CS 181 HW3 2021 CS181

YIQIAO JIN

TOTAL POINTS

26 / 28

QUESTION 1

1 Pumping Lemma 5 / 7

+ **7 pts** Correct or nearly correct

✓ + **1 pts** Appropriate "s"

+ **2 pts** Effective use of constraints on "xyz"

✓ + **2 pts** Show coverage of all cases of "xyz"

- **0.5 pts** should give more complete justification that your proof covers all cases of xyz

✓ + **2 pts** Sound logic in every case

- **0.5 pts** Should clearly state that |xy| $\leq$ p means xy is all a's, not just mention that |xy| $\leq$ p.

- **0.5 pts** minor error in logic

- **1 pts** should give more complete justification logic

- **2 pts** You cannot assign/assume any particular value for x, y, or z.

- **1 pts** Cannot assume p is even

+ **0 pts** No answer

-2 for use of constraints because the three cases are not necessary.

QUESTION 2

2 Regular Expression 4 / 4

✓ - **0 pts** Correct

- **1 pts** Almost correct

- **3 pts** Not correct

- **4 pts** Not Attempted

QUESTION 3

3 NFA 5 / 5

✓ - **0 pts** Correct

- **1 pts** Minor mistakes, your NFA cannot accept some of the strings in the language. We use following strings to test your NFA.

- ababccbcccc
- ababcccc
- abababcccc
- ababbcccc
- ababbacccc
- babaacccc
- babaabcccc
- babaaccccabba
- ababbaccccbaab

- **1 pts** Minor mistakes, your NFA will accept some strings that are not in the language, for example,

- ababccc
- babaabab
- ababbaba
- ccccbaba
- ccccabab
- ccccaababa
- ccccbbabab

- **1 pts** Does not effectively use the nondeterminism in the NFA

- **1 pts** Invalid NFA, or your NFA does not satisfy the definition of NFA, or do not explicitly specify the final states, or forget to specify the transition for some states.

- **5 pts** Totally wrong, or you did not answer this problem at all.

QUESTION 4

Parse Trees & Derivations 4 pts

4.1 a Parse Tree 1 / 1

✓ - **0 pts** Correct

- **0.5 pts** Small mistake

- **1 pts** Incorrect

4.2 b Left-most Derivation 1 / 1

✓ **- 0 pts** Correct

- 0.5 pts Missing Steps

- 0.5 pts Not Left-most

- 1 pts Incorrect

4.3 c Parse Tree 1 / 1

✓ **- 0 pts** Correct

- 0.5 pts Small Mistake

- 1 pts Incorrect

4.4 d Left-most Derivation 1 / 1

✓ **- 0 pts** Correct

- 0.5 pts Missing Steps

- 0.5 pts Not Left-most

- 1 pts Incorrect

QUESTION 5

5 CFG; 4 / 4

✓ **- 0 pts** Correct

- 1 pts Can not express arbitrary number of begin end blocks beside each other, e.g. bbs;e;bs;e;bs;e;e;

- 2 pts Begin and end blocks don't have to line up. e.g. bbs;e; or bs;e;e; is generated even though it shouldn't be.

- 1 pts No specified start variable

- 1 pts Extra semicolons generated (e.g. bs;;bs;e;e; or b;s;e;)

- 1 pts Can not express multiple statements beside each other (e.g. bs;s;s;e;)

- 1 pts Missing semicolons on s (e.g. bse; is generated)

- 1 pts Can not express single statement e.g. bs;e;

- 1 pts Can't do some orders of statements and blocks e.g. bbs;e;s;e; or bs;bs;e;e;

- 1 pts Can't have arbitrary nestings next to each other (like bbbs;e;e;bbs;e;e;e;

- 1 pts Doesn't necessarily have an outside begin end pair (e.g. generates s; or b or nothing at all, or bs;e;bs;e;)

- 1 pts There isn't necessarily an outer be pair (e.g. bs;e;bs;e; can be generated)

- 0 pts Click here to replace this description.

QUESTION 6

6 CFG, 4 / 4

✓ **- 0 pts** Correct

- 1 pts Adjacent begin statements either can't exist (can't generate bbse,bsee) or can be missing commas between (e.g. bbsebsee)

- 1 pts Can't generate arbitrarily many adjacent (or nested) begin/ends (e.g. bbse,bse,bsee or bbsee or bs,bse,bsee)

- 2 pts zBegin/end not guaranteed to match (e.g. could generate bsee or bssee)

- 1 pts No specified start variable

- 1 pts Unnecessary semicolons

- 1 pts Doesn't necessarily generate outside begin end pair (e.g. bsebse or bse,bse or s or s, or epsilon)

- 1 pts Can't generate certain orders of statements, for example bbse,se or bs,bsee

- 1 pts begin/end statements can be empty (e.g. generates be)

- 1 pts Could be missing commas (e.g. this can generate bsse or bss,se or bbsese)

- 1 pts Can't generate arbitrarily many s's (e.g. bs,s,se)

- 1 pts Can generate extra commas (e.g. bse, or bs,e or b,,,,se)

- 1 pts Can't do arbitrary nesting of begin/end statements (e.g. bbbseee

QUESTION 7

7 Postponed to next weeK: GNFA 0 / 0

✓ **- 0 pts** Correct

Homework 3

Name: Yiqiao Jin
UID: 305107551

1

We prove that $L = \{a^{2n}b^n | n \geq 0\}$ is not regular by contradiction.

Suppose L is an FSL. Let p be the pumping length. So we can choose $s = a^{2n}b^n \in L$. Assume L is regular. Here, s can be written as $s = xyz$, the concatenation of some substrings x, y, z, where:

1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| = m > 0$
3. $|xy| \leq p$

We consider 3 cases for the formation of y:

1a

The string y consists only of a's. In this case, the number of a's in the string $xyyz$ is more than $2n$, but the number of b's remains the same (n). So $xyyz$ is not a member of L, which violates condition 1 of the Pumping Lemma. This case is a contradiction.

1b

The string y consists only of b's. In this case, the number of a's in the string $xyyz$ remains $2n$. However, the number of b's $> n$. So $xyyz$ is still not a member of L, which violates condition 1 of the Pumping Lemma.

1c

The string y consists of both a's and b's. In this case, it is possible that within the string $xyyz$, the number of a's is twice the number of b's, specifically, when $y = a^{2m}b^m$ for some $m > 0$). But they will be out of order with some b's before a's. Hence $xyyz$ is still not a member of L, which is a contradiction.

From 1a-c, we cannot avoid the contradiction if we assume that L is regular, so L is not regular.

1 Pumping Lemma 5 / 7

+ **7 pts** Correct or nearly correct

✓ + **1 pts** **Appropriate "s"**

+ **2 pts** Effective use of constraints on "xyz"

✓ + **2 pts** **Show coverage of all cases of "xyz"**

- **0.5 pts** should give more complete justification that your proof covers all cases of xyz

✓ + **2 pts** **Sound logic in every case**

- **0.5 pts** Should clearly state that |xy| $\leq$ p means xy is all a's, not just mention that |xy| $\leq$ p.

- **0.5 pts** minor error in logic

- **1 pts** should give more complete justification logic

- **2 pts** You cannot assign/assume any particular value for x, y, or z.

- **1 pts** Cannot assume p is even

+ **0 pts** No answer

-2 for use of constraints because the three cases are not necessary.

2

Let $\Sigma = \{a, b, c\}$.

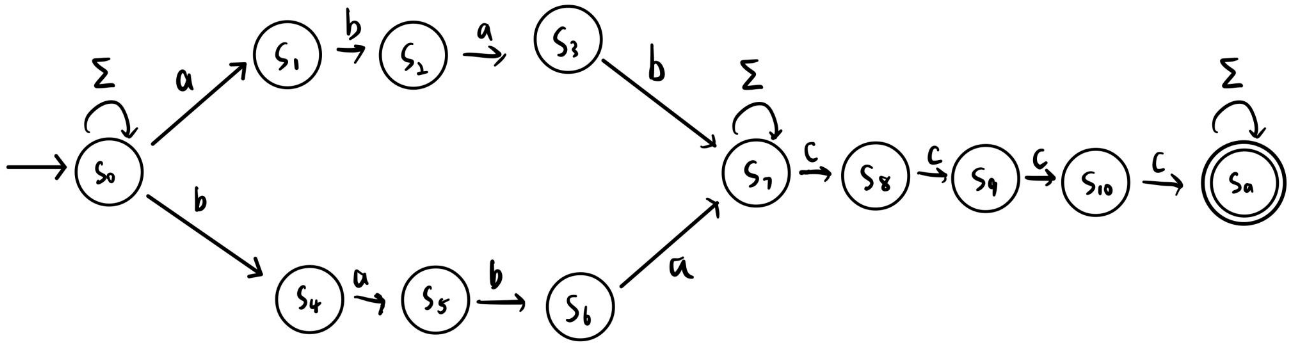
$L_2 = (a(\Sigma)^*(\Sigma \setminus \{a\})) \cup (b(\Sigma)^*(\Sigma \setminus \{b\})) \cup (c(\Sigma)^*(\Sigma \setminus \{c\}))$

The $(\Sigma)^*$ in the middle requires that the arbitrary symbols between the start symbol and end symbol can appear any times in $[0, \infty)$.

The a at the beginning of the string and $(\Sigma \setminus \{a\})$ at the end of the string require that the start and end symbols are different. The same is true for b and c

3

The following NFA recognizes L_3



The above diagram shows that the NFA recognizes strings with the following pattern:

$\Sigma^*(abab \cup baba)\Sigma^* cccc\Sigma^*$, where $\Sigma = (a, b, c)$.

At the beginning of the string, we non-deterministically loop on Σ before we detect the start of substring $abab$ (which is a) and $baba$ (which is b). This means any characters in (a, b, c) are acceptable before we recognize the substring.

We then move onto either of branches representing $abab$ and $baba$ by transition into either S_1 or S_4. After we continuously read the 4 symbols in the substring and before we read the $cccc$, we non-deterministically loop on Σ at S_7. Note that substring like $babab$ is acceptable for both branches, and we can transition into either S_1 or S_4 non-deterministically.

Then, in S_7 to S_{10}, we try to detect $cccc$. Finally, we transition into the final state S_a and non-deterministically loop on Σ since the string has already satisfied all of its requirements.

4

2 Regular Expression 4 / 4

✓ **- 0 pts** Correct

- 1 pts Almost correct

- 3 pts Not correct

- 4 pts Not Attempted

2

Let $\Sigma = \{a, b, c\}$.

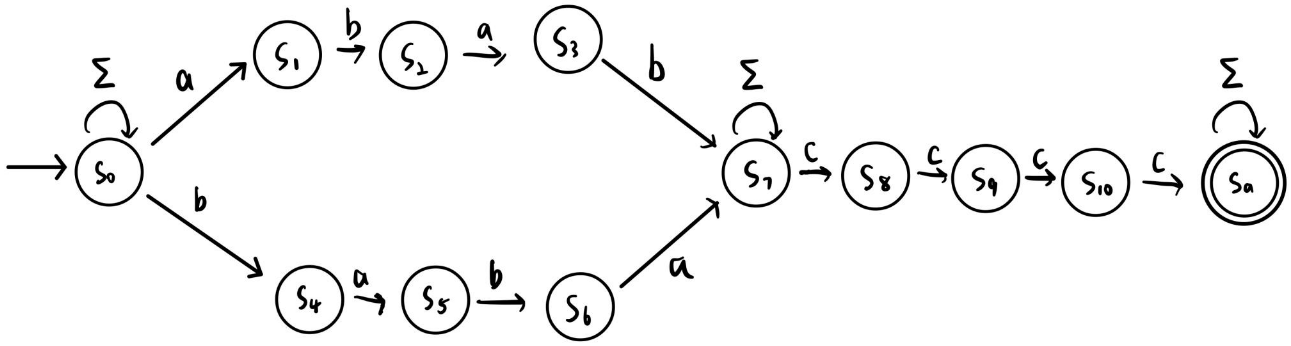
$L_2 = (a(\Sigma)^*(\Sigma \setminus \{a\})) \cup (b(\Sigma)^*(\Sigma \setminus \{b\})) \cup (c(\Sigma)^*(\Sigma \setminus \{c\}))$

The $(\Sigma)^*$ in the middle requires that the arbitrary symbols between the start symbol and end symbol can appear any times in $[0, \infty)$.

The a at the beginning of the string and $(\Sigma \setminus \{a\})$ at the end of the string require that the start and end symbols are different. The same is true for b and c

3

The following NFA recognizes L_3



The above diagram shows that the NFA recognizes strings with the following pattern:

$\Sigma^*(abab \cup baba)\Sigma^* cccc\Sigma^*$, where $\Sigma = (a, b, c)$.

At the beginning of the string, we non-deterministically loop on Σ before we detect the start of substring $abab$ (which is a) and $baba$ (which is b). This means any characters in (a, b, c) are acceptable before we recognize the substring.

We then move onto either of branches representing $abab$ and $baba$ by transition into either S_1 or S_4. After we continuously read the 4 symbols in the substring and before we read the $cccc$, we non-deterministically loop on Σ at S_7. Note that substring like $babab$ is acceptable for both branches, and we can transition into either S_1 or S_4 non-deterministically.

Then, in S_7 to S_{10}, we try to detect $cccc$. Finally, we transition into the final state S_a and non-deterministically loop on Σ since the string has already satisfied all of its requirements.

4

3 NFA 5 / 5

✓ **- 0 pts** Correct

- 1 pts Minor mistakes, your NFA cannot accept some of the strings in the language. We use following strings to test your NFA.

- ababccbcccc
- ababcccc
- ababacccc
- ababbcccc
- ababbacccc
- babaacccc
- babaabcccc
- babaaccccabba
- ababbaccccbaab

- 1 pts Minor mistakes, your NFA will accept some strings that are not in the language, for example,

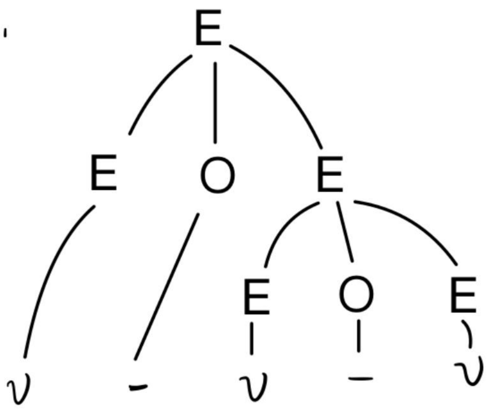
- ababccc
- babaabab
- ababbaba
- ccccbaba
- ccccabab
- ccccaababa
- ccccbbabab

- 1 pts Does not effectively use the nondeterminism in the NFA

- 1 pts Invalid NFA, or your NFA does not satisfy the definition of NFA, or do not explicitly specify the final states, or forget to specify the transition for some states.

- 5 pts Totally wrong, or you did not answer this problem at all.

a.



b.

$E \to \underline{E}OE$

$\to v\,\underline{O}E$

$\to v - \underline{E}$

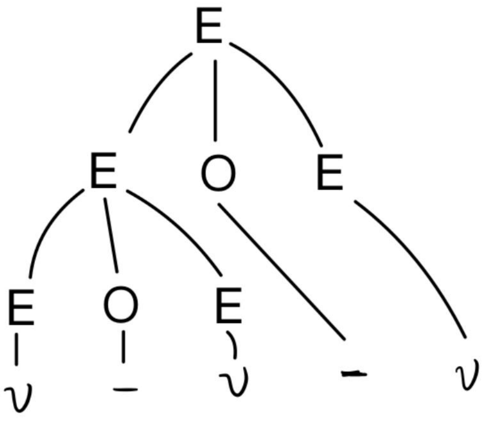
$\to v - \underline{E}OE$

$\to v - v\,\underline{O}E$

$\to v - v - \underline{E}$

$\to v - v - v$

c.



d.

$E \to EO\underline{E}$

$\to \underline{E}OEOE$

$\to v\,\underline{O}EOE$

$\to v - \underline{E}OE$

$\to v - v\,\underline{O}E$

$\to v - v - \underline{E}$

$\to v - v - v$

5

Let $\Sigma = \{b, e, s, ;\}$. Then L_5 is specified by the grammar G:

$\mathbf{S} \to b\mathbf{A}e;$ (Rule 1)

$\mathbf{A} \to b\mathbf{A}e;|s;|\mathbf{AA}$ (Rule 2)

We use $\mathbf{bold}$ capital letters to represent nonterminal symbols, and lowercase letters to represent terminal symbols.

Rule 1 specifies that every string is generated from the start variable $\mathbf{S}$. It must begin with b and end with $e;$

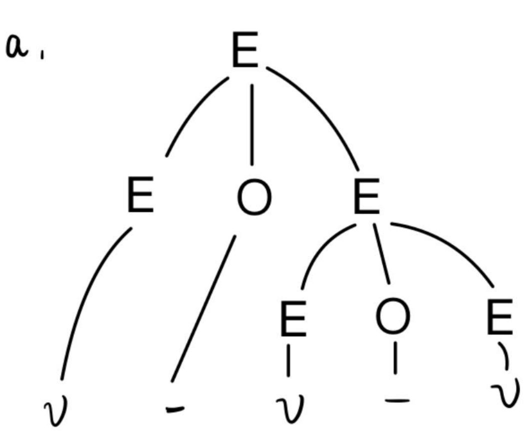
Rule 2 specifies the rules $\mathbf{A}$ uses to produce substrings. $\mathbf{A}$ can perform either of the following:

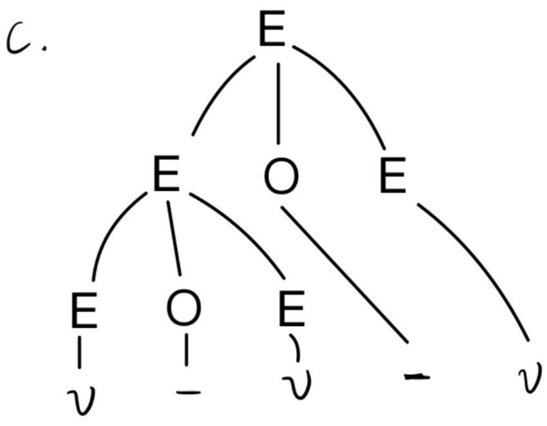
- Spawn a new begin-end statement pair, followed by $';'$
- Generate a single statement $s;$ (this is a terminal)
- Generate two statements $\mathbf{A}$ (variables, or nonterminals), separated by $';'$

6

4.1 a Parse Tree **1 / 1**

- ✓ **- 0 pts** **Correct**
- **- 0.5 pts** Small mistake
- **- 1 pts** Incorrect





b.

$E \rightarrow \underline{E}OE$

$\rightarrow v\,\underline{O}E$

$\rightarrow v - \underline{E}$

$\rightarrow v - \underline{E}OE$

$\rightarrow v - v\,\underline{O}E$

$\rightarrow v - v - \underline{E}$

$\rightarrow v - v - v$

d.

$E \rightarrow EO\underline{E}$

$\rightarrow \underline{E}OEOE$

$\rightarrow v\,\underline{O}\,EOE$

$\rightarrow v - \underline{E}OE$

$\rightarrow v - v\,\underline{O}E$

$\rightarrow v - v - \underline{E}$

$\rightarrow v - v - v$

5

Let $\Sigma = \{b, e, s, ;\}$. Then L_5 is specified by the grammar G:

$\mathbf{S} \rightarrow b\mathbf{A}e;$ (Rule 1)

$\mathbf{A} \rightarrow b\mathbf{A}e;|s;|\mathbf{AA}$ (Rule 2)

We use $\mathbf{bold}$ capital letters to represent nonterminal symbols, and lowercase letters to represent terminal symbols.

Rule 1 specifies that every string is generated from the start variable $\mathbf{S}$. It must begin with b and end with $e;$

Rule 2 specifies the rules $\mathbf{A}$ uses to produce substrings. $\mathbf{A}$ can perform either of the following:

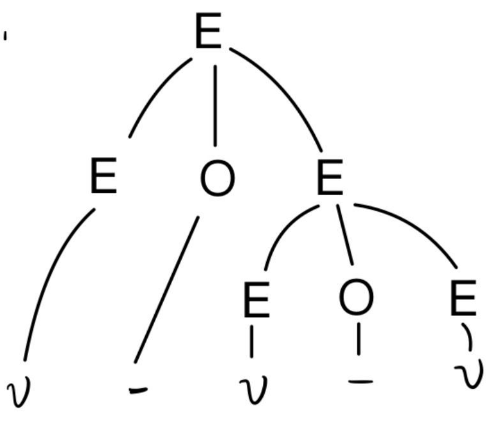
- Spawn a new begin-end statement pair, followed by $';'$
- Generate a single statement $s;$ (this is a terminal)
- Generate two statements $\mathbf{A}$ (variables, or nonterminals), separated by $';'$

6

4.2 b Left-most Derivation 1 / 1

- ✓ **- 0 pts** Correct
- **- 0.5 pts** Missing Steps
- **- 0.5 pts** Not Left-most
- **- 1 pts** Incorrect

a.



b.

$E \rightarrow \underline{E}OE$

$\rightarrow v\,\underline{O}E$

$\rightarrow v - \underline{E}$

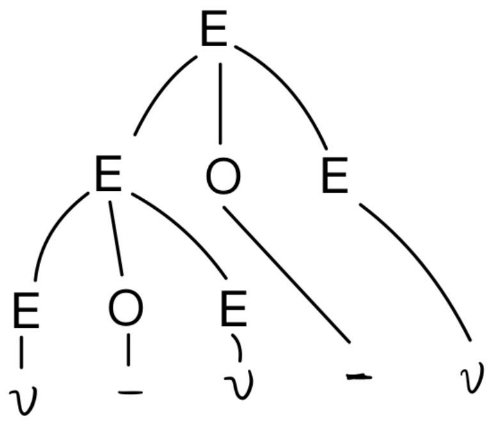
$\rightarrow v - \underline{E}OE$

$\rightarrow v - v\,\underline{O}E$

$\rightarrow v - v - \underline{E}$

$\rightarrow v - v - v$

c.



d.

$E \rightarrow EO\underline{E}$

$\rightarrow \underline{E}OEOE$

$\rightarrow v\,\underline{O}EOE$

$\rightarrow v - \underline{E}OE$

$\rightarrow v - v\,\underline{O}E$

$\rightarrow v - v - \underline{E}$

$\rightarrow v - v - v$

5

Let $\Sigma = \{b, e, s, ;\}$. Then L_5 is specified by the grammar G:

$\mathbf{S} \rightarrow b\mathbf{A}e;$ (Rule 1)

$\mathbf{A} \rightarrow b\mathbf{A}e; | s; | \mathbf{AA}$ (Rule 2)

We use $\mathbf{bold}$ capital letters to represent nonterminal symbols, and lowercase letters to represent terminal symbols.

Rule 1 specifies that every string is generated from the start variable $\mathbf{S}$. It must begin with b and end with $e;$

Rule 2 specifies the rules $\mathbf{A}$ uses to produce substrings. $\mathbf{A}$ can perform either of the following:

- Spawn a new begin-end statement pair, followed by $';'$
- Generate a single statement $s;$ (this is a terminal)
- Generate two statements $\mathbf{A}$ (variables, or nonterminals), separated by $';'$

6

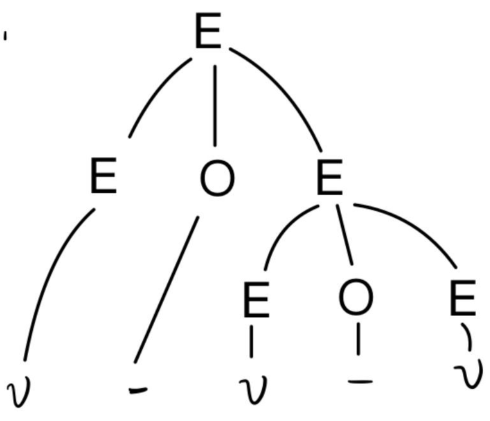
4.3 c Parse Tree 1 / 1

✓ - **0 pts** Correct

- **0.5 pts** Small Mistake

- **1 pts** Incorrect

a.



b.

$E \to \underline{E}OE$

$\to v\,\underline{O}\,E$

$\to v - \underline{E}$

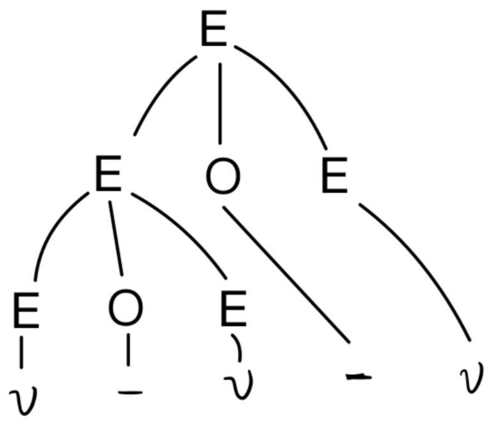
$\to v - \underline{E}OE$

$\to v - v\,\underline{O}E$

$\to v - v - \underline{E}$

$\to v - v - v$

c.



d.

$E \to EO\underline{E}$

$\to \underline{E}OEOE$

$\to v\,\underline{O}\,EOE$

$\to v - \underline{E}OE$

$\to v - v\,\underline{O}\,E$

$\to v - v - \underline{E}$

$\to v - v - v$

5

Let $\Sigma = \{b, e, s, ;\}$. Then L_5 is specified by the grammar G:

$\mathbf{S} \to b\mathbf{A}e;$ (Rule 1)

$\mathbf{A} \to b\mathbf{A}e; | s; | \mathbf{AA}$ (Rule 2)

We use $\mathbf{bold}$ capital letters to represent nonterminal symbols, and lowercase letters to represent terminal symbols.

Rule 1 specifies that every string is generated from the start variable $\mathbf{S}$. It must begin with b and end with $e;$

Rule 2 specifies the rules $\mathbf{A}$ uses to produce substrings. $\mathbf{A}$ can perform either of the following:

- Spawn a new begin-end statement pair, followed by $';'$
- Generate a single statement $s;$ (this is a terminal)
- Generate two statements $\mathbf{A}$ (variables, or nonterminals), separated by $';'$

6

4.4 d Left-most Derivation **1 / 1**

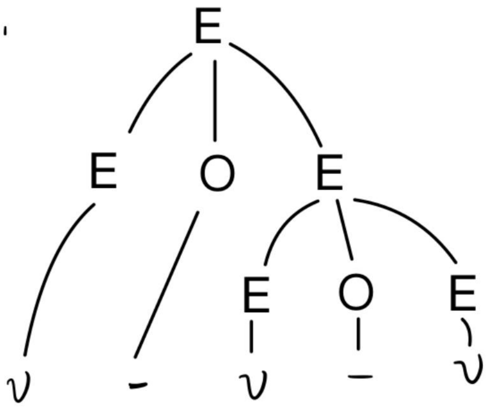
✓ **- 0 pts** **Correct**

- 0.5 pts Missing Steps

- 0.5 pts Not Left-most

- 1 pts Incorrect

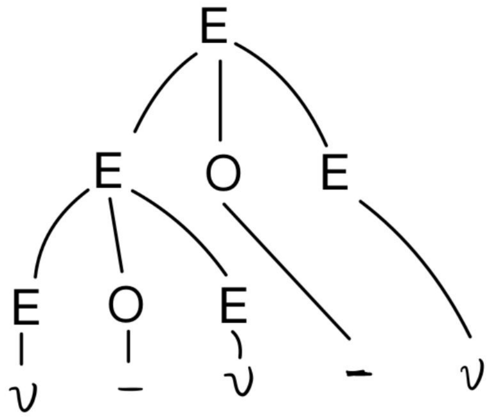
a.



b.

$$\begin{aligned} E &\rightarrow \underline{E}OE \\ &\rightarrow v\,\underline{O}E \\ &\rightarrow v - \underline{E} \\ &\rightarrow v - \underline{E}OE \\ &\rightarrow v - v\,\underline{O}E \\ &\rightarrow v - v - \underline{E} \\ &\rightarrow v - v - v \end{aligned}$$

c.



d.

$$\begin{aligned} E &\rightarrow EO\underline{E} \\ &\rightarrow \underline{E}OEOE \\ &\rightarrow v\,\underline{O}EOE \\ &\rightarrow v - \underline{E}OE \\ &\rightarrow v - v\,\underline{O}E \\ &\rightarrow v - v - \underline{E} \\ &\rightarrow v - v - v \end{aligned}$$

5

Let $\Sigma = \{b, e, s, ;\}$. Then L_5 is specified by the grammar G:

$\mathbf{S} \rightarrow b\mathbf{A}e;$ (Rule 1)

$\mathbf{A} \rightarrow b\mathbf{A}e; | s; | \mathbf{AA}$ (Rule 2)

We use $\mathbf{bold}$ capital letters to represent nonterminal symbols, and lowercase letters to represent terminal symbols.

Rule 1 specifies that every string is generated from the start variable $\mathbf{S}$. It must begin with b and end with $e;$

Rule 2 specifies the rules $\mathbf{A}$ uses to produce substrings. $\mathbf{A}$ can perform either of the following:

- Spawn a new begin-end statement pair, followed by $';'$
- Generate a single statement $s;$ (this is a terminal)
- Generate two statements $\mathbf{A}$ (variables, or nonterminals), separated by $';'$

6

5 CFG; 4 / 4

✓ **- 0 pts** Correct

- 1 pts Can not express arbitrary number of begin end blocks beside each other, e.g. bbs;e;bs;e;bs;e;e;

- 2 pts Begin and end blocks don't have to line up. e.g. bbs;e; or bs;e;e; is generated even though it shouldn't be.

- 1 pts No specified start variable

- 1 pts Extra semicolons generated (e.g. bs;;bs;e;e; or b;s;e;)

- 1 pts Can not express multiple statements beside each other (e.g. bs;s;s;e;)

- 1 pts Missing semicolons on s (e.g. bse; is generated)

- 1 pts Can not express single statement e.g. bs;e;

- 1 pts Can't do some orders of statements and blocks e.g. bbs;e;s;e; or bs;bs;e;e;

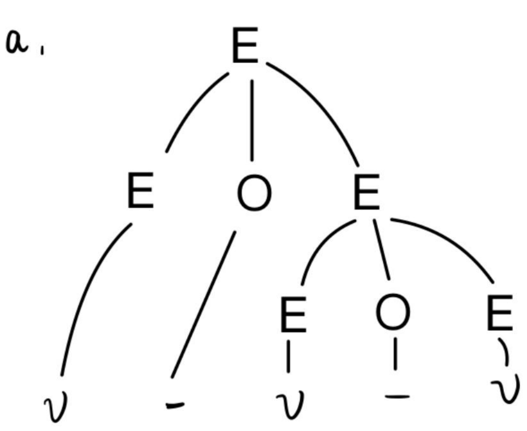
- 1 pts Can't have arbitrary nestings next to each other (like bbbs;e;e;bbs;e;e;e;

- 1 pts Doesn't necessarily have an outside begin end pair (e.g. generates s; or b or nothing at all, or bs;e;bs;e;)

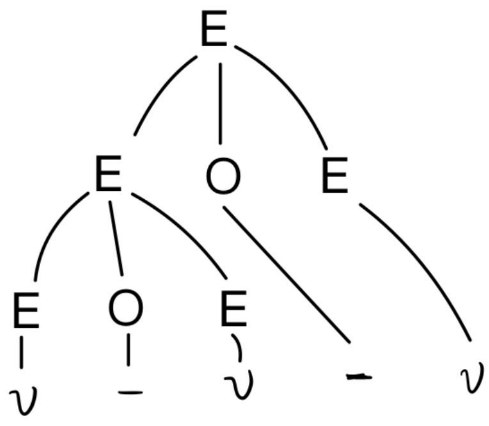
- 1 pts There isn't necessarily an outer be pair (e.g. bs;e;bs;e; can be generated)

- 0 pts Click here to replace this description.

a.



c.



b.

$E \rightarrow \underline{E}OE$

$\rightarrow v\underline{O}E$

$\rightarrow v - \underline{E}$

$\rightarrow v - \underline{E}OE$

$\rightarrow v - v\underline{O}E$

$\rightarrow v - v - \underline{E}$

$\rightarrow v - v - v$

d.

$E \rightarrow EO\underline{E}$

$\rightarrow \underline{E}OEOE$

$\rightarrow v\underline{O}EOE$

$\rightarrow v - \underline{E}OE$

$\rightarrow v - v\underline{O}E$

$\rightarrow v - v - \underline{E}$

$\rightarrow v - v - v$

5

Let $\Sigma = \{b, e, s, ;\}$. Then L_5 is specified by the grammar G:

$\mathbf{S} \rightarrow b\mathbf{A}e;$ (Rule 1)

$\mathbf{A} \rightarrow b\mathbf{A}e; |s; |\mathbf{AA}$ (Rule 2)

We use $\mathbf{bold}$ capital letters to represent nonterminal symbols, and lowercase letters to represent terminal symbols.

Rule 1 specifies that every string is generated from the start variable $\mathbf{S}$. It must begin with b and end with $e;$

Rule 2 specifies the rules $\mathbf{A}$ uses to produce substrings. $\mathbf{A}$ can perform either of the following:

- Spawn a new begin-end statement pair, followed by $';'$
- Generate a single statement $s;$ (this is a terminal)
- Generate two statements $\mathbf{A}$ (variables, or nonterminals), separated by $';'$

6

G is a Context-Free Grammar for the language L_6

$\mathbf{S} \to b\mathbf{A}e$ (Rule 1)

$\mathbf{A} \to b\mathbf{A}e|s|\mathbf{A},\mathbf{A}$ (Rule 2)

In Rule 1, the symbol $\mathbf{S}$ is the start variable. This guarantee that all strings generated are enclosed in a pair of beginning and ending symbol b and e.

The second rule specifies that every new string spawned by $\mathbf{A}$ can be one of

- Some string generated by $\mathbf{A}$, enclosed in a begin-end block
- A single statement s
- Two new strings generated by $\mathbf{A}$, separated by $','$.

7

The string $aaab$ can be accepted by the following ways:

$q_{start} \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{ab} q_{accept}$

$q_{start} \xrightarrow{\varepsilon} q_2 \xrightarrow{aa} q_1 \xrightarrow{ab} q_{accept}$

(Note: there is a 3rd way):

$q_{start} \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{aa} q_1 \xrightarrow{ab} q_{accept}$

6 CFG, **4 / 4**

✓ **- 0 pts** Correct

- 1 pts Adjacent begin statements either can't exist (can't generate bbse,bsee) or can be missing commas between (e.g. bbsebsee)

- 1 pts Can't generate arbitrarily many adjacent (or nested) begin/ends (e.g. bbse,bse,bsee or bbsee or bs,bse,bsee)

- 2 pts zBegin/end not guaranteed to match (e.g. could generate bsee or bssee)

- 1 pts No specified start variable

- 1 pts Unnecessary semicolons

- 1 pts Doesn't necessarily generate outside begin end pair (e.g. bsebse or bse,bse or s or s, or epsilon)

- 1 pts Can't generate certain orders of statements, for example bbse,se or bs,bsee

- 1 pts begin/end statements can be empty (e.g. generates be)

- 1 pts Could be missing commas (e.g. this can generate bsse or bss,se or bbsese)

- 1 pts Can't generate arbitrarily many s's (e.g. bs,s,se)

- 1 pts Can generate extra commas (e.g. bse, or bs,e or b,,,,se)

- 1 pts Can't do arbitrary nesting of begin/end statements (e.g. bbbseee

G is a Context-Free Grammar for the language L_6

$\mathbf{S} \rightarrow b\mathbf{A}e$ (Rule 1)

$\mathbf{A} \rightarrow b\mathbf{A}e|s|\mathbf{A},\mathbf{A}$ (Rule 2)

In Rule 1, the symbol $\mathbf{S}$ is the start variable. This guarantee that all strings generated are enclosed in a pair of beginning and ending symbol b and e.

The second rule specifies that every new string spawned by $\mathbf{A}$ can be one of

- Some string generated by $\mathbf{A}$, enclosed in a begin-end block
- A single statement s
- Two new strings generated by $\mathbf{A}$, separated by $','$.

7

The string $aaab$ can be accepted by the following ways:

$q_{start} \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{ab} q_{accept}$

$q_{start} \xrightarrow{\varepsilon} q_2 \xrightarrow{aa} q_1 \xrightarrow{ab} q_{accept}$

(Note: there is a 3rd way):

$q_{start} \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{aa} q_1 \xrightarrow{ab} q_{accept}$

7 Postponed to next weeK: GNFA 0 / 0

✓ - 0 pts Correct