Final examination — given Thursday 12th December

General instructions

Closed-book, closed-notes, closed-computer, in-class exam.

Time allowed: 120 minutes.

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 Haskell datatypes (20 pts)

Write down the result of evaluating each of the following Haskell expressions.

1. (5 pts) `map (\x -> x * 2) [1,2,3,4,5]`

2. (5 pts) `filter (\x -> x ‘mod‘ 2 == 0) [1,2,3,4,5]`

3. (5 pts) `map (\x -> x ‘mod‘ 2 == 0) [1,2,3,4,5]`

4. (5 pts) `map (\x -> x * x) (take 3 [1..])`

3.2 Proving properties of Haskell list functionals (20 pts)

Prove, by induction on the structure of lists, that

```
map f (map g l) = map g (map f l)
```

for all lists `l :: [a]` and all functions `f :: a -> a` and `g :: a -> a` for which the equation `f (g x) = g (f x)` holds.

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

3.3 Arithmetic in the untyped lambda-calculus (20 pts)

The reduction rules for the call-by-value lambda-calculus are presented in Appendix A.

1. (10 pts) The "successor" function (succ) over Church numerals (c_i) in the untyped lambda-calculus can be defined as follows:

$$\mathsf{succ} = \lambda n.\, \lambda s.\, \lambda z.\, s\ (n\ s\ z).$$

Using this succ function, or otherwise, define an untyped lambda-term add that implements addition of Church numerals, so that $\mathsf{add}\ \mathsf{c}_i\ \mathsf{c}_j \longrightarrow_\beta^* \mathsf{c}_{i+j}$.

2. (10 pts) Using your add function, or otherwise, define an untyped lambda-term times that implements multiplication of Church numerals, so that $\mathsf{times}\ \mathsf{c}_i\ \mathsf{c}_j \longrightarrow_\beta^* \mathsf{c}_{i\times j}$.

3.4 Typing in the simply-typed lambda-calculus (40 pts)

Use the typing rules for the simply-typed lambda-calculus with booleans and numbers from Appendix B to construct a full typing derivation for each of the typing judgments below.

For each, you should **either** (1) determine the type T for which the judgment holds, **or** (2) determine that no such T exists, as appropriate in each case.

1. (10 pts) $\varnothing \vdash \lambda x\!:\!\mathsf{Nat}.\,\mathsf{succ}\,(\mathsf{pred}\,x) : T$

2. (10 pts) $\varnothing \vdash \lambda f\!:\!\mathsf{Bool} \to \mathsf{Nat}.\,\mathsf{iszero}\,(f\ y) : T$

3. (10 pts) $\varnothing, y : \mathsf{Bool} \vdash \lambda f\!:\!\mathsf{Bool} \to \mathsf{Nat}.\,\mathsf{iszero}\,(f\ y) : T$

4. (10 pts) $\varnothing, y : \mathsf{Nat} \vdash \lambda f\!:\!\mathsf{Bool} \to \mathsf{Nat}.\,\mathsf{iszero}\,(f\ y) : T$

3.5 Recursion in the simply-typed lambda-calculus (20 pts)

The reduction axiom for the `fix` extension to the call-by-value lambda-calculus is as follows:

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

The rest of the definition of this extension is presented in Appendix C.

Briefly explain how this rule implements recursive behavior, including the possibility of non-terminating reduction.

You must illustrate your answer by considering the reduction sequence for the expression:

$$\Big(\mathsf{fix}\,(\lambda f\!:\!\mathsf{Nat} \to \mathsf{Nat}.\,\lambda n\!:\!\mathsf{Nat}.\,f\ (\mathsf{succ}\,n))\Big)\ 0$$

3.6 Evaluation in the simply typed lambda-calculus (40 pts)

Using the evaluation rules for the simply-typed lambda calculus with booleans and numbers, as defined in Appendix B, **construct a full derivation for *every* step of evaluation** of the following terms.

In each case, state whether evaluation terminates due to **reaching a value**, or due to **getting stuck at a non-value term that cannot be reduced further.**

1. (20 pts) $\mathsf{pred}\,((\lambda b\!:\!\mathsf{Bool}.\ \mathsf{if}\ b\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{succ}\ 0)\ \mathsf{false})$

2. (20 pts) $(\lambda x : \mathsf{Nat}.\ \mathsf{iszero}\ x)\ (\mathsf{if\ true\ then\ false\ else\ true})$

3.7 Concepts (40 pts)

In the spaces provided, **briefly explain** the following **four** concepts from the course:

1. (10 pts) the manipulation of infinite lists in Haskell

2. (10 pts) type variables and polymorphic types in Haskell

3. (10 pts) constraint-based typing and type unification

4. (10 pts) type safety

END OF EXAM

This page intentionally left (almost) blank

A Reference: untyped call-by-value lambda-calculus

Syntax

$$\begin{array}{rcll} \text{Terms, } t & ::= & x & \textit{variable} \\ & | & \lambda x.\, t & \textit{abstraction} \\ & | & t\ t & \textit{application} \\ \\ \text{Values, } v & ::= & \lambda x.\, t & \textit{abstraction value} \end{array}$$

Evaluation Rules

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

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

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

Capture-avoiding substitution

$$\begin{array}{rcll} [x \mapsto t]x & = & t & \\ [x \mapsto t]y & = & y & \text{if } x \neq y \\ [x \mapsto t](t_1\ t_2) & = & ([x \mapsto t]t_1)\ ([x \mapsto t]t_2) & \\ [x \mapsto t](\lambda x.\, t') & = & \lambda x.\, t' & \\ [x \mapsto t](\lambda y.\, t') & = & \lambda y.\, ([x \mapsto t]t') & \text{if } x \neq y \end{array}$$

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

B 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*
	$\mid$	$\mathsf{if}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t$	*constant false*
	$\mid$	$\mathsf{0}$	*constant zero*
	$\mid$	$\mathsf{succ}\ t$	*successor*
	$\mid$	$\mathsf{pred}\ t$	*predecessor*
	$\mid$	$\mathsf{iszero}\ t$	*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	$::=$	$\mathsf{0}$	*zero value*
	$\mid$	$\mathsf{succ}\ nv$	*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

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

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

Evaluation Rules

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

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

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

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

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

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

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

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

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

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

$$\frac{}{\mathsf{iszero}\ 0 \longrightarrow \mathsf{true}}\ (\textsc{E-IszeroZero})$$

$$\frac{}{\mathsf{iszero}\ (\mathsf{succ}\ nv_1) \longrightarrow \mathsf{false}}\ (\textsc{E-IszeroSucc})$$

$$\frac{t_1 \longrightarrow t_1'}{\mathsf{iszero}\ t_1 \longrightarrow \mathsf{iszero}\ t_1'}\ (\textsc{E-Iszero})$$

Typing Rules

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

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

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

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

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

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

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

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

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

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

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

Extended Syntax

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

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

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