CSE 461: Programming Languages Concepts Taylan Unal

Prof. G. Tan, Spring 2020

Homework 1: **Due on Jan 25th at 6pm in Gradescope.** Total: 22 points.

1. (5 points) We have the following grammar with the start symbol `<e>`:

```
<e>  ->  <d> | <e> + <e> | <e> - <e>
<d>  ->  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

(a) Show a leftmost derivation for the expression "7 + 4 - 5"; show every step.

(b) Show a rightmost derivation for the above expression; show every step.

(c) Show two different parse trees for the above expression.

(d) The grammar is ambiguous. Show a new grammar that removes the ambiguity and makes "+" and "-" left-associative. Show the parse tree for "7 + 4 - 5" in your new grammar. Argue why this is the only parse tree in the new grammar.

(e) Show a new grammar that removes the ambiguity and makes "+" and "-" right-associative. Show the parse tree for "7 + 4 - 5" in the new grammar.

a) Leftmost Derivation of "7+4-5"

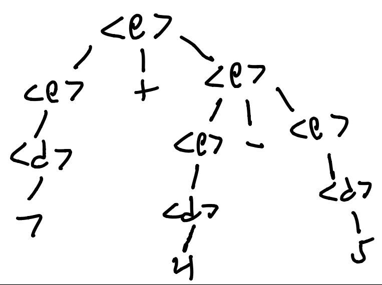
```
<e> -> <e> - <e>
    -> <e>+<e> - <e>
    -> <d>+<d> - <d>
    -> 7 + 4 - 5
```

b) Rightmost Derivation of "7+4-5"

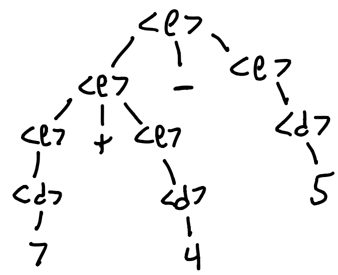
```
<e> -> <e>+<e>
    -> <e>+ <e>-<e>
    -> <d>+<d> - <d>
    -> 7 + 4 - 5
```

c) Parse trees for "7+4-5"

1. Leftmost Parse Tree

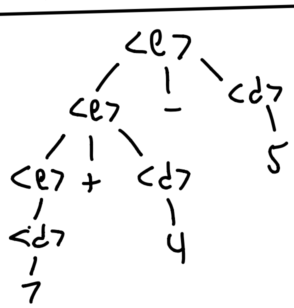


2. Rightmost Parse Tree



d) Write New Grammar removing ambiguity, making "+", "-" left associative

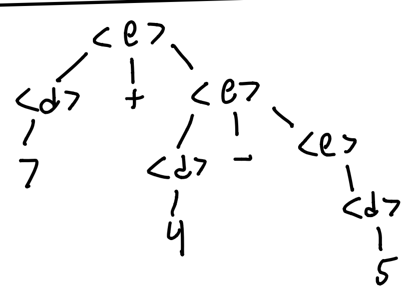
```
<e> -> <d> | <e>+<d> | <e>-<d>
<d> -> 0|1|2|3|4|5|6|7|8|9
```



This is the only tree because by using <e>+<d> and <e>-<d>, the tree grows only on the left side of the tree, making the tree's operations left associative.

e) Write New Grammar removing ambiguity, making "+", "-" right associative

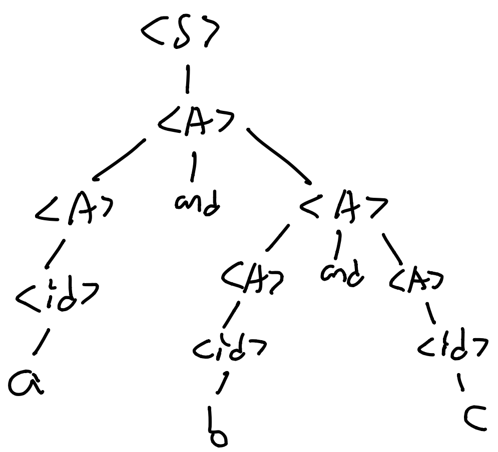
```
<e> -> <d> | <d>+<e> | <d>-<e>
<d> -> 0|1|2|3|4|5|6|7|8|9
```



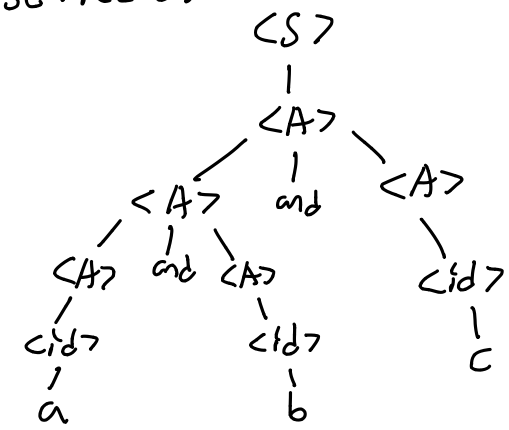
2. (3 points) Show the following BNF grammar (with start symbol <S>) is ambiguous by giving an example input and drawing its two different parse trees. Give an equivalent unambiguous grammar.

```
<S>  ->  <A>
<A>  -> <A> and <A> | <id>
<id> -> a | b | c
```

Parse Tree 1:



Parse Tree 2:



Therefore, since there are 2 parse trees for the same expression, it can be concluded that the grammar is ambiguous.

Equivalent Unambiguous Grammar:

<S> → <A>
<A> → <A> and <B> | <B>
<B> → <id>
<id> → a | b | c

3. (2 points) Consider the language consisting of strings that have n copies of the letter a followed by 2n copies of the letter b where n > 0. For example, the strings abb, aabbbb, and aaabbbbbb are in the language but a, ab, ba and aabbb are not. Give an unambiguous BNF grammar for the language.

Unambiguous BNF grammar:

<S> → <A><S><B> | <A><B>

<A> → a

<B> → bb

So grammar builds from inside out.

<A> <S> <B>

<A> <A> <S> <B> <B>

<A> <A> <A> <S> <B> <B> <B>

End Expand: <A> <A> <A> <A> <B> <B> <B> <B>

Rewrite: a a a a bb bb bb bb

4. (4 points) Consider the grammar given bellow:

```
<assign> -> <id> = <expr>
<id>     -> x | y | z
<expr>   -> <expr> + <term> | <term>
<term>   -> <term> * <factor> | <factor>
<factor> -> (<expr>) | <id>
```

add <power> -> <term>**<power> | <term>

left assoc

+	−
*	/

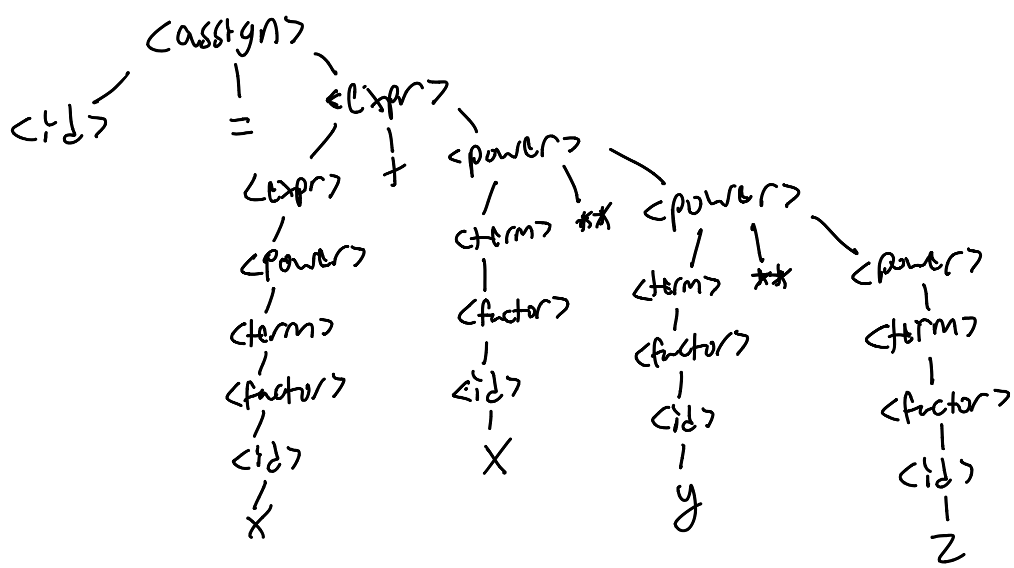
↑ higher precedence

Give a complete grammar that extends the above grammar to include a binary exponentiation operator ** (i.e., b ** n is used in some languages to mean b raised to the n-th power). In this grammar, make the ** operator right-associative and give it a higher precedence over +, but a lower precedence over *. For example, "x + x ** y ** z" should be parsed the same as "x + (x ** (y ** z))".

<assign> -> <id> = <expr>
<id> -> x | y | z
<expr> -> <expr> + <power> | <power>
<power> -> <term> ** <power> | <term>
<term> -> <term> * <factor> | <factor>
<factor> -> (<expr>) | <id>

The <power> rule grows out the tree along the right hand side, and recursively moves from lowest level of tree first, making this grammar right associative.

$\boxed{\text{Parse Tree for: "x + x ** y ** z"}}$



5. (4 points) A simplified email address has (i) an account name starting with a letter and continuing with any number of letters or digits; (ii) an @ character; (iii) a host with two or more sequences of letters or digits separated by periods; the last sequence must be a toplevel domain—either 'edu', 'org', or 'com'. Define a context-free grammar to model this language.

Use either BNF or E-BNF

BNF Grammar for Email:

<email> → <username> @ <subdomain> . <domain>

forces letter as start → <username> → <letter> | <username><letter> | <username><digit>

<subdomain> → <lettersDigits> | <lettersDigits> . <subdomain>

<lettersDigits> → <letter> | <digit> | <letter> <lettersDigits> | <digit> <lettersDigits>

<domain> → edu | org | com

<letter> → a | b | c | ... | z | A | B | C | ... | Z

<digit> → 0 | 1 | 2 | ... | 9

can create new nonterminals.

Repetition in {}

Optional Parts in [ ]

6. (4 points) The following E-BNF is the grammar for a simplified version of LISP. Convert it to a BNF grammar. Note in the following "{", "}", "[", "]", and "|" are meta-symbols of E-BNF, while "(", ")", and "." are terminals.

```
<s-exp> -> <atomic-sym> | ( <s-exp> . <s-exp> ) | ( <s-exp-list> )
<s-exp-list> -> { <s-exp> }
<atomic-sym> -> <letter> { <letter> | <number> }
<letter> -> a | b | ... | z
<number> -> 0 | 1 | ... | 9
```

```
<s-exp> -> <atomic-sym> | (<s-exp>.<s-exp>) | (<s-exp-list>)
<s-exp-list> -> <s-exp-list><s-exp> | <s-exp>
<atomic-sym> -> <atomic-sym><letter><number> | <atomic-sym><letter><number>
              | <atomic-sym><letter> | <letter>
<letter> -> a|b|c|...|z
<number> -> 0|1|2|...|9
```