Lab 5
Due: at the end of lab

1 Exercises

Write out solutions to the exercises below, following the formatting in the lecture slides.

1. In class, we saw a grammar that used only `<exp>` productions to build expressions with + and ∗, but allowed us to draw multiple valid parse trees for the same expression. We can rewrite the grammar with additional non-terminal symbols so that ∗ operations are grouped before + when building a parse tree from the bottom up. This gives the operator precedence that we expect from arithmetic.

   Given the grammar:

   ```
   <exp> ::= <exp> + <mulexp> | <mulexp>
   <mulexp> ::= <mulexp> * <rootexp> | <rootexp>
   <rootexp> ::= ( <exp> ) | a | b | c | d | e
   ```

   Draw parse trees for the two expressions below. Make sure the parentheses are included. Use `<exp>` as the start symbol in both trees.

   `(c+d)`

   `(a+b)*c+d`

2. The following two grammars approximately describe valid `bool lists` in F#.

   Grammar A

   ```
   <list> ::= [ ] | [ <sublist> ; <sublist> ]
   <sublist> ::= <sublist> ; <value> | <value>
   <value> ::= true | false
   ```

   Grammar B

   ```
   <list> ::= [ <sublist> ] | [ ]
   <sublist> ::= <sublist> ; <sublist> | <value>
   <value> ::= true | false
   ```

   Draw parse trees for the expression `[true;false]` using each grammar.

   Grammar A | Grammar B

3. Two grammars are equivalent if every expression that is valid in one grammar is also valid in the other grammar. The shape of the resulting parse trees is unimportant.

   Are grammars **A** and **B** equivalent? If you think they are equivalent, write an argument that every case of lists is valid in both grammars. If you think they are not equivalent, give a counterexample sentence that is valid in one grammar but not the other.