Solutions for Sample Questions for Midterm 2 (CS 421 Fall 2014)

On the actual midterm, you will have plenty of space to put your answers.
Some of these questions may be reused for the exam.

1. Given a polymorphic type derivation for
   {} |- let pair = fun x -> (x, x) in pair(pair 3) : ((int * int) * (int * int))

Solution:

Let Γ_1 = {x : ‘a}. Γ_2 = {pair : ∀’a. ‘a -> ‘a * ‘a}.
The infixed data construct , (comma) has type ∀’a ‘b. ‘a -> ‘b -> ‘a * ‘b

Let LeftTree =

$$
\text{Fun}\ \dfrac{\text{App}\ \dfrac{\text{App}\ \dfrac{\text{Const}\ \dfrac{\text{Instance: ‘a} \to \text{‘a, ‘b} \to \text{‘a}}{\Gamma_1 \vdash (,)\ :\ \text{‘a -> ’a -> ‘a * ‘a}} \qquad \text{Var}\ \dfrac{}{\Gamma_1 \vdash x : \text{‘a}}}{\Gamma_1 \vdash (,)\ x : \text{‘a -> ‘a * ‘a}} \qquad \text{Var}\ \dfrac{}{\Gamma_1 \vdash x : \text{‘a}}}{\{x : \text{‘a}\}. \vdash (x, x) : \text{‘a * ‘a}}}{\{\} \vdash \text{fun } x \text{ -> } (x,x) : \text{‘a -> ‘a * ‘a}}
$$

Let RightTree =

$$
\text{App}\ \dfrac{\text{Var}\ \dfrac{\text{Instance: ‘a} \to \text{int * int}}{\Gamma_2 \vdash \text{pair : int * int -> ((int * int) * (int * int))}} \qquad \text{App}\ \dfrac{\text{Var}\ \dfrac{\text{Instance: ‘a} \to \text{int}}{\Gamma_2 \vdash \text{pair : int -> int * int}} \qquad \text{Const}\ \dfrac{}{\Gamma_2 \vdash 3 : \text{int}}}{\Gamma_2 \vdash \text{pair(3) : int * int}}}{\{\text{pair : ∀’a. ‘a -> ‘a * ‘a}\} \vdash \text{pair(pair 3) : ((int * int) * (int * int))}}
$$

Then the full proof is

$$
\dfrac{\text{LeftTree} \qquad\qquad \text{RightTree}}{\{\} \vdash \text{let pair = fun x -> (x, x) in pair(pair 3) : ((int * int) * (int * int))}}
$$

2. Give a (most general) unifier for the following unification instance. Capital letters denote variables of unification. Show your work by listing the operation performed in each step of the unification and the result of that step.

   {X = f(g(x),W); h(y) = Y; f(Z,x) = f(Y,W)}

Solution:

Unify {X = f(g(x),W); h(y) = Y; f(Z,x) = f(Y,W)}
= Unify {h(y) = Y; f(Z,x) = f(Y,W)} o {X → f(g(x),W)} by eliminate (X = f(g(x),W))

= Unify {Y = h(y); f(Z,x) = f(Y,W)} o {X → f(g(x),W)}   by orient (h(y) = Y)
= Unify {f(Z,x) = f(h(y),W)} o {X → f(g(x),W), Y → h(y)} by eliminate (Y = h(y))
= Unify {Z = h(y); x=W} o {X → f(g(x),W), Y → h(y)}   by decompose (f(Z,x) = f(h(y),W))
= Unify {x = W} o {X → f(g(x),W), Y → h(y), Z → h(y)}   by eliminate (Z = h(y))
= Unify {W = x} o {X → f(g(x),W), Y → h(y), Z → h(y)}   by orient (x = W)
= Unify{} o {X → f(g(x),x), Y → h(y), Z → h(y), W → x} by eliminate (W = x)
Answer: {X → f(g(x),x), Y → h(y), Z → h(y), W → x}

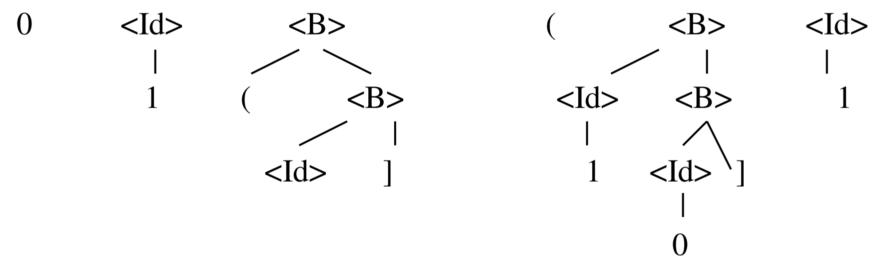
3. For each of the following descriptions, give a regular expression over the alphabet {a,b,c}, and a regular grammar that generates the language described.
   a. The set of all strings over {**a**, **b**, **c**}, where each string has at most one **a**
      Solution: (b ∨ c)*(a ∨ ε) (b ∨ c)*
      <S> ::= b<S> | c<S> | a<NA> | ε
      <NA> ::= b<NA> | c<NA> | ε

   b. The set of all strings over {**a**, **b**, **c**}, where, in each string, every **b** is immediately followed by at least one **c**.
      Solution: (a ∨ c)*(bc(a ∨ c)*)*
      <S> ::= a<S> | c<S> | b<C> | ε
      <C> ::= c<S>

   c. The set of all strings over {**a**, **b**, **c**}, where every string has length a multiple of four.
      Solution: ((a ∨ b ∨ c) (a ∨ b ∨ c) (a ∨ b ∨ c) (a ∨ b ∨ c))*
      <S> ::= a<TH> | b<TH> | c<TH> | ε
      <TH> ::= a<TW> | b<TW> | c<TW>
      <TW> ::= a<O> | b<O> | c<O>
      <O> ::= a<S> | b<S> | c<S>

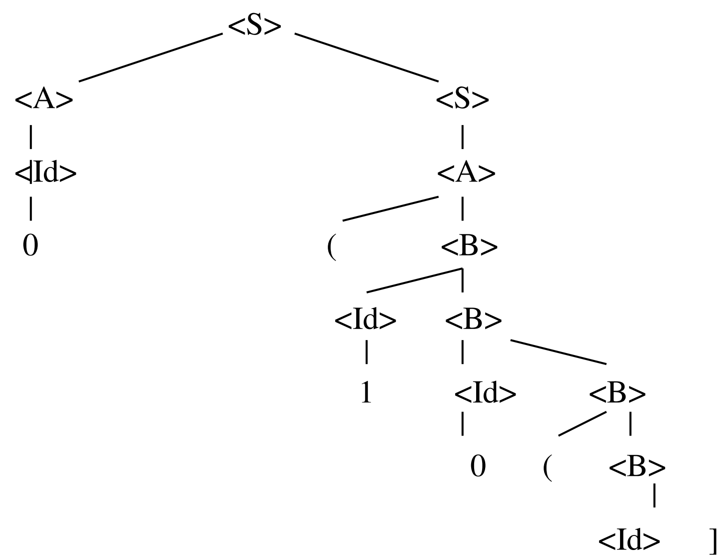
4. Consider the following grammar:
   <S> ::= <A> | <A> <S>
   <A> ::= <Id> | (<B>
   <B> ::= <Id>] | <Id><B> | (<B>
   <Id> ::= 0 | 1
   For each of the following strings, give a parse tree for the following expression as an <S>, if one exists, or write "No parse" otherwise:

   a. (0 1 (1] ((1 0] 1
   Solution:



b. 0 (1 0 (1]

Solution:



c. (0 (1 0 1] 0]

Solution: No parse tree

5. Demonstrate that the following grammar is ambiguous (Capitals are non-terminals, lowercase are terminals):

$$S \rightarrow A\ a\ B \mid B\ a\ A$$
$$A \rightarrow b \mid c$$
$$B \rightarrow a \mid b$$

Solution: String: bab

6. Write an unambiguous grammar generating the set of all strings over the alphabet {0, 1, +, **-**} , where + and – are infixed operators which both associate to the left and such that + binds more tightly than -.

Solution:

<S> ::= <plus> | <S> - <plus>
<plus> :: <id> | <plus> + <id>
<id> ::= 0 | 1

7. Write a recursive descent parser for the following grammar:,
<S> ::= <N> % <S> | <N>
<N> ::= g <N> | a | b
You should include a datatype **token** of tokens input into the parser, one or more datatypes representing the parse trees produced by parsing (the abstract syntax trees), and the function(s) to produce the abstract syntax trees. Your parser should take a list of tokens as input and generate an abstract syntax tree corresponding to the parse of the input token list.
Solution:

```
type  token = ATk | BTk | GTk | PercentTk
type  s = Percent of (n * s) | N_as_s of n
and n = G of n | A | B

let rec s_parse tokens  =
   match n_parse tokens with (n, tokens_after_n) ->
      (match tokens_after_n with PercentTk::tokens_after_percent ->
           (match s_parse tokens_after_percent
            with (s, tokens_after_s) -> (Percent (n,s), tokens_after_s))
         | _ -> (N_as_s n, tokens_after_n))
and n_parse tokens =
   match tokens
   with GTk::tokens_after_g ->
      (match n_parse tokens_after_g
          with (n, tokens_after_n) -> (G n, tokens_after_n))
     | ATk::tokens_after_a -> (A, tokens_after_a)
     | BTk::tokens_after_b -> (B, tokens_after_b)

let parse tokens =
   match s_parse tokens
   with (s, []) -> s
     | _ -> raise (Failure "No parse")
```