Final examination — given Tuesday 8th December

General instructions

Online, take-home exam.

Deadline to submit via UNM Learn: 9:00pm.

Total points available: 200 pts.

Answer in the spaces provided.

Your name (print):

I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.

Please sign and date:

3.1 Binary trees in Haskell (30 pts)

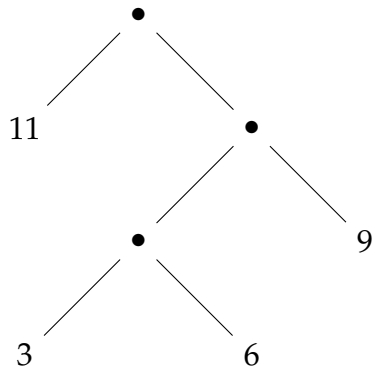
Here is the definition of a Haskell type of polymorphic binary trees with exactly two sub-trees per node:

```
data Tree a = Leaf a
            | Node (Tree a) (Tree a)
```

where the first and second arguments to `Node` represent the "left" and "right" sub-trees of that node, respectively.

1. (10 pts) Briefly explain the role of the type variable `a` in the type definition above.

2. (10 pts) Write down the Haskell value of type `Tree Int` that represents the following tree structure:



3. (10 pts) Define a Haskell function

   ```
   treeMap :: (a -> b) -> Tree a -> Tree b
   ```

   that maps the supplied function across every leaf value in the tree that is passed as the second argument.

3.2 Proving properties of Haskell list functionals (20 pts)

Assume that the functions `f :: Int -> Int` and `g :: Int -> Int` are such that

```
(f . g) x = (g . f) x
```

for all integers `x :: Int`.

Now prove, by induction on the structure of lists, that

```
map (f . g) xs = map (g . f) xs
```

for all lists `xs :: [Int]`.

You must clearly state your inductive hypothesis in the inductive case of your proof.

3.3 Programming in the untyped lambda-calculus (40 pts)

1. (20 pts) The desired behavior of the untyped lambda-term *iszero* that tests whether a Church numeral represents zero can be specified as follows:

$$\mathit{iszero}\ \mathsf{c}_0 \longrightarrow_\beta^* \mathit{tru} \qquad\qquad \mathit{iszero}\ \mathsf{c}_{i+1} \longrightarrow_\beta^* \mathit{fls}$$

where $\longrightarrow_\beta^*$ represents any number of applications of full-beta reduction (that is, the reduction strategy where any redex can be reduced in any order via the beta-rule).

Write down a definition of *iszero* that satisfies this specification **and briefly explain why your definition is correct.**

You may assume untyped lambda-terms *tru* and *fls*, representing truth and falsity, and standard definitions of Church numerals, as presented in class.

2. (20 pts) Consider the untyped-lambda term f, defined as follows:

$$f \;=\; \lambda n.\, n\ (\lambda b.\, b\ \mathit{fls}\ \mathit{tru})\ \mathit{tru}$$

What test does f implement on the Church numeral passed as its argument?

Explain your answer, by reference to the term $f\ \mathbf{c_2}$ which applies this test to the Church numeral representing the number two.

You may assume untyped lambda-terms *tru* and *fls*, representing truth and falsity, and standard definitions of Church numerals, as presented in class.

3.4 Typing and evaluation in the simply-typed lambda-calculus (30 pts)

1. (15 pts) Using the **typing rules** for the simply-typed lambda calculus with numbers and booleans (presented in Appendix A), **attempt to construct a typing derivation for the following term**, starting in the empty typing environment:

$$\left(\lambda f : \mathsf{Nat} \to \mathsf{Bool}.\, f\ 0\right)\ \left(\lambda x : \mathsf{Nat}.\, \mathsf{iszero}\ \left(\mathsf{pred}\ x\right)\right)$$

   If your derivation is successful, clearly state this and identify the resulting type.

   If your derivation fails, construct as much of the derivation as possible and explain where and why the derivation fails.

2. (15 pts) Using the **CBV evaluation rules** for the simply-typed lambda calculus with numbers and booleans (presented in Appendix A), **construct a full derivation for *each* step of reduction of the following term**:

$$\left(\lambda f : \mathsf{Nat} \to \mathsf{Bool}.\, f\ 0\right)\ \left(\lambda x : \mathsf{Nat}.\, \mathsf{iszero}\ (\mathsf{pred}\ x)\right)$$

Identify when no more reduction steps can be taken and whether the resulting term is a value or not. **If the resulting term is not a value,** explain, with justification, why no further reduction steps are possible.

You should attempt to reduce this term **regardless** of whether you found it to be well-typed or not in part 1.

3.5 General recursion in the simply-typed lambda-calculus (40 pts)

1. (15 pts) Explain why the mechanism we studied for implementing general recursion in the untyped lambda-calculus cannot be employed in the simply-typed lambda-calculus.

2. (25 pts) The evaluation rules for the simply-typed lambda-calculus with booleans and numbers are presented in Appendix A and the extension to implement general recursion is presented in Appendix B.

   With reference to those rules, **explain and illustrate the behavior of the following term:**

   $$\left(\mathsf{fix}\ \left(\lambda f : \mathsf{Nat} \to \mathsf{Nat}.\ \lambda n : \mathsf{Nat}.\ \mathsf{if}\ \left(\mathsf{iszero}\ n\right)\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{succ}\ \left(\mathsf{succ}\ \left(f\ \left(\mathsf{pred}\ n\right)\right)\right)\right)\right)\ \left(\mathsf{succ}\ \left(\mathsf{succ}\ 0\right)\right)$$

   You should sketch out the **overall structure of the reduction sequence, including the key steps involved in the operation of fix.**

   You **do not** need to present every single step or construct full derivations for the steps.

3.6 Concepts (40 pts)

In the spaces provided, **briefly explain** the following concepts from the course, **illustrating each by explaining a simple example:**

1. (10 pts) partial application of curried functions in Haskell

2. (10 pts) implementing languages via metaprogramming in Haskell

3. (10 pts) implementing Boolean logic in the untyped lambda calculus

4. (10 pts) the substitution lemma for simply-typed lambda calculus

END OF EXAM

A Reference: simply typed lambda calculus with booleans and numbers

Syntax

Terms, t	$::=$	x	*variable*
	$\mid$	$\lambda x : T.\, t$	*abstraction*
	$\mid$	$t\ t$	*application*
	$\mid$	true	*constant true*
	$\mid$	false	*constant false*
	$\mid$	if t then t else t	*conditional*
	$\mid$	0	*constant zero*
	$\mid$	succ t	*successor*
	$\mid$	pred t	*predecessor*
	$\mid$	iszero t	*zero test*
Values, v	$::=$	$\lambda x : T.\, t$	*abstraction value*
	$\mid$	true	*true value*
	$\mid$	false	*false value*
	$\mid$	nv	*numeric value*
Numeric values, nv	$::=$	0	*zero value*
	$\mid$	succ nv	*successor value*
Types, T	$::=$	$T \to T$	*type of functions*
	$\mid$	Bool	*type of booleans*
	$\mid$	Nat	*type of natural numbers*
Typing contexts, Γ	$::=$	$\varnothing$	*empty context*
	$\mid$	$\Gamma, x : T$	*variable type assumption*

Capture-Avoiding Substitution

$$
\begin{array}{rcl}
[x \mapsto t]x & = & t \\
[x \mapsto t]y & = & y \qquad \text{if } x \neq y \\
[x \mapsto t](t_1\ t_2) & = & ([x \mapsto t]t_1)\ ([x \mapsto t]t_2) \\
[x \mapsto t](\lambda x{:}T.\, t') & = & \lambda x{:}T.\, t' \\
[x \mapsto t](\lambda y{:}T.\, t') & = & \lambda y{:}T.\, ([x \mapsto t]t') \qquad \text{if } x \neq y \\
[x \mapsto t]\mathsf{true} & = & \mathsf{true} \\
[x \mapsto t]\mathsf{false} & = & \mathsf{false} \\
[x \mapsto t]\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 & = & \mathsf{if}\ ([x \mapsto t]t_1)\ \mathsf{then}\ ([x \mapsto t]t_2)\ \mathsf{else}\ ([x \mapsto t]t_3) \\
[x \mapsto t]0 & = & 0 \\
[x \mapsto t](\mathsf{succ}\ t') & = & \mathsf{succ}\ ([x \mapsto t]t') \\
[x \mapsto t](\mathsf{pred}\ t') & = & \mathsf{pred}\ ([x \mapsto t]t') \\
[x \mapsto t](\mathsf{iszero}\ t') & = & \mathsf{iszero}\ ([x \mapsto t]t')
\end{array}
$$

NB: this definition is capture avoiding if we assume that t is a closed term.

Evaluation Rules

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}\ (\text{E-App1})$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'}\ (\text{E-App2})$$

$$\frac{}{(\lambda x{:}T_{11}.\, t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12}}\ (\text{E-AppAbs})$$

$$\frac{}{\mathsf{if\ true\ then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow t_2}\ (\text{E-IfTrue})$$

$$\frac{}{\mathsf{if\ false\ then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow t_3}\ \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \longrightarrow \mathsf{if}\ t_1'\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3}\ \text{(E-If)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{succ}\ t_1 \longrightarrow \mathsf{succ}\ t_1'}\ \text{(E-Succ)}$$

$$\frac{}{\mathsf{pred}\ 0 \longrightarrow 0}\ \text{(E-PredZero)}$$

$$\frac{}{\mathsf{pred}\ (\mathsf{succ}\ nv_1) \longrightarrow nv_1}\ \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{pred}\ t_1 \longrightarrow \mathsf{pred}\ t_1'}\ \text{(E-Pred)}$$

$$\frac{}{\mathsf{iszero}\ 0 \longrightarrow \mathsf{true}}\ \text{(E-IsZeroZero)}$$

$$\frac{}{\mathsf{iszero}\ (\mathsf{succ}\ nv_1) \longrightarrow \mathsf{false}}\ \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{iszero}\ t_1 \longrightarrow \mathsf{iszero}\ t_1'}\ \text{(E-IsZero)}$$

Typing Rules

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-VAR)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash \lambda x{:}T_1.\, t_2 : T_1 \to T_2} \text{ (T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \text{ (T-APP)}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \text{ (T-TRUE)}$$

$$\frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}} \text{ (T-FALSE)}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 : T} \text{ (T-IF)}$$

$$\frac{}{\Gamma \vdash 0 : \mathsf{Nat}} \text{ (T-ZERO)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, t : \mathsf{Nat}} \text{ (T-SUCC)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}\, t : \mathsf{Nat}} \text{ (T-PRED)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{iszero}\, t : \mathsf{Bool}} \text{ (T-ISZERO)}$$

B Reference: extending the simply typed lambda-calculus with general recursion

Extended Syntax

$$\text{Extended terms, } t \quad ::= \quad \cdots \mid \mathsf{fix}\, t \qquad\qquad \textit{fixed point of } t$$

Extended Capture-Avoiding Substitution

$$[x \mapsto t](\mathsf{fix}\, t') \quad = \quad \mathsf{fix}\, ([x \mapsto t]t')$$

New Evaluation Rules

$$\frac{}{\mathsf{fix}\, (\lambda f : T_1.\, t_2) \longrightarrow [f \mapsto (\mathsf{fix}\, (\lambda f : T_1.\, t_2))]t_2}\ \text{(E-FixBeta)}$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{fix}\, t_1 \longrightarrow \mathsf{fix}\, t_1'}\ \text{(E-Fix)}$$

New Typing Rules

$$\frac{\Gamma \vdash t_1 : T_1 \to T_1}{\Gamma \vdash \mathsf{fix}\, t_1 : T_1}\ \text{(T-Fix)}$$