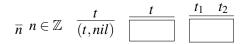


## Problem 1 [Rules of inferences] (20 pts)

a) (4 pts) Complete the following inference rules for binary trees. Examples of binary trees include 1, (1,*nil*), (1,2), (*nil*,(1,2)), ((1,2),(3,4)), etc. The inference rules are:



b) (8 pts) Consider the following grammar for a set *S* of Boolean formulas.

$$f \to T \mid F \mid f \land f \mid f \lor f$$

Complete the following derivation tree for proving that  $(T \land F) \lor (F \lor T)$  is in *S*:

$$\overline{(T \wedge F) \vee (F \vee T)}$$

c) (8 pts) The semantics of formulas is defined as follows:

$$\begin{bmatrix} T \end{bmatrix} = true \\ \begin{bmatrix} F \end{bmatrix} = false \\ \begin{bmatrix} f_1 \land f_2 \end{bmatrix} = \begin{bmatrix} f_1 \end{bmatrix} \text{ and also } \begin{bmatrix} f_2 \end{bmatrix} \\ \begin{bmatrix} f_1 \lor f_2 \end{bmatrix} = \begin{bmatrix} f_1 \end{bmatrix} \text{ orelse } \begin{bmatrix} f_2 \end{bmatrix}$$

where and also returns *true* only if both arguments are *true* (otherwise, *false*), and orelse returns *false* only if both arguments are *false* (otherwise, *true*).

Let n(f) denote the number of occurrences of F in formula f. For example,  $n(T \land (F \lor F)) = 2$ . Complete the proof of the following property over every f:

$$n(f) = 0 \implies [[f]] = true \tag{1}$$

- (Base case 1) The first base case is when f = T. Because n(f) = 0 and [[f]] = true, the property holds.
- (Base case 2) The other base case is when f = F. Because n(f) = 1, the premise of (1) is false. Therefore, the entire property (1) is true.
- (Inductive case 1) The first inductive case is when  $f = f_1 \wedge f_2$ . The inductive hypotheses (I.Hs) are

$$n(f_1) = 0 \implies [[f_1]] = true \tag{2}$$

$$n(f_2) = 0 \implies [[f_2]] = true \tag{3}$$

If n(f) = 0, then  $n(f_1) = \bigsqcup$  and  $n(f_2) = \bigsqcup$  because there is no occurrence of *F* in *f*. By I.Hs (2) and (3),  $\llbracket f_1 \rrbracket = \bigsqcup$  and  $\llbracket f_2 \rrbracket = \bigsqcup$ . By the definition of andalso,  $\llbracket f \rrbracket = true$ , which completes the proof for the case.

• (Inductive case 2) The other inductive case is when  $f = f_1 \lor f_2$ . If n(f) = 0, then  $n(f_1) =$ and  $n(f_2) =$  because there is no occurrence of *F* in *f*. By I.Hs (2) and (3),  $[[f_1]] =$  and  $[[f_2]] =$ . By the definition of orelse, [[f]] = true, which completes the proof for the case.

Problem 2 [Functional Programming] (20 pts)

a) (10 pts) Consider the following similar two functions written in OCaml.

```
let rec double_all l =
match l with
| [] -> []
| hd::tl -> (hd+hd) :: (double_all tl)
let rec dec_all l =
match l with
| [] -> []
| hd::tl -> (hd - 1) :: (dec_all tl)
```

Using the following higher-order function map,

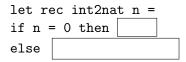
```
let rec map f l =
match l with
| [] -> []
| hd::tl -> (f hd)::(map f tl)
```

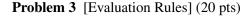
rewrite the two functions:

<pre>let double_all 1 =</pre>	
let dec_all l =	

b) (10 pts) Let us define the type of natural numbers as follows:

Complete the following definition of int2nat : int -> nat, which converts integers to natural numbers. For example, int2nat 3 evaluates to (Succ (Succ (Succ Zero))).





Consider the following language.

$$E \rightarrow n \mid x \mid E + E \mid E - E \mid E! \mid fib(E)$$

where E! denotes the factorial of E, and fib(E) denotes the *E*-th Fibonacci number.

The factorial of *n* for  $n \ge 0$  is 1 and the factorial of *n* for  $n \ge 1$  is  $n \times (n-1)!$ . The Fibonacci numbers are inductively defined as follows:

$$\operatorname{fib}(n) = \begin{cases} 1 & (n \le 1) \\ \operatorname{fib}(n-1) + \operatorname{fib}(n-2) & (n \ge 2) \end{cases}$$

Complete the following evaluation rules.

$$\overline{\rho \vdash n \Rightarrow n} \qquad \overline{\rho \vdash x \Rightarrow \rho(x)}$$

$$\frac{\rho \vdash E_1 \Rightarrow n_1 \quad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 + E_2 \Rightarrow n_1 + n_2}$$

$$\frac{\rho \vdash E_1 \Rightarrow n_1 \quad \rho \vdash E_2 \Rightarrow n_2}{\rho \vdash E_1 - E_2 \Rightarrow n_1 - n_2}$$

$$\frac{\rho \vdash E \Rightarrow n}{\rho \vdash E! \Rightarrow 1} \quad n \le 0$$

$$\frac{\rho \vdash E \Rightarrow n}{\rho \vdash E! \Rightarrow \Box} \quad n \ge 1$$

$$\frac{\rho \vdash E \Rightarrow n}{\rho \vdash \operatorname{fib}(E) \Rightarrow 1} \quad n \le 1$$

$$\rho \vdash E \Rightarrow n$$

$$p \vdash \operatorname{fib}(E) \Rightarrow \Box \quad n \le 2$$

## **Problem 4** [Nameless Representation] (10 pts)

Write the nameless representation of the following program *P*:

which is trans(P)([]). The trans function is defined as follows:

$$\begin{aligned} & \operatorname{trans}(n)(\rho) &= n \\ & \operatorname{trans}(x)(\rho) &= \#n & (n \text{ is the first} \\ & \operatorname{position of } x \text{ in } \rho) \\ & \operatorname{trans}(E_1 + E_2)(\rho) &= \operatorname{trans}(E_1)(\rho) + \operatorname{trans}(E_2)(\rho) \\ & \operatorname{trans}(\operatorname{let} x = E_1 \text{ in } E_2)(\rho) &= \operatorname{let} \operatorname{trans}(E_1)(\rho) \text{ in} \\ & \operatorname{trans}(E_2)(x :: \rho) \\ & \operatorname{trans}(\operatorname{proc}(x) E)(\rho) &= \operatorname{proc} \operatorname{trans}(E)(x :: \rho) \end{aligned}$$

Problem 5 [Scoping] (10 pts)

Recall the language with explicit references and static scoping for procedures, and consider the following two programs.

type nat = Zero | Succ of nat

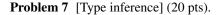
- a) Write the evaluation result of the first program.
- b) Write the evaluation result of the second program.

Problem 6 [Evaluation strategy] (10 pts).

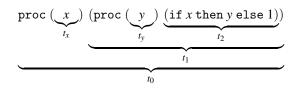
Consider lambda calculus with the normal order strategy and the following expression.

 $(\lambda x. (x x)) (\lambda x. (x x))$ 

- a) Write the result after *one step* of  $\beta$ -reduction.
- b) Will the program eventually terminate? (write yes/no)



Consider the following program.



a) Generate type equations.

b) Solve the equations using the unification algorithm, and write the final substitution.

Problem 8 [Garbage Collection] (10 pts).

Recall the language with records, pointers, and automatic garbage collection we learned in class. Consider the following environment  $\rho$  and memory  $\sigma$  before GC:

$$\rho = \begin{bmatrix} x \mapsto l_1 \\ y \mapsto l_2 \\ z \mapsto l_3 \end{bmatrix} \qquad \sigma = \begin{bmatrix} l_1 \mapsto 0 \\ l_2 \mapsto \{a \mapsto l_3, b \mapsto l_1, c \mapsto l_5\} \\ l_3 \mapsto l_5 \\ l_4 \mapsto (x, E, [z \mapsto l_5]) \\ l_5 \mapsto 0 \\ l_6 \mapsto l_7 \\ l_7 \mapsto l_8 \\ l_8 \mapsto l_6 \end{bmatrix}$$

Describe the memory  $GC(\rho, \sigma)$  that can be obtained after GC:

$$\mathsf{GC}(\rho,\sigma) =$$

Problem 9 [O/X questions] (20 pts).

Mark O for each correct statement (X for wrong statement).

- a) The type checker for C programs is sound and complete. (O,X)
- b) Manual memory management in C is difficult in general, leading to memory-leak, double-free, and use-after-free errors. (O,X)
- c) Any lambda calculus expression can be translated into a Turing machine. (O,X)
- d) A type system that always accepts input programs is sound. (O,X)

e) The type of f in the following OCaml code is (int -> bool) -> int -> int -> int.(O,X)

```
let rec sum_if_true test first second =
  (if test first then first else 0)
  + (if test second then second else 0)
```

f) The following function is tail-recursive. (O,X)

let rec f () = f ()

g) We cannot further remove syntactic sugars from Lambda calculus. (O,X)

h) Using eager evaluation, the following program terminates. (O,X)

```
letrec infinite(x) = (infinite x)
in let f = proc (x) (1)
    in (f (infinite 0))
```

- i) Determining the values of program variables is a dynamic property. (O,X)
- j) Imperative languages encourage to use statements and loops, whereas functional languages encourage to use expressions and recursion. (O,X)