Australian National University

COMP3630 Theory of Computation

Assignment 02

Edited by LaTeX

College of Engineering and Computer Science

©*Author*
Jimmy Lin
u5223173

¶ *Lecturer*
Jinbo Huang

¶ *Lecturer*
Dirk Pattinson

† *Release Date*
April. 15 2013

‡ *Due Date*
April. 29 2013

τ *Time Spent*
23h

April 29, 2013

Contents

1 Proof by Induction

Consider the CFG G defined by productions:

$$S \to aS \mid Sb \mid a \mid b$$

1.1 Prove by induction on the string

Prove by induction on the string length that no string in $L(G)$ has ba as a substring.

In $L(G)$, it is impossible for those strings, whose length is 0 and 1, to have ba as substring, since the length of a string cannot be smaller than its substring. Hence, we just need to make mathematical induction for the string whose length is no less than 2 ($n \geq 2$).

Base Case: for $n = 2$, prove that

$$\forall string\ w \in L(G)\ and\ |w| = 2,\ have\ no\ substring\ ba$$

Proof:
All possible derivations for string of length 2 are as follows,

$$S \Rightarrow aS \Rightarrow aa \tag{1}$$

$$S \Rightarrow aS \Rightarrow ab \tag{2}$$

$$S \Rightarrow Sb \Rightarrow ab \tag{3}$$

$$S \Rightarrow Sb \Rightarrow bb \tag{4}$$

It can be seen from above derivations that all possible strings of length 2 are,

$$aa,\ ab,\ bb$$

Obviously, ba is not included in the possible set of strings of length 2. Therefore, NO strings of length two in language $L(G)$ has substring ba. Formally, we have

$$\forall string\ w \in L(G)\ and\ |w| = 2,\ have\ no\ substring\ ba. \tag{5}$$

Inductive Cases: for arbitrary $n \geq 2$
Assume that

$$\forall\ string\ w \in L(G)\ and\ |w| = m,\ have\ no\ substring\ ba \tag{6}$$

Prove that for string of length $m + 1$ have the same property. Formally

$$\forall\ string\ w' \in L(G)\ and\ |w'| = m + 1,\ have\ no\ substring\ ba \tag{7}$$

Proof:
We consider the first-step derivation for string w'. There are two possible cases based on the production of grammar G.

$$\textbf{First case: } S \Rightarrow aS \Rightarrow aw \qquad w' = aw \tag{8}$$

$$\textbf{Second case: } S \Rightarrow Sb \Rightarrow wb \qquad w' = wb \tag{9}$$

Based on the inductive hypothesis (6), the string w of length m always has no substring ba. We use this property of string w to derive the followings,

- First Case: We have shown that no substring ba in string w (inductive hypothesis). Besides, whatever the first letter of string w is a or b, append one a in head of the string w will not introduce occurence of ba. Therefore, in this case, string w' of length $m+1$ has no substring ba.
- Second Case: We have shown that no substring ba in string w (inductive hypothesis). Besides, whatever the last letter of string w is a or b, append one b in tail of the string w will not introduce occurence of ba. Therefore, in this case, string w' of length $m + 1$ has no substring ba.

In summary, we have proven that there is no substring ba in any string of length $m+1$ given no substring ba in any string of length m. That is, the inductive cases hold. According to the principle of mathematical induction, it is concluded that

$$\forall string\ w \in L(G),\ |w| \geq 2,\ w\ have\ no\ substring\ ba \tag{10}$$

Since the string of length 0 or 1 in the language of grammar G is impossible to have substring ba, as we discussed in the very beginning, it can be concluded that

$$\text{NO string in } L(G) \text{ has } ba \text{ as a substring.} \tag{11}$$

1.2 Describe $L(G)$ informally

Describe $L(G)$ informally. Justify your answer using part (1).

To get the informal description, we observe some patterns from those production rules that contain variable in the body.

- ◇ terminal a always occur before the variable S.
- ◇ b always occurs after variable S.
- ◇ variable S by itself can be instantiated to both terminal a and b.

Based on the observation above, the informal description of $L(G)$ is as follows,

$$L(G) = \{\text{all occurrences of } a \text{ come before any occurrence of } b\} \tag{12}$$

Then justify my answer.

Let L_1 to be set of strings in which all occurrences of a come before any occurrence of b, and let L_2 to be the set of strings that contains ba as a substring. Now, our ultimate objective is to prove that $L(G) = L_1$.

But first we should prove another lemma, that is

$$\overline{L_2} = L_1 \tag{13}$$

To prove this lemma, we should prove both of followings,

$$\overline{L_2} \subseteq L_1 \tag{14}$$
$$L_1 \subseteq \overline{L_2} \tag{15}$$

First things first, we prove (14), that is to prove

$$\forall w \in \overline{L_2},\ w \in L_1 \tag{16}$$

Prove it by contradition. Assume that there exists $w^* \in \overline{L_2},\ w^* \notin L_1$. Since w^* is not in L_1, there exists some b before a in the string w^*. Thus, we can write $w^* = ZbXaY$, where Z, X, Y are arbitrary strings. Then we use induction on the length of X to prove w^* must have substring ba, that is $w^* \notin \overline{L_2}$.

Base Case, when $|X| = 0$, we have $w^* = ZbaY$, which apparently has substring ba.

Inductive Cases, assume that for $|q| = n$, $w^* = ZbqaY$ has substring ba, prove that for $|q'| = n+1$, $w^{*'} = Zbq'aY$ has substring ba. Because the $|q'| = |q| + 1$, we have two possibilities for the form of q'. They are $q' = qa$ and $q' = qb$.

- In case of $q' = qa$, $w^{*'} = Zbq'aY = ZbqaaY$ and because we have shown in the inductive hypothesis that $ZbqaY$ has substring ba, obviously $w^{*'} = ZbqaaY$ must have substring ba.
- In the case of $q' = qb$, we have $w^{*'} = Zbq'aY = ZbqbaY$, which apparently has substring ba.

As shown above, we prove that w^* must have substring ba, which contradicts to our assumption that $w^* \in \overline{L_2}$. Therefore, we should negate the assumption and conclude that $\forall w \in \overline{L_2},\ w \in L_1$. Hence, we have $\overline{L_2} \subseteq L_1$.

Then, we turn to prove $L_1 \subseteq \overline{L_2}$ (15), that is to prove

$$\forall w \in L_1,\ w \in \overline{L_2} \tag{17}$$

We can **prove it by contradiction**. Assume there exists $w^* \in L_1, w^* \notin \overline{L_2}$ *holds*. since $w^* \notin \overline{L_2}$, w^* have substring ba, which make w^* not in L_1 because we have the definition of L_1 is that all occurrence a comes before b. Therefore, we prove (17) by negating the assumption.

Until now, we proved the (14) and (15), so we achieved the proof of $\overline{L_2} = L_1$.

Now let us start to prove the ultimate objective, $L(G) = L_1$. We should prove both of followings,

$$L(G) \subseteq L_1 \tag{18}$$
$$L_1 \subseteq L(G) \tag{19}$$

Proof for $L(G) \subseteq L_1$:
The result of part (1) demonstrates that

$$L(G) \subseteq \overline{L_2} \tag{20}$$

Since we have proven the lemma (13) $\overline{L_2} = L_1$,

$$L(G) \subseteq L_1 \tag{21}$$

Proof for $L_1 \subseteq L(G)$:
To prove $L_1 \subseteq L(G)$, we need to show that for any string $w \in L_1$, it holds that $w \in L(G)$. We prove this by induction on the length of string w.

Base Case: when $|w| = 1$, it is true without any doubt because a and b are both in L_1 and $L(G)$.

Inductive Case: Postulate the claim that if $w \in L_1$, then $w \in L(G)$ holds for $|w| = n$, show that this claim still holds for $|w'| = |w| + 1 = n + 1$.

Since $w' \in L_1$, we have two possibilities for the form of w' whose length is $n + 1$, that is,

$$a^i b^{n+1-i} \ (0 \leq i \leq n+1) \text{ or } b^{n+1}. \tag{22}$$

- In the case of $w' = a^i b^{n+1-i} = a a^{i-1} b^{n+1-i}$, since $w = a^{i-1} b^{n+1-i}$ is of length n and belongs to L_1, $w \in L(G)$ according to inductive hypothesis. Hence, w' can be derived by $S \Rightarrow aS \Rightarrow aa^{i-1}b^{n+1-i}$.
- The other case is that $w' = b^{n+1}$, and similarly we can derive the string w' by $S \Rightarrow bS \Rightarrow bb^n$.

It can be concluded by those two cases above, if $w' \in L_1$, w' can always be derived by grammar G given $w = |w'| - 1$ belongs to L_1 and $L(G)$.

Therefore, we prove that if $w \in L_1$, then $w \in L(G)$, that is to say, $L_1 \subseteq L(G)$.

All in all, we fulfill the proof for $L(G) \subseteq L_1$ and $L_1 \subseteq L(G)$. Hence, $L(G) = L_1$ holds. That is,

$$L(G) = L_1 = \{\text{all occurrences of } a \text{ come before any occurrence of } b\} \tag{23}$$

2 Unambiguous Grammars

The following grammar generates prefix expressions with operands x and y and binary operators $+, -, *$,

$$E \to +EE \mid \times EE \mid -EE \mid x \mid y$$

2.1 Give leftmost and rightmost derivations, and a parse tree

Give leftmost and rightmost derivations, and a parse tree for the string +*-xyxy.

First left-most derivation for string +*-xyxy,

$$\begin{aligned}
E &\underset{lm}{\Longrightarrow} +EE && \text{by } E \to +EE && (24)\\
&\underset{lm}{\Longrightarrow} +\times EEE && \text{by } E \to \times EE && (25)\\
&\underset{lm}{\Longrightarrow} +\times -EEEE && \text{by } E \to -EE && (26)\\
&\underset{lm}{\Longrightarrow} +\times -xEEE && \text{by } E \to x && (27)\\
&\underset{lm}{\Longrightarrow} +\times -xyEE && \text{by } E \to y && (28)\\
&\underset{lm}{\Longrightarrow} +\times -xyxE && \text{by } E \to x && (29)\\
&\underset{lm}{\Longrightarrow} +\times -xyxy && \text{by } E \to y && (30)
\end{aligned}$$

Then right-most derivation for string +*-xyxy,

$$\begin{aligned}
E &\underset{rm}{\Longrightarrow} +EE && \text{by } E \to +EE && (31)\\
&\underset{rm}{\Longrightarrow} +Ey && \text{by } E \to y && (32)\\
&\underset{rm}{\Longrightarrow} +\times EEy && \text{by } E \to \times EE && (33)\\
&\underset{rm}{\Longrightarrow} +\times Exy && \text{by } E \to x && (34)\\
&\underset{rm}{\Longrightarrow} +\times -EExy && \text{by } E \to -EE && (35)\\
&\underset{rm}{\Longrightarrow} +\times -Eyxy && \text{by } E \to y && (36)\\
&\underset{rm}{\Longrightarrow} +\times -xyxy && \text{by } E \to x && (37)
\end{aligned}$$

Finally, the parse tree (yield) for string +*-xyxy,

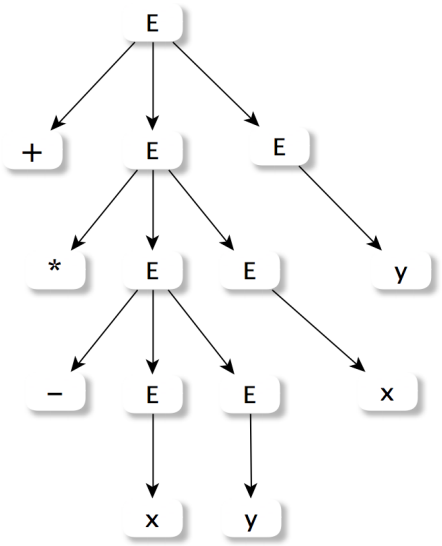

Fig. Parse Tree for word +*-xyxy

2.2 Prove unambiguity of given grammar

Prove that this grammar is unambiguous.

To prove the unambiguity of the grammar, we need to prove that the leftmost derivation for all strings in that language are unique. That is,

$$\forall \text{string } w \in L(G),\ E \underset{lm}{\overset{*}{\Longrightarrow}} w \text{ is unique.} \tag{38}$$

Proof: Mathematical induction implemented on the nubmer of steps of "expanded derivation", under which there is one variable E instantiated to $+EE \mid \times EE \mid -EE$. (Note that we just call such derivation as "expanded derivation" for that only "expanded derivation" will enlarge the length of string. And by contrast, $E \to x \mid y$ is not "expanded derivation".)

Notational definition: E_k represents one variable E to be instantiated to a string with k step expanded derivation.

Base Case: For those strings with no expanded derivation in the language of grammar E, it is evident that they are unique in leftmost derivation.

$$E_0 \underset{lm}{\overset{*}{\Longrightarrow}} x \tag{39}$$

$$E_0 \underset{lm}{\overset{*}{\Longrightarrow}} y \tag{40}$$

Inductive Cases:
Assume that the

$$\forall \text{ string } r, \text{ obtained by } k \text{ step expanded derivation, } E_k \underset{lm}{\overset{*}{\Longrightarrow}} r \text{ is unique.} \tag{41}$$

Prove that

$$\forall \text{ string } r', \text{ obtained by } k+1 \text{ step expanded derivation, } E_{k+1} \underset{lm}{\overset{*}{\Longrightarrow}} r' \text{ is unique.} \tag{42}$$

Proof:
Generally, we consider first step of leftmost derivation for variable E with k+1 expanded derivation. There are three possibibilities as follows,

$$E_{k+1} \underset{lm}{\Longrightarrow} +E_iE_j \tag{43}$$

$$E_{k+1} \underset{lm}{\Longrightarrow} \times E_iE_j \tag{44}$$

$$E_{k+1} \underset{lm}{\Longrightarrow} -E_iE_j \tag{45}$$

where i is the number of expanded derivation to be instantiated in the first variable, while j is that of second variable E. $i + j = k$, which means there are still k step expanded derivation in total remained for the the first E and second E in right-hand side of above derivation.

Written in a compact form as follows,

$$E_{k+1} \underset{lm}{\Longrightarrow} (+ \mid \times \mid -)E_iE_j \tag{46}$$

Since $i + j = k$, $i \geq 0$ and $j \geq 0$, it can be obtained that

$$i \leq k \text{ and } j \leq k \tag{47}$$

Now we need to use the following lemma, which obviously holds. (The proof for this lemma is provided after the solution for this question. We just uses it first without interrupting this proof.)

Lemma: if leftmost derivation for E_x is unique, the leftmost derivation for E_y ($y < x$) must be unique.

According to the introduced lemma, since $i \leq k$ and $j \leq k$ (see (47)) and E_k is unique (inductive hypothesis), it is concluded that

$$E_i \text{ and } E_j \text{ are both unique in its leftmost derivation.} \tag{48}$$

Therefore, leftmost derivation for E_{k+1} is unique because all terminals and variables in its first expanded derivation (46) is unique.

Hence, by principle of mathematical induction, it is concluded that all strings that are derived in by grammar G in leftmost derivation is unique, which is sufficient to induce the unambiguity of the grammar G by definition of unambiguity.

Now we prove the lemma introduced in the above proof.

Lemma: if leftmost derivation for E_x is unique, the leftmost derivation for E_y, $(y < x)$ must be unique.

We prove this by contradiction.

It is known that leftmost derivation for E_x is unique, and we assume that leftmost derivation $E_y(y < x)$ is not unique.

The E_x can be represented as,

$$E_x \underset{lm}{\Longrightarrow} (+| \times |-)E_y E_{x-y-1} \tag{49}$$

Since the $E_y(y < x)$ is not unique in its leftmost derivation, E_x **must not be unique**. This is because E_x has one component E_y, which is not unique.

Therefore, we find a contradiction that E_x cannot be both unique and not unique, for which we should negate the assumption that $E_y(y < x)$ is not unique. That is to say,

$$\text{if } E_x \text{ is unique, then } E_y(y < x) \text{ must be unique.} \tag{50}$$

3 Acceptance by Final State vs. Empty Stack

Consider the PDA $P = (\{p, q\}, \{0, 1\}, \{Z_0, X\}, \delta, Z_0, q, \{p\})$, whose transition function is,

$$\delta(q, 0, Z_0) = \{(q, XZ_0)\} \tag{51}$$
$$\delta(q, 0, X) = \{(q, XX)\} \tag{52}$$
$$\delta(q, 1, X) = \{(q, X)\} \tag{53}$$
$$\delta(q, \epsilon, X) = \{(p, \epsilon)\} \tag{54}$$
$$\delta(p, \epsilon, X) = \{(p, \epsilon)\} \tag{55}$$
$$\delta(p, 1, X) = \{(p, XX)\} \tag{56}$$
$$\delta(p, 1, Z_0) = \{(p, \epsilon)\} \tag{57}$$

3.1 Convert P to another PDA that accepts by empty stack

Convert P to another PDA P_1 that accepts by empty stack the same language that P accepts by final state; i.e., $N(P_1) = L(P)$.

To establish an equivalent PDA with acceptance by empty stack, we utilize an additional state s_0 to introduce the initial bottom of statck Z_0 in the original PDA P, and another new state s_f to label acceptance.

The P_1 we construct is $(\{s_0, s_f, p, q\}, \{0, 1\}, \{Y_0, Z_0, X\}, \delta, Y_0, s_0)$, transition functions are as follows,

$$\delta(q, 0, Z_0) = \{(q, XZ_0)\} \tag{58}$$
$$\delta(q, 0, X) = \{(q, XX)\} \tag{59}$$
$$\delta(q, 1, X) = \{(q, X)\} \tag{60}$$
$$\delta(q, \epsilon, X) = \{(p, \epsilon)\} \tag{61}$$
$$\delta(p, \epsilon, X) = \{(p, \epsilon), (s_f, \epsilon)\} \tag{62}$$
$$\delta(p, 1, X) = \{(p, XX)\} \tag{63}$$
$$\delta(p, 1, Z_0) = \{(p, \epsilon)\} \tag{64}$$
$$\delta(s_0, \epsilon, Y_0) = \{(q, Z_0Y_0)\} \tag{65}$$
$$\delta(p, \epsilon, Z_0) = \{(s_f, \epsilon)\} \tag{66}$$
$$\delta(p, \epsilon, Y_0) = \{(s_f, \epsilon)\} \tag{67}$$
$$\delta(s_f, X, \epsilon) = \{(s_f, \epsilon)\} \tag{68}$$
$$\delta(s_f, Y_0, \epsilon) = \{(s_f, \epsilon)\} \tag{69}$$
$$\delta(s_f, Z_0, \epsilon) = \{(s_f, \epsilon)\} \tag{70}$$

As we can see, only (62), (65), (66), (67), (68), (69) and (70) are new transitions.

Note that six components are sufficient to describe formally a PDA with acceptance by empty stack.

Then is the graphical representation of new PDA P_1,

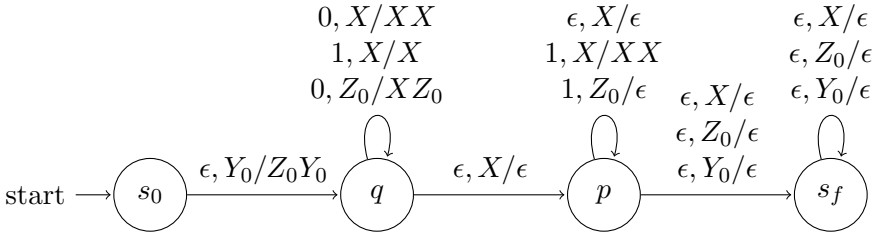

Fig. PDA P_1 with acceptance by empty stack

3.2 Find a PDA P2 such that $L(P_2) = N(P)$

Find a PDA P_2 such that $L(P_2) = N(P)$; i.e., P_2 accepts by final state what P accepts by empty stack.

To establish an equivalent PDA P_2 with acceptance by final state, we utilize an additional state s_0 to introduce the initial bottom of statck Z_0 in the original PDA P, and another new state s_f as final state of new PDA P_2.

The P_2 we construct is $(\{s_0, s_f, p, q\}, \{0, 1\}, \{Y_0, Z_0, X\}, \delta, Y_0, s_0, \{s_f\})$, transition functions are as follows,

$$\delta(q, 0, Z_0) = \{(q, XZ_0)\} \tag{71}$$
$$\delta(q, 0, X) = \{(q, XX)\} \tag{72}$$
$$\delta(q, 1, X) = \{(q, X)\} \tag{73}$$
$$\delta(q, \epsilon, X) = \{(p, \epsilon)\} \tag{74}$$
$$\delta(p, \epsilon, X) = \{(p, \epsilon)\} \tag{75}$$
$$\delta(p, 1, X) = \{(p, XX)\} \tag{76}$$
$$\delta(p, 1, Z_0) = \{(p, \epsilon)\} \tag{77}$$
$$\delta(s_0, \epsilon, Y_0) = \{(q, Z_0Y_0)\} \tag{78}$$
$$\delta(q, \epsilon, Y_0) = \{(s_f, \epsilon)\} \tag{79}$$
$$\delta(p, \epsilon, Y_0) = \{(s_f, \epsilon)\} \tag{80}$$

Then is the graphical representation of new PDA P_2,

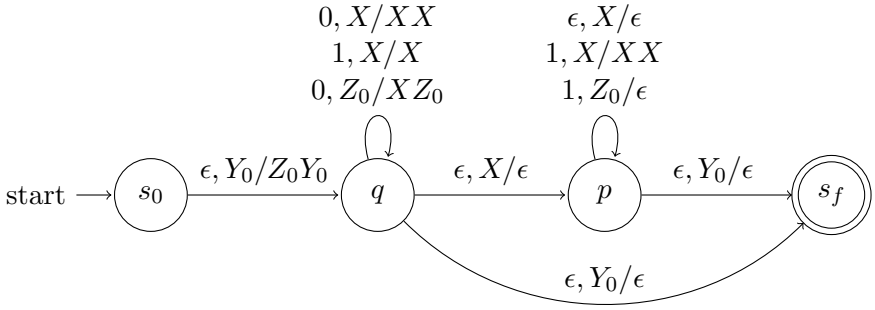

Fig. PDA P_2 with acceptance by final state

4 From Grammar to PDA

Convert the grammar

$$S \to aAA$$
$$A \to aS \mid bS \mid a$$

to a PDA that accepts the same language by empty stack, and show an accepting sequence of IDs for input string *aabaaa*.

4.1 Conversion to PDA P_{Stack}

Based on the given grammer, construct a PDA with acceptance by empty stack $P_{Stack}(Q, \Sigma, T, q_s, \delta, I)$ whose formal definition is

- State sets $Q = \{q\}$
- Alphabet of Input String $\Sigma = \{a, b\}$
- Stack Symbols $T = \{S, A, a, b\}$
- Start State $q_s = q$
- Initial symbol in the stack $I = S$
- Transition functions δ are defined as follows, (here write $\delta(q, \epsilon, A)$ separately for better reference)

$$\delta(q, \epsilon, S) = \{(q, aAA)\} \tag{81}$$
$$\delta(q, \epsilon, A) = \{(q, aS), (q, bS), (q, a)\} \tag{82}$$
$$\delta(q, a, a) = \{(q, \epsilon)\} \tag{83}$$
$$\delta(q, b, b) = \{(q, \epsilon)\} \tag{84}$$

4.2 Show sequence of IDs for *aabaaa*

The sequence of IDs is presented to show acceptance of string *aabaaa* by P_{Stack}, the number over the derivation mark is index of the transition equation based on which that step of derivation is made,

$$(q, aabaaa, S) \underset{P_{Stack}}{\vdash} (q, aabaaa, aAA) \quad \text{by (81)} \tag{85}$$
$$\underset{P_{Stack}}{\vdash} (q, abaaa, AA) \quad \text{pop } a \text{ by (83)} \tag{86}$$
$$\underset{P_{Stack}}{\vdash} (q, abaaa, aA) \quad \text{pop } A, \text{ push } a \text{ by (82)} \tag{87}$$
$$\underset{P_{Stack}}{\vdash} (q, baaa, A) \quad \text{pop } a \text{ by (83)} \tag{88}$$
$$\underset{P_{Stack}}{\vdash} (q, baaa, bS) \quad \text{pop } A, \text{ push } bS \text{ by(82)} \tag{89}$$
$$\underset{P_{Stack}}{\vdash} (q, aaa, S) \quad \text{pop } b \text{ by (84)} \tag{90}$$
$$\underset{P_{Stack}}{\vdash} (q, aaa, aAA) \quad \text{pop } S, \text{ push } aAA \text{ by (81)} \tag{91}$$
$$\underset{P_{Stack}}{\vdash} (q, aa, AA) \quad \text{pop } a \text{ by (83)} \tag{92}$$
$$\underset{P_{Stack}}{\vdash} (q, aa, aA) \quad \text{pop } A, \text{ push } a \text{ by (82)} \tag{93}$$
$$\underset{P_{Stack}}{\vdash} (q, a, A) \quad \text{pop } a \text{ by (83)} \tag{94}$$
$$\underset{P_{Stack}}{\vdash} (q, a, a) \quad \text{pop } A, \text{ push } a \text{ by (82)} \tag{95}$$
$$\underset{P_{Stack}}{\vdash} (q, \epsilon, \epsilon) \quad \text{pop } a \text{ by (83)} \tag{96}$$
$$\Rightarrow \textit{accepted} \quad \text{acceptance by empty stack} \tag{97}$$

5 Use of CFL Pumping Lemma

5.1 Show language $\{a^i b^j c^k \mid i \times j = k\}$ not to be context-free

Use the CFL pumping lemma to show the following language not to be context-free:

$$\{a^i b^j c^k \mid i \times j = k\}$$

First, we assume that the language

$$L = \{a^i b^j c^k \mid i \times j = k\} \textit{ is context free language.} \tag{98}$$

Consider the $z = a^{2n} b^n c^{2n^2}$ for arbitrary n.
Decompose the word, we have $z = uvwxy$

$$z = uvwxy \tag{99}$$

Such decomposition is valid because the length of z,

$$2n + n + 2n^2 \geq n \tag{100}$$

By pumping lemma, the following condition is satisfied since L is context-free language,

$$|vwx| \leq n \tag{101}$$
$$|vx| > 0 \tag{102}$$
$$\forall r \geq 0, uv^r wx^r y \in L \tag{103}$$

Since the length of vwx is restricted to be at most n, all possibibilities for assignment of v and x are easier to be detected. And also we should show that in each case, after pumping v and x, the word z is no longer in the language L.

- vwx is all as, in this case, both v and x must be all as. If we dumplicate the v and x, the number of as in resulting word will increase, while the number of bs and cs remain unchanged. The resulting word does not satisfy $i \times j = k$ any more and thus not in language L.
- vwx is all bs, in this case, both v and x must be all bs. If we dumplicate the v and x, the number of bs in resulting word will increase, while the number of as and cs remain unchanged. The resulting word does not satisfy $i \times j = k$ any more and thus not in language L.
- vwx is all cs, in this case, both v and x must be all cs. If we dumplicate the v and x, the number of cs in resulting word will increase, while the number of as and bs remain unchanged. The resulting word does not satisfy $i \times j = k$ any more and thus not in language L.
- vwx consists of as and bs, in this case, v consist of as or possibly as with some bs, and x consists of bs or possibly bs with some as. If we dumplicate the v and x, the number of a and b in resulting word will increase, while the number of cs remain unchanged. The resulting word does not satisfy $i \times j = k$ any more and thus not in language L.
- vwx is combination of b and c, specific scenarios are showns in the following.
  - v consists of only bs, x consists of only cs. Since all cases we talked about here has one or more b, suppose the number of b in v is $p \geq 1$. Now we want to duplicate the v and x. To keep the resulting word satisfy the property $i \times j = k$, the number of c in x must be $2n \times p$, which is impossible because we have restricted the length of vwx to be smaller than or equal to n. (here $|vwx| > |x| = 2np > n$)
  - v consists of only bs, x consists of bs and cs. Obviously, duplicating the x will violate the form of $a^i b^j c^k$, hence the result would never in language L.
  - v consists of bs and cs, x is all cs. Obviously, duplicating the v will violate the form of $a^i b^j c^k$, hence the result would never in language L.

As we can see from the analysis above, in all cases, (103) is contradicted after pumping or duplicating of v and x for r times. Therefore, we should negate our assumption (98) at the beginning.

Hence, we sucessfully prove that

$$\{a^i b^j c^k \mid i \times j = k\} \; is \; not \; context \; free \; language \tag{104}$$

6 Closures Properties of CFLs

6.1 Show $max(\cdot)$ has no closure property for CFL

Show that the CFLs are *not* closed under the following operation:

$$max(L) = \{\text{w} \mid \text{w is in } L \text{ and for no } x \text{ other than } \epsilon \text{ is } wx \text{ in } L\}$$

Proof:
First of all, we assume that context-free language is closed under the $max(\cdot)$ operator. Hence, we have

$$\forall \text{context-free language } L, max(L) \text{ is also context-free language.} \tag{105}$$

And then pick out a language $L^* = \{0^i1^j2^k \mid i \geq k \text{ or } j \geq k\}$, which is a context-free because we can easily construct a context-free grammar E to genereate this language,

$$E \to AB \mid D \tag{106}$$
$$A \to A0 \mid \epsilon \tag{107}$$
$$B \to 1B \mid 1B2 \mid \epsilon \tag{108}$$
$$D \to 0D \mid F \tag{109}$$
$$F \to 0F2 \mid G \tag{110}$$
$$G \to G1 \mid \epsilon \tag{111}$$

Note that, we guarantee that $j \geq k$ if E is instantiated to AB, and we secure that $i \geq k$ if E is instantiated to D.

After verifying that the L^* we pick up is context-free language, we start to show the language $max(L^*)$, which is processed by operator $max(\cdot)$, is not context-free language.

Based on the given definition of operator $max(\cdot)$, we can derive the form of $max(L^*)$ as follows,

$$max(L^*) = \{0^i1^j2^k \mid k = max(i, j)\} \tag{112}$$

Now prove the $max(L^*)$ is not contex-free language by pumping lemma.

We assume that the language $max(L^*)$ in (112) is context-free language.

Consider the string $s = 0^Q1^Q2^Q$, which is obviously in $max(L^*)$. (Q is an arbitrary constant.) Since the language $max(L^*)$ is a context-free language by our assumption, we can decompose the string s into five part, that is,

$$s = uvwxy, \ |uvwxy| \geq Q \tag{113}$$

Such that,

$$|vwx| \leq Q \tag{114}$$
$$|vx| > 0 \tag{115}$$
$$\forall z \geq 0, \ uv^zwx^zy \in max(L^*) \tag{116}$$

Now we make analysis for all possible assignment of vwx, since the length of vwx is limited, the string vwx cannot access all three different characters at the same time.

- if vwx consists of only $0s$, pump v and x will make resulting $s' = uv^zwx^zy$ remain in $max(L^*)$, but if we duplicate the v and x, in which way $k \neq max(i, j)$ since the i has increased but k kept unchanged, the resulting s' will be not in $max(L^*)$. Hence, (116) does not hold in this case.
- if vwx consists of only $1s$, similar to the all-0 case, pump v and x will make resulting s' remain in $max(L^*)$, but if we duplicate the v and x, in which way $k \neq max(i, j)$ since the j has increased but k kept unchanged, the resulting s' will be not in $max(L^*)$. Hence, (116) does not hold in this case.

The condition provided by pumping lemma in (116) says no matter pumping or duplicting v and z for any times, the resulting string must remain in the context-free language.

- if vwx consists of only $2s$, pumping or duplication of v and x will make the resulting s' not in $max(L^*)$ in that the k has a different value while the i and j remained unvaried. Hence, (116) does not hold in this case.
- if vwx consists of $0s$ and $1s$, (116) does not hold in all possible scenarios of this case as follows:
  - ◇ if v consists of both $0s$ and $1s$, and x only consists of $1s$, the duplicated string will not be in the form $0^i1^j2^k$.
  - ◇ if v consists of only $0s$, and x consists of both $0s$ and $1s$, the duplicated string will not be in the form $0^i1^j2^k$.
  - ◇ if v consists of only $0s$ and x consists of only $1s$, any pumping or duplication of v and x will make the resulting s' not in $max(L^*)$ in that both i and j changed but k has no move.
- if vwx consists of $1s$ and $2s$, (116) does not hold in all possible scenarios of this case as follows:
  - ◇ if v consists of both $1s$ and $2s$, and x only consists of $2s$, the duplicated string will not be in the form $0^i1^j2^k$.
  - ◇ if v consists of only $1s$, and x consists of both $1s$ and $2s$, the duplicated string will not be in the form $0^i1^j2^k$.
  - ◇ if v consists of only $1s$ and x consists of only $2s$, duplication of v and x may keep the resulting s' in $max(L^*)$ if $|v| = |x|$, but pumping v and x inevitably break the $k = max(i, j)$ since the $max(i, j)$ is i, which has no variation.

In summary, for all possible assignment of vwx, the (116) does not hold for $max(L^*)$. That is to say, we find a conflict for the assumption made before that the $max(L^*)$ is context-free language. Hence, we ought to negate the assumption. Therefore, we have

$$\text{the language } max(L^*) \text{ is not context-free language.}$$

Thus, we find a example L^* to illustrate that if L is context-free language, its maximized language $max(L)$ is not necessary to be context-free language. Formally,

$$\exists \text{ context-free language } L,\ max(L) \text{ is } \textit{not} \text{ context-free language.}$$

which contradicts to what we have derived in (105) by assuming the closure property of $max(\cdot)$ in the very beginning.

As the consequence of this second contradiction, we negate the assumption in the very beginning and obtain

$$\text{CFLs are } \textit{not} \text{ closed under the operator } max(\cdot) \tag{117}$$