Department of Computer Science
Ashoka University

Programming Language Design and Implementation (PLDI): CS-1319-1

MidSem Test: Offline *Marks: 100*
Date: *October 17, 2023* Time: *16:40 - 18:10*

Instructions:

1. The test comprises 10 questions (totalling 100 marks) and one bonus question (for 10 marks). Some questions have multiple parts with marks shown for each.
2. The test is entirely closed book.
3. Any copying from peers will be dealt with zero tolerance - both to get zero in the question.
4. Remember to write both your name & roll number on the answer-script.
5. No query or doubt will be entertained. If you have any query, make your own assumptions, state them clearly in your answer and proceed.
6. Write in clear handwriting and in an unambiguous manner. If TF/TAs have difficulty reading / understanding your answer, they will make assumptions at their best capacity to evaluate. You would not get an opportunity for explanation or rebuttal.

Grading Guidelines for TAs: *Please follow these guidelines for marking.*

Consider grammar G:

$$\begin{aligned} S &\rightarrow (\ L \ ) \mid a \\ L &\rightarrow L \ , \ S \mid S \end{aligned}$$

G is used across all questions in this test.

1. Write grammar $G = < T, \ N, \ S, \ P >$ formally, stating its four components where symbols have the usual meaning. Also explain the meaning of every symbol. **[2 + 2 = 4]**

   BEGIN SOLUTION

$$\begin{aligned} T &= \{(, \ ), \ ',', \ a\} \\ N &= \{L, S\} \\ S &= S \\ P &= \{S \rightarrow (\ L \ ); \ S \rightarrow a; \ L \rightarrow L \ , \ S; \ L \rightarrow S\} \end{aligned}$$

   where

$$\begin{aligned} T &= Set \ of \ terminal \ symbols \\ N &= Set \ of \ non-terminal \ symbols \\ S &= Start \ non-terminal \\ P &= Set \ of \ production \ rules \ of \ the \ form \ A \rightarrow \alpha \in (T \cup N)^* \end{aligned}$$

   Grading Guidelines: *0.5 mark for each line above. No partial marking for any missing / wrong line.*

   END SOLUTION

2. Consider a string $x = (a, (a, a), (a))$

   (a) Write a leftmost derivation for x using G. **[7]**

   BEGIN SOLUTION

$$\begin{aligned} S\$ &\Rightarrow \underline{(L)}\$ \ by \ S \rightarrow (L) \\ &\Rightarrow (\underline{L, S})\$ \ by \ L \rightarrow L, S \\ &\Rightarrow (\underline{L, S}, S)\$ \ by \ L \rightarrow L, S \\ &\Rightarrow (\underline{S}, S, S)\$ \ by \ L \rightarrow S \\ &\Rightarrow (\underline{a}, S, S)\$ \ by \ S \rightarrow a \\ &\Rightarrow (a, \underline{(L)}, S)\$ \ by \ S \rightarrow (L) \\ &\Rightarrow (a, (\underline{L, S}), S)\$ \ by \ L \rightarrow L, S \\ &\Rightarrow (a, (\underline{S}, S), S)\$ \ by \ L \rightarrow S \\ &\Rightarrow (a, (\underline{a}, S), S)\$ \ by \ S \rightarrow a \\ &\Rightarrow (a, (a, \underline{a}), S)\$ \ by \ S \rightarrow a \\ &\Rightarrow (a, (a, a), \underline{(L)})\$ \ by \ S \rightarrow (L) \\ &\Rightarrow (a, (a, a), (\underline{S}))\$ \ by \ L \rightarrow S \\ &\Rightarrow (a, (a, a), (\underline{a}))\$ \ by \ S \rightarrow a \\ &= x\$ \end{aligned}$$

   Grading Guidelines: *0.5 mark for each step in the derivation. It it neither mandatory to mention the production rule used nor to underline the handle.*

   50% marks for a sentential form if the replacement is correct but some other part of the string is wrongly copied from the previous step. While marks will not be deducted for missing the \$, for future reference, note that the LR table will not work without it.

   END SOLUTION

(b) Write a rightmost derivation for x using G. [**7**]

BEGIN SOLUTION

$$
\begin{aligned}
S\$ &\Rightarrow (\underline{L})\$ \text{ by } S \to (L) \\
&\Rightarrow (\underline{L, S})\$ \text{ by } L \to L, S \\
&\Rightarrow (L, (\underline{L}))\$ \text{ by } S \to (L) \\
&\Rightarrow (L, (\underline{S}))\$ \text{ by } L \to S \\
&\Rightarrow (L, (\underline{a}))\$ \text{ by } S \to a \\
&\Rightarrow (\underline{L, S}, (a))\$ \text{ by } L \to L, S \\
&\Rightarrow (L, (\underline{L}), (a))\$ \text{ by } S \to (L) \\
&\Rightarrow (L, (\underline{L, S}), (a))\$ \text{ by } L \to L, S \\
&\Rightarrow (L, (L, \underline{a}), (a))\$ \text{ by } S \to a \\
&\Rightarrow (L, (\underline{S}, a), (a))\$ \text{ by } L \to S \\
&\Rightarrow (L, (\underline{a}, a), (a))\$ \text{ by } S \to a \\
&\Rightarrow (\underline{S}, (a, a), (a))\$ \text{ by } L \to S \\
&\Rightarrow (\underline{a}, (a, a), (a))\$ \text{ by } S \to a \\
&= x\$
\end{aligned}
$$

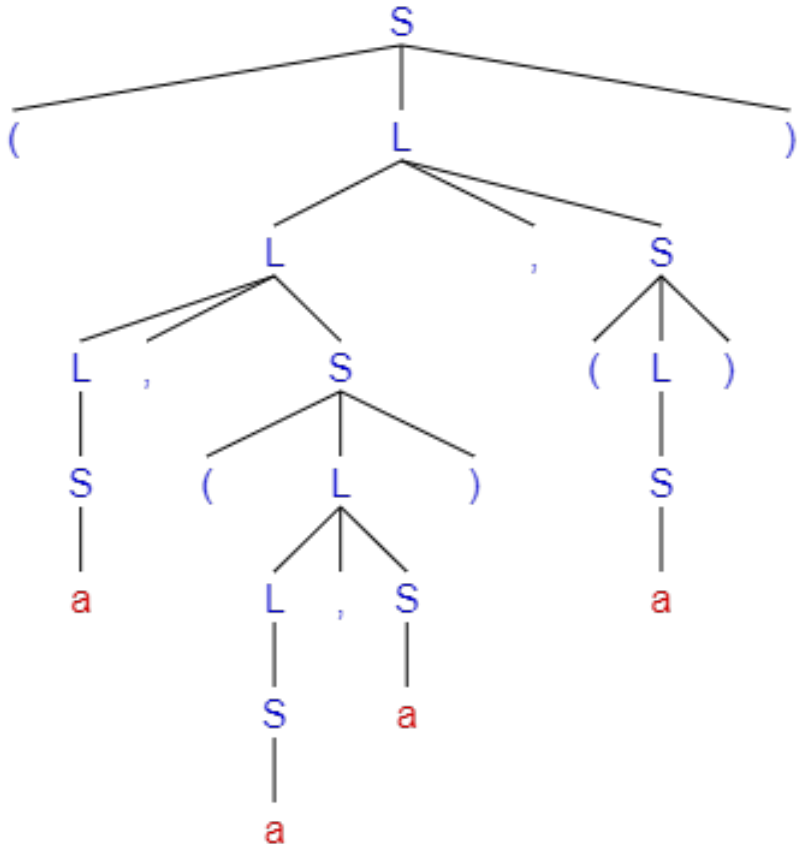
Grading Guidelines: *0.5 mark for each step in the derivation. It it neither mandatory to mention the production rule used nor to underline the handle.*

50% marks for a sentential form if the replacement is correct but some other part of the string is wrongly copied from the previous step. While marks will not be deducted for missing the \$, for future reference, note that the LR table will not work without it.

END SOLUTION

(c) Draw a parse tree for x using G. [**6**]

BEGIN SOLUTION



Grading Guidelines: *0.5 mark for each expansion of the nodes (capped at 6).*

END SOLUTION

3. Describe $L(G)$. Justify your answer. **[2 + 3 = 5]**

BEGIN SOLUTION

$L(G)$ contains all lists built from atomic expression a, list builder () and list concatenater $',$'.

Justification:

Let $L_L(G) = \{x \mid L \Rightarrow_G x\}$ and $L(G) = L_S(G) = \{x \mid S \Rightarrow_G x\}$.

(a) From $S \to a$: $a \in L(G)$. **atomic**

(b) From $S \to (L)$: $x \in L_L(G) \Rightarrow (x) \in L(G)$. **list builder**

(c) From $L \to S$: $x \in L(G) \Rightarrow x \in L_L(G)$. $L(G) \subset L_L(G)$. **recursion**

(d) From $L \to L, S$: $x \in L_L(G) \wedge y \in L(G) \Rightarrow x, y \in L_L(G)$. **concateneter**

Grading Guidelines: *Mention of parenthesised, comma-separated list with atomic element would get 2. If list is not mentioned, deduct 1 mark.*

For justification, each of atom, list build and concatenation should be associated with the production rule, one mark for each.

END SOLUTION

4. Is $L(G)$ regular? If it is regular, design a regular expression $\mathbf{r}$ such that $L(G) = L(\mathbf{r})$. If it is not regular, justify why. **[1 + 5 = 6]**

BEGIN SOLUTION

G is not regular.

Justification 1: We can rewrite G as:

$$
\begin{array}{lcl}
G_1: & & \\
S & \to & (\ L\ ) \mid A \\
L & \to & L\ A\ S \mid S \\
A & \to & a \mid ,
\end{array}
$$

Clearly, $L(G_1) = L(G)$. Replacing a and $',$' by ϵ, we get:

$$
\begin{array}{lcl}
G_2: & & \\
S & \to & (\ L\ ) \mid A \\
L & \to & L\ A\ S \mid S \\
A & \to & \epsilon
\end{array}
$$

Clearly, $G \equiv G_1$ is regular, if G_2 is regular. Now let's have:

$$
\begin{array}{lcl}
G_3: & & \\
S & \to & (\ L\ ) \mid \epsilon \\
L & \to & L\ S \mid S
\end{array}
$$

Clearly, $G_2 \equiv G_3$ and G is regular if G_3 is regular. Now in G_3, $S \Rightarrow (L) \Rightarrow^+ (L^+S) \Rightarrow^+ (S^+S) \Rightarrow \cdots$. Therefore, $G_3 \equiv G_4$ and G is regular if G_4 is regular where:

$$
\begin{array}{lcl}
G_4: & & \\
S & \to & (\ SS\ ) \mid \epsilon
\end{array}
$$

G_4 is the grammar for balanced parenthesised expressions. It is known to be non-regular. Hence, G cannot be regular.

Justification 2: Consider $x = ((((\cdots a \cdots))))$ where the number n of ('s and the number of)'s must match for every n. If G were regular, we could use it to build an infinite counters which is not possible with finite number of states.

Grading Guidelines: *Any of the above justifications or something equivalent would be acceptable for full credit. In some way, it must be established that with finite number of states we can always produce two inputs that would be indistinguishable.*

END SOLUTION

5. Prove: $\forall x \in L(G),\ |x|\ mod\ 2 = 1$. [**4**]

BEGIN SOLUTION

We can make the following observations from G:

$$\begin{aligned} S &\rightarrow (L) \mid a \\ L &\rightarrow L\ ,\ S \mid S \end{aligned}$$

(a) From $S \rightarrow a$: S can produce a string of length 1 which has odd parity.

(b) From $S \rightarrow (L)$ and $L \rightarrow S$: S and L produce strings of the same parity.

(c) From $L \rightarrow L, S$: Using 5b, L produces strings of odd parity (as odd + 1 + odd = odd and even + 1 + even = odd).

(d) Combining 5b and 5c, S produces strings of odd parity.

Grading Guidelines: *Any equivalent argument involving inductive reasoning (or contradiction) would be acceptable. One mark to be deducted for missing the parity argument on any production rule.*

END SOLUTION

6. Can you implement a recursive descent parser for G? Justify your answer. [**2**]

BEGIN SOLUTION

No, we cannot.

G is a left recursive grammar for the rule $L \rightarrow L\ ,\ S$. Any attempt to build an RD parser would lead to a code like below which gets stuck at infinite recursion:

```
L() {
    L(); // infinite recursion
    if (l == ',')
        match(',');
    // ...
    S();
}
```

Hence, we cannot have a recursive descent parser for it.

Grading Guidelines: *Award 0 for claiming that an RD parser is possible. Mark 0.5 to be given for saying that RD parser is not possible. Justification would get full credit if:*

(a) Left recursion is mentioned as the root cause. (0.5 mark)

(b) Illustrating (by code) or justifying infinite recursion. (1 mark)

END SOLUTION

7. G is left-recursive. Perform the following on G.

(a) Remove left recursion in G to create an equivalent grammar G_r. That is, G_r should not have any left-recursive / left-co-recursive production/s and $L(G_r) = L(G)$. [**3**]

BEGIN SOLUTION

Since the productions in G

$$\begin{aligned} S &\rightarrow (L) \\ S &\rightarrow a \\ L &\rightarrow L\ ,\ S \\ L &\rightarrow S \end{aligned}$$

have only a single depth of (direct) left recursion, one application of left recursion removal would suffice. So, $G_r = <T,\ N',\ S,\ P'>$ where $N' = N \cup \{L'\}$ and P' is:

$$\begin{aligned} S &\rightarrow (L) \\ S &\rightarrow a \\ L &\rightarrow S\ L' \\ L' &\rightarrow ,\ S\ L' \\ L' &\rightarrow \epsilon \end{aligned}$$

Grading Guidelines: *1 mark each for every production of the left-recursion removed grammar. No partial mark for a missed or wrong production.*
The following is for understanding. Not expected in the answer.
Recall that a grammar with left recursion

$$\begin{aligned} A &\rightarrow A\alpha \\ A &\rightarrow \beta \end{aligned}$$

can be converted to one without left recursion as:

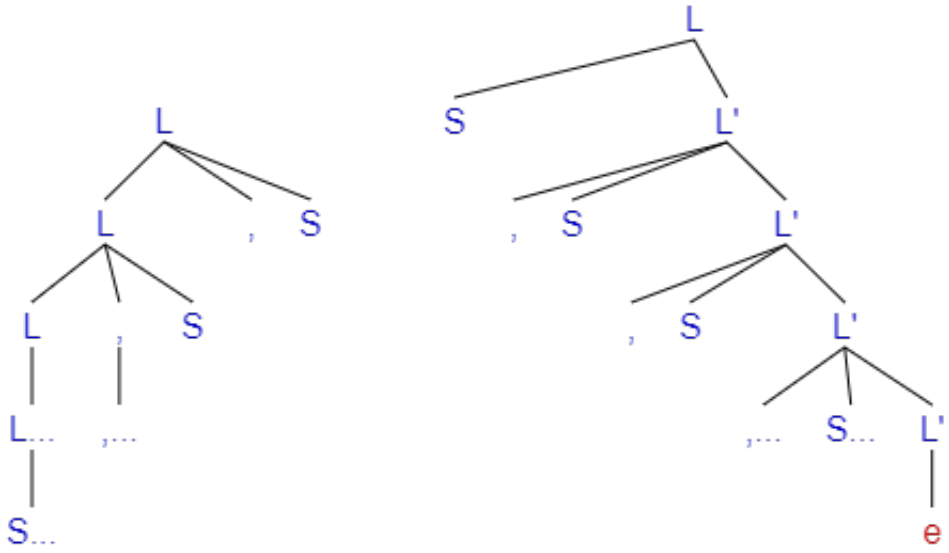
$$\begin{aligned} A &\rightarrow \beta A' \\ A' &\rightarrow \alpha A' \\ A' &\rightarrow \epsilon \end{aligned}$$

END SOLUTION

(b) Prove that $L(G_r) = L(G)$. **[3]**

BEGIN SOLUTION

Since the rules from S have not changed, we just need to show that productions from L produce corresponding parse trees as the trees are rotated from left to right recursion.



Regular Expressions: Parentheses below are for regular expression, not elements of T

$L \Rightarrow L, S \Rightarrow^* L(, S)^+ \Rightarrow S(, S)^+$ $\qquad$ $L \Rightarrow SL' \Rightarrow S, SL' \Rightarrow^* (S,)^+ SL' \Rightarrow (S,)^+ S$

$$S(, S)^+ \equiv (S,)^+ S$$

Grading Guidelines: *Full credit for showing both parse trees to justify rotation.*
If derivation is used for the equivalence proof, all steps to get the regular expression of the sentential forms are needed for full credit. For part of derivation, 1 mark would be deducted if the expressions are correct. Otherwise, more marks are deducted.
We can also show $L(G_r) \subseteq L(G)$ and $L(G) \subseteq L(G_r)$. each direction would earn 1.5 marks.
END SOLUTION

8. Consider a string $x = (a, (a, a), (a))$

(a) Write a leftmost derivation for x using G_r (Q7). **[5]**

BEGIN SOLUTION

Given

$$\begin{aligned} S &\rightarrow (\ L\ ) \\ S &\rightarrow a \\ L &\rightarrow S\ L' \\ L' &\rightarrow ,\ S\ L' \\ L' &\rightarrow \epsilon \end{aligned}$$

the leftmost derivation is:

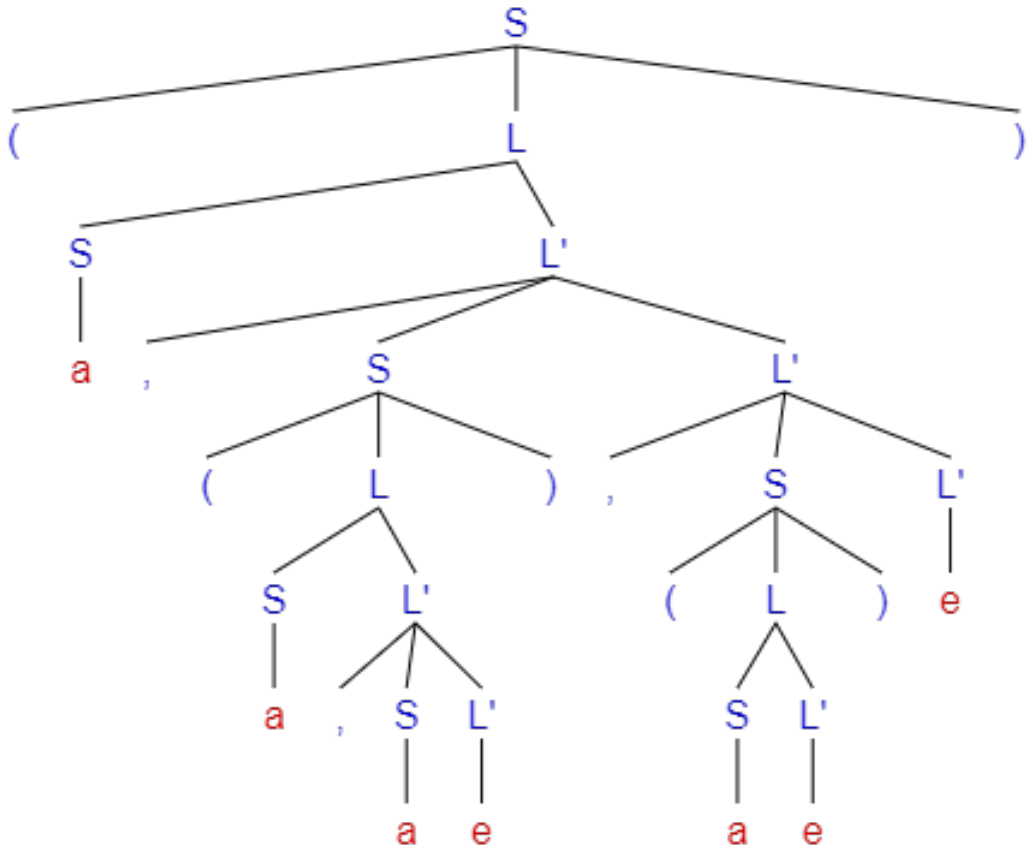
$$
\begin{aligned}
S\$ &\Rightarrow (L)\$ \\
&\Rightarrow (SL')\$ \\
&\Rightarrow (aL')\$ \\
&\Rightarrow (a, SL')\$ \\
&\Rightarrow (a, (L)L')\$ \\
&\Rightarrow (a, (SL')L')\$ \\
&\Rightarrow (a, (aL')L')\$ \\
&\Rightarrow (a, (a, SL')L')\$ \\
&\Rightarrow (a, (a, aL')L')\$ \\
&\Rightarrow (a, (a, a)L')\$ \\
&\Rightarrow (a, (a, a), SL')\$ \\
&\Rightarrow (a, (a, a), (L)L')\$ \\
&\Rightarrow (a, (a, a), (SL')L')\$ \\
&\Rightarrow (a, (a, a), (aL')L')\$ \\
&\Rightarrow (a, (a, a), (a)L')\$ \\
&\Rightarrow (a, (a, a), (a))\$ \\
&= x\$
\end{aligned}
$$

Grading Guidelines: *0.3 mark each for every step in the derivation (capped at 5). Full credit if at most one step is wrong.*

END SOLUTION

(b) Draw a parse tree for x using G_r (Q7). [**8**]

BEGIN SOLUTION



Grading Guidelines: *0.5 mark for each expansion of the nodes.*

END SOLUTION

9. Write a recursive descent parser for G_r (Q7) using C-like pseudo-code which is called from the following `main()` driver and uses the `match()` function. [**10**]

Be careful about handling the epsilon production. Else you would get into an infinite recursion.

```
int main() {
    l = getchar(); // lookahead
    S(); // S is the start symbol

    // End of the string if l = $
    if (l == '$')
        printf("Parsing Successful");
    else
        printf("Error");
}
match(char t) { // Match function: Matches and consumes
    if (l == t) {
        l = getchar();
    }
    else
        printf("Error");
}
```

BEGIN SOLUTION

```
S() { // S -> ( L )
    // Mark = 2
    if (l == '(')
        match('(');
        L();
        match(')');
    else
        // Mark = 2
        // S -> a
        match('a');
}

L() { // L -> S L'
    // Mark = 2
    S();
    L'();
}

L'() {
    // Mark = 2
    if (l == ',') // L' -> , S L'
        match(',');
        S();
        L'();
    else

        // Mark = 2
        return; // L' -> epsilon
}
```

Grading Guidelines: *The pseudo-code is segmented into 5 parts having 2 marks each. No partial marking within a part.*

The complete C code is presented as a reference and for understanding (not expected in the answer):

```
#include <stdio.h>
void S();
void S_prime();
char l;

int main() {
    l = getchar(); // lookahead
    S(); // S is the start symbol

    // End of the string if l = $
    if (l == '$')
        printf("Parsing Successful");
    else
        printf("Error");
}
void match(char t) { // Match function: Matches and consumes
    if (l == t) {
        l = getchar();
    }
    else
        printf("Error");
}
void S() { // S -> ( L )
    if (l == '(') {
        match('(');
        L();
        match(')');
    }
    else
        // S -> a
        match('a');
}
L() { // L -> S L'
    S();
    L_prime();
}
L_prime() {
    // Mark = 2
    if (l == ',') { // L' -> , S L'
        match(',');
        S();
        L_prime();
    }
    else
        // Mark = 2
        return; // L' -> epsilon
}
```

END SOLUTION

10. A bottom-up LR parser table for G is given below:

State	Action					GOTO	
	a	(	)	,	\$	S	L
1	s2	s3				4	
2	r2	r2	r2	r2	r2		
3	s2					7	5
4					acc		
5			s6	s8			
6	r1	r1	r1	r1	r1		
7	r4	r4	r4	r4	r4		
8	s2	s3				9	
9	r3	r3	r3	r3	r3		

where the production rules are:

1: $S \rightarrow (L)$
2: $S \rightarrow a$
3: $L \rightarrow L, S$
4: $L \rightarrow S$

Parse $x = (a, (a, a), (a))$ using the parse table. Show the parsing steps in the following format: [**30**]

Step	Stack Of		Input	Act.
	States	Symbols		
(1)	1		$(a, (a, a), (a))$\$	$\cdots$
(2)	$\cdots$	$\cdots$	$\cdots$	$\cdots$
		$\cdots$		
		$\cdots$		
		$\cdots$		
$\cdots$	1 4	S	\$	acc

BEGIN SOLUTION

Given:

State	Action					GOTO	
	a	(	)	,	\$	S	L
1	s2	s3				4	
2	r2	r2	r2	r2	r2		
3	s2					7	5
4					acc		
5			s6	s8			
6	r1	r1	r1	r1	r1		
7	r4	r4	r4	r4	r4		
8	s2	s3				9	
9	r3	r3	r3	r3	r3		

where the production rules are:

1: $S \rightarrow (L)$
2: $S \rightarrow a$
3: $L \rightarrow L, S$
4: $L \rightarrow S$

The parsing trace is as follows:

Step	Stack	Symbols	Input	Act.
(1)	1		$(a,(a,a),(a))\$$	s3
(2)	1 3	$($	$a,(a,a),(a))\$$	s2
(3)	1 3 2	$(a$	$,(a,a),(a))\$$	r2
(4)	1 3 7	$(S$	$,(a,a),(a))\$$	r4
(5)	1 3 5	$(L$	$,(a,a),(a))\$$	s8
(6)	1 3 5 8	$(L,$	$(a,a),(a))\$$	s3
(7)	1 3 5 8 3	$(L,($	$a,a),(a))\$$	s2
(8)	1 3 5 8 3 2	$(L,(a$	$,a),(a))\$$	r2
(9)	1 3 5 8 3 7	$(L,(S$	$,a),(a))\$$	r4
(10)	1 3 5 8 3 5	$(L,(L$	$,a),(a))\$$	s8
(11)	1 3 5 8 3 5 8	$(L,(L,$	$a),(a))\$$	s2
(12)	1 3 5 8 3 5 8 2	$(L,(L,a$	$),(a))\$$	r2
(13)	1 3 5 8 3 5 8 9	$(L,(L,S$	$),(a))\$$	r3
(14)	1 3 5 8 3 5	$(L,(L$	$),(a))\$$	s6
(15)	1 3 5 8 3 5 6	$(L,(L)$	$,(a))\$$	r1
(16)	1 3 5 8 9	$(L,S$	$,(a))\$$	r3
(17)	1 3 5	$(L$	$,(a))\$$	s8
(18)	1 3 5 8	$(L,$	$(a))\$$	s3
(19)	1 3 5 8 3	$(L,($	$a))\$$	s2
(20)	1 3 5 8 3 2	$(L,(a$	$))\$$	r2
(21)	1 3 5 8 3 7	$(L,(S$	$))\$$	r4
(22)	1 3 5 8 3 5	$(L,(L$	$))\$$	s6
(23)	1 3 5 8 3 5 6	$(L,(L)$	$)\$$	r1
(24)	1 3 5 8 9	$(L,S$	$)\$$	r3
(25)	1 3 5	$(L$	$)\$$	s6
(26)	1 3 5 6	(L)	$\$$	r1
(27)	1 4	S	$\$$	acc

Grading Guidelines: *1 marks for each step. In a step, deduct 0.25 mark for each wrong state stack, wrong symbol stack, wrong input buffer, or wrong action.*

If a step is wrongly acted, it is likely that the following steps would be different and lead to a wrong end. In such a case, the first wrong step would get 0 while the remaining (forced wrong) steps, if correct, should be graded with full credit assuming as if that the DPDA has started on a fresh configuration.

Overall bonus of 1.5 marks to be awared if all steps are correctly done.

END SOLUTION

11. **Bonus Problem**: Is G ambiguous or unambiguous? **[2]**

Based on your answer above, attempt the appropriate question below: **[8]**

- G is ambiguous: Give an example string $y \in T^+$ which has two leftmost derivations and draw the corresponding parse trees.
- G is unambiguous: Prove that G is unambiguous using induction on the length of the strings in $L(G)$ or any other method you deem suitable.

*The credit for a bonus problem (**10** here) is not counted in the total of **100**. However, marks scored in a bonus problem will be added to total score (capped, of course, at **100**).*

BEGIN SOLUTION

G is unambiguous. Hence there cannot be ambiguity.

Theorem: G is unambiguous.

Proof: We proceed by induction on the length $n = |x|$ of $x \in L(G)$. We need to show that for any x we can construct a unique parse tree.

We start by defining an infinite collection of finite sets as follows:

$$L_n(G) = \{x \mid x \in L(G) \text{ and } |x| = n\}, \forall n \geq 1$$

That is, $L_n(G)$ is the set of strings of *length n* that can be derived from S. Let us also define the set $L'_n(G)$ of strings of *length up to n* that can be derived from S:

$$L'_n(G) = \bigcup_{m=1}^{n} L_m(G), \forall n \geq 1$$

We get,

$$\begin{aligned} L'_1(G) &= L_1(G) && (1)\\ L'_n(G) &= L'_{n-1}(G) \bigcup L_n(G), \forall n > 1 && (2)\\ L(G) &= \bigcup_{n=1}^{\infty} L_n(G) && (3) \end{aligned}$$

Clearly, Eqn. (1) & (2) lend to the induction while Eqn. (3) gives the language.

Observation: We have shown in Q5 that $\forall x \in L(G),\ |x|\ mod\ 2 = 1$. That is, $L(G)$ does not contain any even length string. Or $\forall n \geq 0,\ L_{2n}(G) = \{\}$.

$$\begin{aligned} L_1(G) &= \{a\}\\ L_3(G) &= \{(a)\}\\ L_5(G) &= \{(a,a),((a))\}\\ L_7(G) &= \{(a,a,a),((a,a)),(a,(a)),((a),a),(((a)))\}\\ &\dots \end{aligned}$$

Recursive Substitution: Note that from $L_5(G)$ onward, every string in $L_n(G)$ is obtained from a string in $L_{n-2}(G)$ by replacing an a:

- either by a, a.
- or by (a).

So we can see a simple recursive generator.

Hypothesis: $\mathcal{P}(n)$: Any $x \in L'_n(G)$ (having $|x| \leq n$) has a unique derivation $\forall n \geq 1$.

Basis:

$\mathcal{P}(1)$: Holds for $x \in L'_1(G)$ (having $|x| = 1$) as $x = a$ derived by $S \Rightarrow a$. However, we will not use this basis as it does not fit the recursive generator.

$\mathcal{P}(3)$: Holds for $x \in L'_3(G)$ (having $|x| = 3$) as $x = (a)$ derived by $S \Rightarrow (L) \Rightarrow (S) \Rightarrow (a)$.

Induction: We prove $\mathcal{P}(n+2)$ given that $\mathcal{P}(n)$ holds for any $n > 1,\ n\ \mod 2 = 1$.

Let $x \in L_{n+2}(G) \subseteq L'_{n+2}(G)$ ($|x| = n+2$). So there exists $y \in L_n(G) \subseteq L'_n(G)$ ($|y| = n$) in which one of the above substitutions have been effected. By the induction hypothesis, the parse tree for $S \Rightarrow^+ y$ is unique. The parse tree of $S \Rightarrow^+ x$ can be obtained from the parse tree for $S \Rightarrow^+ y$ as:

- *Case* $a \vdash a, a$: So in $S \Rightarrow^+ y$, there exists steps such that
  - $L \Rightarrow S \Rightarrow a$. This is replaced by $L \Rightarrow L, S \Rightarrow S, S \Rightarrow a, S \Rightarrow a, a$ to get $S \Rightarrow^+ x$.
  - $L \Rightarrow L, S \Rightarrow S, S \Rightarrow a, S$. This is replaced by $L \Rightarrow L, S \Rightarrow L, S, S \Rightarrow S, S, S \Rightarrow a, S, S \Rightarrow a, a, S$ to get $S \Rightarrow^+ x$.
- *Case* $a \vdash (a)$: So in $S \Rightarrow^+ y$, there exists steps such that $S \Rightarrow a$. This is replaced by $S \Rightarrow (L) \Rightarrow (S) \Rightarrow (a)$ to get $S \Rightarrow^+ x$.

Either of the above sequence of derivations steps to go from $S \Rightarrow^+ y$ to $S \Rightarrow^+ x$ are unique. Hence, $S \Rightarrow^+ x$ has a unique parse tree and G is unambiguous.

Grading Guidelines: *If G is stated to be unambiguous, 2 out of 2 to be awarded. If G is said to be ambiguous, 0 out of 2 to be awarded.*

Any correct inductive argument that handles the cases of $a \vdash a, a$ and $a \vdash (a)$ should get full credit. Explicit construction of sets like $L_n(G)$ or $L'_n(G)$ is not mandatory. Drawing the parse tree is also not mandatory.

Deduct 2 marks if basis statement is not stated. Deduct 2 marks if the hypothesis is not stated. Deduct 1 mark if the hypothesis is stated for $|x| = n$ instead of $|x| \leq n$. Deduct 2 marks if basis is not proved. Deduct 2 marks if the specific cases are not proved.

END SOLUTION