Gradiance Online Accelerated Learning

Zayd

- Home Page
- Assignments Due
- Progress Report
- Handouts
- Tutorials
- Homeworks
- Lab Projects
- Log Out

Help

Submission number:	71347
Submission certificate:	BA748834
Submission time:	2014-03-22 18:45:49 PST (GMT - 8:00)

Number of questions:	4
Positive points per question:	3.0
Negative points per question:	1.0
Your score:	8

These are questions based on Section 7.3 of HMU that were not selected for the main homework on the topic.

1. The homomorphism *h* is defined by h(a) = 01 and h(b) = 10. What is h(babb)?

a) 10101010

b) 100110

c) babb

d) 10011010

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

We must concatenate h(b), h(a), h(b), and h(b), in that order. These strings are 10, 01, 10, and 10, respectively, so the answer is 10011010.

2. Let L_1 and L_2 be two languages produced by grammars of a certain type. Let L be the language which is the concatenation of L_1 and L_2. We want to tell for various types of grammars that produce L_1 and L_2 what type is the concatenation L. Choose the triple ($type_1$, $type_2$, $type_3$) so that when the grammar that produces the language L_1 is of $type_1$ and the grammar that produces the language L_2 is of $type_2$, then the grammar that produces the concatenation language L may not be of $type_3$.

Note: A *linear grammar* is a context-free grammar in which no production body has more than one occurrence of one variable. For example, $A \to 0B1$ or $A \to 001$ could be productions of a linear grammar, but $A \to BB$ or $A \to A0B$ could not. A *linear language* is a language that has at least one linear grammar.

a) (regular,regular,context-free)

b) (linear,regular,regular)

c) (regular,regular,linear)

d) (linear,regular,context-free)

Answer submitted: **c)**

Your answer is incorrect.

The concatenation of two regular languages is always a regular language, hence a linear language as well. Section 4.2.1 (p. 133) discusses closure of regular languages under concatenation. To show that every regular language is a linear language, start with a DFA for the regular language. Design a grammar whose variables are the states of the DFA. The start state becomes the start symbol, and whenever there is a transition from state A to state B on input *c*, add the production A → cB. Also, if A is an accepting state, add the production A → ε. The result is a linear grammar.

Question Explanation:

Explanations for the incorrect choices:

(regular,regular,regular): The concatenation of two regular languages is always a regular language because concatenation is one of the three operations that define regular expressions.

(linear,regular,linear): Suppose the first part of the concatenation is a word from the regular language. We can generate the concatenation by using only rules that are linear: Consider the disjoint union of the productions of the two grammars. The initial symbol of the concatenation is the initial symbol of the regular grammar. Use the rules of the regular grammar and replace all terminal productions (rhs does not contain nonterminal symbol) by concatenating in the end the initial symbol of the linear grammar. Now add also the rules of the linear grammar.

(context-free,context-free,context-free): The concatenation of two context-free languages is a context-free language. The reason is that the following grammar generates it: The disjoint union of the two grammars and a new rule which produces the initial symbol of the concatenation language by concatenating the two initial symbols of the two grammars.

(regular,context-free,context-free), (linear,linear,context-free), (regular,regular,context-free), (linear,regular,context-free): Same reason as above since regular and linear are also context-free languages.

(regular,regular,linear): Same reason as in the case of (regular,regular,regular) since linear are also regular languages.

Explanations for the correct choices:

(linear,linear,linear): The language $\{a^ib^ic^jd^j, i,j=1,2,..\}$ is the concatenation of two linear languages. There is a variant of the pumping lemma (Section 7.2, p.279) that applies only to linear languages. For every linear language there is a constant *n* such that if *z* is in the language and of length less than *n*, then we can write z = uvwxy such that |uy| ≤ n, |w| ≤ n, and for all *i*, uv^iwx^iy is in the language. Note that unlike the general pumping lemma for CFL's, here it is *v* and/or *x* that are allowed to be long, and the other components must be short. Can you prove this lemma, making use of the fact that for a linear grammar, the path leading to *w* in Fig. 7.6 (p. 282) must have all the variables in the entire parse tree?

(linear,regular,regular), (regular, linear,regular): A pumping lemma argument shows that the linear language $\{a^ib^i, i=1,2,...\}$ is not regular.

The correct choice is: **b)**

3. A *linear grammar* is a context-free grammar in which no production body has more than one occurrence of one variable. For example, A → 0B1 or A → 001 could be productions of a linear grammar, but A → BB or A → A0B could not. A *linear language* is a language that has at least one linear grammar.

The following statement is false: "The concatenation of two linear languages is a linear language." To prove it we use a counterexample: We give two linear languages L_1 and L_2 and show that their concatenation is not a linear language. Which of the following can serve as a counterexample?

a) L_1 ={w|w=a^nb^n, where n is a positive integer}
L_2={w|w=$(aa)^n(bb)^n$, where n is a positive integer}

b) L_1 ={w|w=$(ab)^na^nb^n$, where n is a positive integer}
L_2={w|w=$c(aaa)^n(aa)^n(ab)^n$, where n is a positive integer}

c) L_1 ={w|w=$(aaa)^n(ab)^n$, where n is a positive integer}
L_2={w|w=$(ab)^n$, where n is a positive integer}

d) L_1 ={w|w=$(ab)^na^nb^n$, where n is a positive integer}
L_2={w|w=$c(aaa)^n(ab)^n$, where n is a positive integer}

Answer submitted: **a)**

You have answered the question correctly.

Question Explanation:

The correct answer should have all properties:

1. L_1 and L_2 are both linear languages.
2. Their concatenation is provably not a linear language.

To prove the first property we need a linear grammar that generates the language.

To prove the second property, there is a variant of the pumping lemma (Section 7.2, p.279) that applies only to linear languages. For every linear language there is a constant *n* such that if *z* is in the language and of length at least *n*, then we can write z = uvwxy such that $|uy| \leq n$, $|w| \leq n$, and for all *i*, uv^iwx^iy is in the language. Note that unlike the general pumping lemma for CFL's, here it is *v* and/or *x* that are allowed to be long, and the other components must be short. Can you prove this lemma, making use of the fact that for a linear grammar, the path leading to *w* in Fig. 7.6 (p. 282) must have all the variables in the entire parse tree?

However, in order to show that a choice is incorrect, we need to prove either that one of the languages is not linear or that their concatenation is linear.

4. Let *h* be the homomorphism defined by h(a) = 01, h(b) = 10, h(c) = 0, and h(d) = 1. If we take any string *w* in (0+1)*, $h^{-1}(w)$ contains some number of strings, N(w). For example, $h^{-1}(1100)$ = {ddcc, dbc}, i.e., N(1100) = 2. We can calculate the number of strings in $h^{-1}(w)$ by a recursion on the length of *w*. For example, if w = 00x for some string x, then N(w) = N(0x), since the first 0 in w can only be produced from *c*, not from *a*.

Complete the reasoning necessary to compute N(w) for any string *w* in (0+1)*. Then, choose the correct value of N(110010).

a) 6

b) 4

c) 32

d) 13

Answer submitted: **a)**

You have answered the question correctly.

Question Explanation:

First, notice that N(ε) = 1; i.e., h^{-1}(ε) = {ε}. Also, N(0) = N(1) = 1, since h^{-1}(0) = {c} and h^{-1}(1) = {d}.

For the inductive step, If w = 00x or w = 11x, then the first symbol of *w* comes from *c* or *d*, respectively, while 0x or 1x can come from N(0x) or N(1x) different strings. Thus, N(00x) = N(0x) and N(11x) = N(1x).

Now, consider the case when w = 01x. Then *w* comes from either a string ay where h(y) = x, or *w* comes from a string cy, where h(y) = 1x. Thus, N(01x) = N(1x) + N(x). If w = 10x, the reasoning is analogous; N(10x) = N(0x) + N(x).

We now have a way to count h^{-1}(w) for any *w*. For example, if w = 00110, we reason:

- N(ε) = 1 (basis).
- N(0) = 1 (basis).
- N(10) = N(0) + N(ε) = 2 (induction, with x = ε).
- N(110) = N(10) = 2.
- N(0110) = N(110) + N(10) = 4.
- N(00110) = N(0110) = 4.