 2. The resulting string is $a^{k+p} b^{k+1}$. It is not in *L* because there are not more b's than a's.

b) $\{a^i b^j c^k: i, j, k \geq 0 \text{ and } j > max(i, k)\}$.

Not context-free. We prove it using the Pumping Theorem. Let *k* be the constant from the Pumping Theorem and let $w = a^k b^{k+1} c^k$. Let region 1 contain all the a's, region 2 contain all the b's, and region 3 contain all the c's. If either *v* or *y* crosses numbered regions, pump in once. The resulting string will not be in *L* because it will violate the form constraint. We consider the remaining cases:

(1, 1): Pump in once. This increases the number of a's and thus the *max* of the number of a's and c's. But the number of b's is unchanged so it no longer greater than that maximum.
(2, 2): Pump out once. The *max* of the number of a's and c's is unchanged. But the number of b's is decreased and so it is no longer greater than that maximum.
(3, 3): Same argument as (1, 1) but increases the number of c's.
(1, 2).(2, 3): Pump out once. The *max* of the number of a's and c's is unchanged. But the number of b's is decreased and so it is no longer greater than that maximum.
(1, 3): Not possible since $|vxy|$ must be less than or equal to *k*.

c) $\{ww^R w : w \in \{a, b\}^*\}$.

Not context free. We prove it using the Pumping Theorem. Let $w = a^k b^k b^k a^k a^k b^k$.
1 | 2 | 3 | 4

In each of these cases, pump in once:

- If any part of *v* is in region 1, then to produce a string in *L* we must also pump a's into region 3. But we cannot since $|vxy| \leq k$.
- If any part of *v* is in region 2, then to produce a string in *L* we must also pump b's into region 4. But we cannot since $|vxy| \leq k$.
- If any part of *v* is in region 3, then to produce a string in *L* we must also pump a's into region 1. But we cannot since *y* must come after *v*.
- If any part of *v* is in region 4, then to produce a string in *L* we must also pump b's into region 2. But we cannot since y must come after *v*.

d) $\{a^n b^m c^k: n, m, k \geq 0 \text{ and } m \leq min(n, k)\}$.

Not context-free. We prove it using the Pumping Theorem. Let *k* be the constant from the Pumping Theorem and let $w = a^k b^k c^k$. Let region 1 contain all the a's, region 2 contain all the b's, and region 3 contain all the c's. If either *v* or *y* crosses numbered regions, pump in once. The resulting string will not be in *L* because it will violate the form constraint. We consider the remaining cases for where nonempty *v* and *y* can occur:

(1, 1): Pump out once. This reduces the number of a's and thus the *min* of the number of a's and c's. But the number of b's is unchanged so it is greater than that minimum.
(2, 2): Pump in once. The *min* of the number of a's and c's is unchanged. But the number of b's is increased and so it is greater than that minimum.
(3, 3): Same argument as (1, 1) but reduces the number of c's.
(1, 2), (2, 3): Pump in once. The *min* of the number of a's and c's is unchanged. But the number of b's is increased and so it is greater than that minimum.
(1, 3): Not possible since $|vxy|$ must be less than or equal to $k$.

**Exercise 4)** Are the context-free languages closed under each of the following functions? Prove your answer.

a) $chop(L) = \{w : \exists x \in L\ (x = x_1 c x_2 \wedge x_1 \in \Sigma_L^* \wedge x_2 \in \Sigma_L^* \wedge c \in \Sigma_L \wedge |x_1| = |x_2| \wedge w = x_1 x_2)\}$.

Not closed. We prove this by showing a counterexample. Let $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}\mathtt{a}^m\mathtt{b}^m,\ n, m \geq 0\}$. $L$ is context-free.

$chop(L) =$

$\mathtt{a}^n\mathtt{b}^n\mathtt{a}^m\mathtt{b}^m$ (in case, in the original string $n = m$)

$\cup$

$\mathtt{a}^n\mathtt{b}^{n-1}\mathtt{c}\mathtt{a}^m\mathtt{b}^m$ (in case, in the original string, $n > m$)

$\cup$

$\mathtt{a}^n\mathtt{b}^n\mathtt{c}\mathtt{a}^{m-1}\mathtt{b}^m$ (in case, in the original string, $n < m$)

We show that $chop(L)$ is not context free. First, note that if $chop(L)$ is context free then so is:

$L' = chop(L) \cap \mathtt{a}^*\mathtt{b}^*\mathtt{a}^*\mathtt{b}^*$. $L' = \mathtt{a}^n\mathtt{b}^n\mathtt{a}^n\mathtt{b}^n$.

We show that $L'$ is not context free by pumping. Let $w = \mathtt{a}^k\mathtt{b}^k\mathtt{a}^k\mathtt{b}^k$. The rest is straightforward.

b) $mix(L) = \{w: x, y, z: (x \in L, x = yz, |y| = |z|, w = yz^R)\}$.

Not closed. We prove this by showing a counterexample. Let $L = \{(\mathtt{aa})^n(\mathtt{ba})^{3n},\ n \geq 0\}$. $L$ is context-free, since it can be generated by the grammar:

$S \rightarrow \mathtt{aa}S\mathtt{bababa}$
$S \rightarrow \varepsilon$

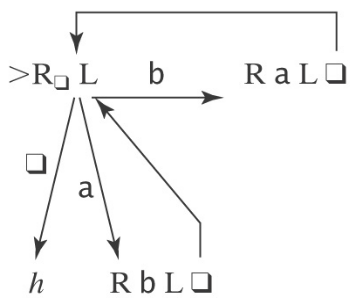
So every string in $L$ is of the form $(\mathtt{aa})^n(\mathtt{ba})^n \mid (\mathtt{ba})^n(\mathtt{ba})^n$, with the middle marked with a |.
$mix(L) = (\mathtt{aa})^n(\mathtt{ba})^n \mid (\mathtt{ab})^n(\mathtt{ab})^n = (\mathtt{aa})^n(\mathtt{ba})^n \mid (\mathtt{ab})^{2n}$. We show that this language is not context-free using the Pumping Theorem.

Let $w = (\mathtt{aa})^k(\mathtt{ba})^k\ (\mathtt{ab})^{2k}$
1 | 2 | 3

If either $v$ or $y$ crosses regions, pump in once and the resulting string will not have the correct form to be in $mix(L)$. If $|vy|$ is not even, pump in once, which will result in an odd length string. All

strings in *mix*(*L*) have even length. We consider the remaining cases for where nonempty *v* and *y* can occur:
(1, 1) Pump in. need 5 a's for every 3 b's. Too many a's.
(2, 2) Pump in. In every string in *mix*(*L*), there's an instance of aa between the ba region and the ab region. There needs to be the same number of ba pairs before it as there are ab pairs after it. There are now more ba pairs.
(3, 3) Pump in. " except now more ab pairs.
(1, 2) Pump in. Same argument as (2, 2).
(2, 3) Pump in. In every string in *mix*(*L*), there must be 3 a's after the first b for every 2 a's before it. There are now too many.
(1, 3) $|vxy| \le k$.

**Exercise 5)** Give a short English description of what each of this Turing machines does. Note the alphabet $\Sigma_M = \{a, b\}$.




Shift the input string one character to the right and replace each b with an a and each a with a b.


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