Mid-term examination #2 — given Tuesday 10th November

General instructions

Online, take-home exam.

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

Total points available: 150 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:

2.1 Concepts in syntax and semantics of programming languages (40 pts)

Briefly explain the following concepts in the syntax and semantics of programming languages. **Illustrate each answer with an explanation of a simple example.**

1. (10 pts) Context-free grammars.

2. (10 pts) Small-step operational semantics.

3. (10 pts) Reduction of terms getting "stuck."

4. (10 pts) Type safety.

2.2 Typing judgments (30 pts)

1. (15 pts) Using the typing rules for the typed calculus of numbers and booleans, as defined in Appendix A, identify the type T such that

$$\mathsf{pred}\ \big(\mathsf{if}\ (\mathsf{iszero}\ (\mathsf{succ}\ (\mathsf{pred}\ 0)))\ \mathsf{then}\ \mathsf{pred}\ 0\ \mathsf{else}\ 0\big) : T$$

holds, if such a type exists, by **constructing a full typing derivation for this judgment.** If no such type exists, construct a partial derivation and show how and why the typing derivation fails.

2. (15 pts) Using the typing rules for the typed calculus of numbers and booleans, as defined in Appendix A, identify the type T such that

$$\mathsf{iszero}\ \big(\mathsf{if}\ (\mathsf{iszero}\ (\mathsf{succ}\ 0))\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ 0\big) : T$$

holds, if such a type exists, by **constructing a full typing derivation for this judgment.** If no such type exists, construct a partial derivation and show how and why the typing derivation fails.

2.3 Untyped call-by-value lambda-calculus—evaluation (30 pts)

Using the evaluation rules for the untyped call-by-value lambda calculus, as defined in Appendix B, **construct a full derivation for *every* step of evaluation** of the term

$$\Big(\big(\lambda f.\, \lambda g.f\ (\lambda y.\, y\ y)\big)\ (\lambda x.\, x)\Big)\ \Big(\lambda y.\, y\ y\Big)$$

and identify when you have reduced the term to a value.

2.4 Programming in the untyped lambda-calculus (50 pts)

1. (20 pts) Define an untyped lambda-term *xnor* that represents the corresponding two-input Boolean operation, whose behavior is specified as follows:

$$(xnor\ fls)\ fls \longrightarrow^*_\beta tru \qquad (xnor\ fls)\ tru \longrightarrow^*_\beta fls$$

$$(xnor\ tru)\ fls \longrightarrow^*_\beta fls \qquad (xnor\ tru)\ tru \longrightarrow^*_\beta tru$$

   where the lambda-terms *tru* and *fls* are as defined in class, and where $\longrightarrow^*_\beta$ represents 0 or more applications of full-beta reduction.

   Define any helper functions that you use in your answer.

2. (10 pts) Write down definitions for the untyped lambda-terms corresponding to the Church numerals c_0, c_1, c_2, and c_3, which represent the natural numbers 0, 1, 2, and 3, respectively.

3. (20 pts) The desired behavior of the untyped lambda-term *times* that implements multiplication of two Church numerals m and n can be specified as follows:

$$times\ \mathsf{c}_m\ \mathsf{c}_n \longrightarrow_\beta^* \mathsf{c}_{m \times n}$$

where $\longrightarrow_\beta^*$ represents 0 or more applications of full-beta reduction.

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

You may assume untyped lambda-terms *succ* and *add* as defined in class.

A Reference: the typed calculus of numbers and booleans

Syntax

Terms, t	::=	true	*constant true*
	|	false	*constant false*
	|	if t then t else t	*conditional*
	|	0	*constant zero*
	|	succ t	*successor*
	|	pred t	*predecessor*
	|	iszero t	*zero test*
Values, v	::=	true	*true value*
	|	false	*false value*
	|	nv	*numeric value*
Numeric values, nv	::=	0	*zero value*
	|	succ nv	*successor value*
Types, T	::=	Bool	*type of booleans*
	|	Nat	*type of natural numbers*

Evaluation Rules

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

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

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

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

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

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

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

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

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

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

Typing Rules

$$\frac{}{\mathsf{true} : \mathsf{Bool}} \text{ (T-True)}$$

$$\frac{}{\mathsf{false} : \mathsf{Bool}} \text{ (T-False)}$$

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

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

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

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

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

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

Syntax

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

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.