1. Give a type inference and a constraint set for each of the following expressions. You don't need to solve the constrains, just present them. Show the proof tree with all of the steps together with the name of the rule that you are applying in each step and the constraints arising from each step. A reference sheet containing all rules is at the end of this document.

   (a) `fun x -> (x+1; fun y-> let x = 1 in y)`

   (b)
   ```
   let f = fun x -> 2 * x in let g = fun x -> x + f x
                                in if f 3 = g 2 then f 3 else g 2
   ```

   (c) `(fun x -> fun y -> x y+x (y+1)) (fun x -> 2*x) (5+1)`

   (d) `let rec f = fun zws -> f(zws) + 1 in f`

2. Find the most general substitution that satisfies each set of the following constrains. Show each step of unification algorithm as you perform it (even if the unification fails, you should write down all the steps till the point that it fails). Capital letters represent constructors, others are variables.

   (a) $\Big\{ \Big(F\big(x, G(y)\big), F\big(G(z), G(G(w))\big)\Big), \Big(H(H(y)), H(z)\Big) \Big\}$

   (b) $\Big\{ \Big(S(S(z)), S(w)\Big), \Big(F(x, G(x)), F(y, z)\Big), \Big(G(y, x), G(S(v), y)\Big) \Big\}$

   (c) $\Big\{ \Big(A(B(x), y), A(z, x)\Big), \Big(C(w, B(x)), C(D(z, z), B(y))\Big), \Big(F\big(x, y\big), F\big(A(w, B(w)), x\big)\Big) \Big\}$

3. For each of the following sets of strings, first design a DFA (or NFA) and then write a regular expression describing that set. Note that your regular expression cannot use the extensions provided by `ocamllex`, that is you can use only `*`, | and concatenation (parentheses for grouping subexpressions are allowed too).

   (a) All strings w over alphabet $\{a, b\}$ where the number of a's or b's in w is even.

(b) All strings w over alphabet $\{0, 1, 2\}$ where w contains the substring 2210.

(c) All strings w over alphabet $\{0, 1, 2\}$ where w does not contain the substring 2210.

(d) All strings w over alphabet $\{a, b\}$ where the number of substrings ab in w is the same as the number of substrings ba.

4. Consider the following grammar. As usual the the capital letters (in italics) are non-terminals. S is the start symbol.

$$S \Rightarrow \texttt{(}L\texttt{)} \mid \texttt{a}$$
$$L \Rightarrow L\texttt{,}S \mid S$$

What is the set of terminals in this grammar? Show a parse tree, a leftmost derivation and a rightmost derivation for each of the strings `(a,a)`, `(a,(a,a))` and `(a,((a,a),(a,a)))`. Compute all the First and Follow sets. Build the SLR parsing tables and parse all these given strings using those tables. Introduce an ocaml type to represent the token set of this grammar. Introduce necessary types to represent the parse trees of this grammar. Use those types to actually represent parse trees for all the strings given.

5. Consider the following grammars.

G_1 : $S \Rightarrow \texttt{a}S\texttt{b}S \mid \texttt{b}S\texttt{a}S \mid \epsilon$

G_2 :
```
<bexp> ::= <bexp> or <bterm> | <bterm>
<bterm> ::= <bterm> and <bfactor> | <bfactor>
<bfactor> ::= not <bfactor> | ( <bexp> ) | true | false
```

G_3 : $R \Rightarrow R\texttt{+}R \mid RR \mid R\texttt{*} \mid \texttt{(}R\texttt{)} \mid \texttt{a} \mid \texttt{b}$

Start symbols in G_1, G_2 and G_3 are S, `<bexp>` and R respectively. Determine whether these grammars are ambiguous or not. Present two parse trees that produce the same string for ambiguous ones.

6. For each of the following grammars, first determine whether they are LL(1) or not using the First sets disjointness test. If they are not LL(1) modify them so that they become LL(1) (for example use the modification we did on page 38 in `14-rec-dec-parsing.pdf`). Then introduce a type to represent the tokens in each grammar and then a type to encode all the parse trees of that grammar. Then write a recursive descent parser for each of them as we did in class.

   (a) S is the start symbol.

$$\begin{aligned} S &\Rightarrow \mathtt{i}E\mathtt{t}SG \mid \mathtt{a} \\ G &\Rightarrow \mathtt{e}S \mid \epsilon \\ E &\Rightarrow \mathtt{b} \end{aligned}$$

   (b) S is the start symbol.

$$\begin{aligned} S &\Rightarrow \mathtt{aa}A\mathtt{d}S \mid \mathtt{a} \\ A &\Rightarrow \mathtt{b}A \mid \mathtt{a} \end{aligned}$$

   (c) E is the start symbol.

$$\begin{aligned} E &\Rightarrow TH \\ H &\Rightarrow \mathtt{+}TH \mid \epsilon \\ T &\Rightarrow FY \\ Y &\Rightarrow \mathtt{*}FY \mid \epsilon \\ F &\Rightarrow \mathtt{(}E\mathtt{)} \mid \mathtt{id} \end{aligned}$$

7. Prove the following judgements once using natural semantics and once using transition semantics. A list of rules of natural semantics is at the end of this document, for transitional semantics look up the lecture notes.

   (a) $\big($`skip; x:=x+y ; y:=x-y; x:=x-y; if x > 10 then x:=y+2*x else y:= 2*y+x` , $\{$`x`$\to$ `1, y`$\to$ `20`$\}\big) \Downarrow \{$`x`$\to$ `41, y`$\to$ `1`$\}$

   (b) $\big($`let y=5 in let x = y+5 in x:=x+y,` $\{$`x`$\to$ `1, y`$\to$ `20`$\}\big) \Downarrow \{$`x`$\to$ `1, y`$\to 20\}$

   (c) $\big($`let y=5 in let x = y+5 in z:=x+y,` $\{$`x`$\to$ `1, y`$\to$ `20, z`$\to 3\}\big) \Downarrow \{$`x`$\to$ `1, y`$\to 20$`, z`$\to$ `15`$\}$

Type Rules:

(Unit) $$\frac{}{\Gamma \vdash () \ : \ \mathsf{unit}}$$

(Nil) $$\frac{}{\Gamma \vdash [\,] \ : \ \tau \ \mathsf{list}}$$

(Bool) $$\frac{}{\Gamma \vdash b \ : \ \mathsf{bool}} \qquad \text{where } b = \mathsf{true} \text{ or } \mathsf{false}$$

(Int) $$\frac{}{\Gamma \vdash n \ : \ \mathsf{int}} \qquad \text{where } n \text{ is an integer}$$

(Float) $$\frac{}{\Gamma \vdash n \ : \ \mathsf{float}} \qquad \text{where } n \text{ is a floating point number}$$

(Var) $$\frac{}{\Gamma \vdash x \ : \ \tau} \qquad \text{where } \Gamma(x) = \tau$$

(Pair) $$\frac{\Gamma \vdash e_1 \ : \ \tau_1 \qquad \Gamma \vdash e_2 \ : \ \tau_2}{\Gamma \vdash (e_1,\, e_2) \ : \ (\tau_1, \tau_2)}$$

(List) $$\frac{\Gamma \vdash e_h \ : \ \tau \qquad \Gamma \vdash e_t \ : \ \tau \ \mathsf{list}}{\Gamma \vdash e_h{::}e_t \ : \ \tau \ \mathsf{list}}$$

(Fun) $$\frac{[x \ : \ \tau_x] + \Gamma \vdash e \ : \ \tau_e}{\Gamma \vdash \mathsf{fun}\ x \ \text{->}\ e \ : \ \tau_x \to \tau_e}$$

(IF) $$\frac{\Gamma \vdash e_1 \ : \ \mathsf{bool} \qquad \Gamma \vdash e_2 \ : \ \tau \qquad \Gamma \vdash e_3 \ : \ \tau}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \ : \ \tau}$$

(App) $$\frac{\Gamma \vdash e_f \ : \ \tau_a \to \tau_r \qquad \Gamma \vdash e_a \ : \ \tau_a}{\Gamma \vdash e_f\ e_a \ : \ \tau_r}$$

(LetIn) $$\frac{\Gamma \vdash e_x \ : \ \tau_x \qquad [x \ : \ \tau_x] + \Gamma \vdash e \ : \ \tau}{\Gamma \vdash \mathsf{let}\ x = e_x\ \mathsf{in}\ e \ : \ \tau}$$

(LetRecIn) $$\frac{[x \ : \ \tau_x] + \Gamma \vdash e_x \ : \ \tau_x \qquad [x \ : \ \tau_x] + \Gamma \vdash e \ : \ \tau}{\Gamma \vdash \mathsf{let\ rec}\ x = e_x\ \mathsf{in}\ e \ : \ \tau}$$

(Arithmetic Operators) $$\frac{\Gamma \vdash e_2 : \mathtt{int} \qquad \Gamma \vdash e_1 : \mathtt{int}}{\Gamma \vdash e_1 \oplus e_2 : \mathtt{int}} \ (\oplus \in \{+, -, *, /, \cdots\})$$

(Boolean Operators) $$\frac{\Gamma \vdash e_2 : \mathtt{bool} \qquad \Gamma \vdash e_1 : \mathtt{bool}}{\Gamma \vdash e_1 \oplus e_2 : \mathtt{bool}} \ (\oplus \in \{\&\&, ||\})$$

(NOT Operator) $$\frac{\Gamma \vdash e_1 : \mathtt{bool}}{\Gamma \vdash \mathtt{not}\ e_1 : \mathtt{bool}}$$

(Relational Operators) $$\frac{\Gamma \vdash e_2 : \mathtt{int} \qquad \Gamma \vdash e_1 : \mathtt{int}}{\Gamma \vdash e_1 \oplus e_2 : \mathtt{bool}} \ (\oplus \in \{<, >, \leq, \geq, =, \cdots\})$$

(Sequenece) $$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1;\ e_2 : \tau_2}$$

Natural Semantic Rules:

(Identifier)	$(I, m) \Downarrow m(I)$	
(Const)	$(n, m) \Downarrow n$	where n is a constant integer
(Bool Const)	$(\mathbf{true}, m) \Downarrow \mathbf{true}$	$(\mathbf{false}, m) \Downarrow \mathbf{false}$
(Bool Op)	$\dfrac{(B, m) \Downarrow \mathbf{false}}{(B \;\mathbf{\&\&}\; B', m) \Downarrow \mathbf{fasle}}$	$\dfrac{(B, m) \Downarrow \mathbf{true}}{(B \;\vert\vert\; B', m) \Downarrow \mathbf{true}}$
	$\dfrac{(B, m) \Downarrow \mathbf{true} \quad (B', m) \Downarrow b}{(B \;\mathbf{\&\&}\; B', m) \Downarrow b}$	$\dfrac{(B, m) \Downarrow \mathbf{false} \quad (B', m) \Downarrow b}{(B \;\vert\vert\; B', m) \Downarrow b}$
	$\dfrac{(B, m) \Downarrow \mathbf{true}}{(\mathbf{not}\; B, m) \Downarrow \mathbf{false}}$	$\dfrac{(B, m) \Downarrow \mathbf{false}}{(\mathbf{not}\; B, m) \Downarrow \mathbf{true}}$
(Relational Op)	$\dfrac{(E, m) \Downarrow u \quad (E', m) \Downarrow v \quad u \oplus v = b}{(E \oplus E', m) \Downarrow b}$	where $\oplus \in \{=, \leq, \geq, \dots\}$
(Arithmetic Op)	$\dfrac{(E, m) \Downarrow u \quad (E', m) \Downarrow v \quad u \oplus v = b}{(E \oplus E', m) \Downarrow b}$	where $\oplus \in \{+, -, *, \dots\}$
(Skip)	$(\mathbf{skip}, m) \Downarrow m$	
(Assignment)	$\dfrac{(E, m) \Downarrow v}{(I := E, m) \Downarrow m[I \leftarrow v]}$	where I is an indentifier name
(Sequencing)	$\dfrac{(C, m) \Downarrow m' \quad (C', m') \Downarrow m''}{(C; C', m) \Downarrow m''}$	
(If)	$\dfrac{(B, m) \Downarrow \mathbf{true} \quad (C, m) \Downarrow m'}{(\mathbf{if}\; B\; \mathbf{then}\; C\; \mathbf{else}\; C', m) \Downarrow m'}$	$\dfrac{(B, m) \Downarrow \mathbf{false} \quad (C', m) \Downarrow m'}{(\mathbf{if}\; B\; \mathbf{then}\; C\; \mathbf{else}\; C', m) \Downarrow m'}$
(While)	$\dfrac{(B, m) \Downarrow \mathbf{false}}{(\mathbf{while}\; B\; \mathbf{do}\; C, m) \Downarrow m}$	
	$\dfrac{(B, m) \Downarrow \mathbf{true} \quad (C, m) \Downarrow m' \quad (\mathbf{while}\; B\; \mathbf{do}\; C, m') \Downarrow m''}{(\mathbf{while}\; B\; \mathbf{do}\; C, m) \Downarrow m''}$	
(LetIn)	$\dfrac{(E, m) \Downarrow v \quad (C, m[I \leftarrow v]) \Downarrow m'}{(\mathbf{let}\; I = E\; \mathbf{in}\; C, m) \Downarrow m''}$	where for all $x \neq I$, we have $m''(x) = m'(x)$. If $m(I)$ is defined, then $m''(I) = m(I)$, otherwise $m''(I)$ is undefined.