Midterm Exam

15-814 Types and Programming Languages
Frank Pfenning

October 17, 2019

Name: **Sample Solution** Andrew ID: **fp**

Instructions

- This exam is closed-book, closed-notes.
- You have 80 minutes to complete the exam.
- There are 4 problems.
- For reference, on pages 9–11 there is an appendix with sections on the syntax, statics, and dynamics.

	λ-Calculus	Type Isomorphisms	Fork/Join Parallelism	Small-Step Determinacy	
	Prob 1	Prob 2	Prob 3	Prob 4	Total
Score	**40**	**30**	**50**	**30**	**150**
Max	40	30	50	30	150

1 λ-Calculus (40 pts)

Recall the definition of Church numerals in the λ-calculus:

$$\overline{n} = \lambda s.\, \lambda z.\, \underbrace{s\,(s\,\ldots(s}_{n \text{ times}}\, z))$$

Task 1 (20 pts). Fill in the missing definitions. You may use any definition in all subsequent answers, including function composition in infix notation $(f \circ g)$.

zero	$=_\beta$	$\overline{0}$
zero	$=$	$\lambda s.\, \lambda z.\, z$
succ $\overline{n}$	$=_\beta$	$\overline{n+1}$
succ	$=$	$\lambda n.\, \lambda s.\, \lambda z.\, s\,(n\,s\,z)$
compose f g		function composition, usually written in infix notation as $f \circ g$
compose	$=$	$\lambda f.\, \lambda g.\, \lambda x.\, f\,(g\,x)$
double $\overline{n}$	$=_\beta$	$\overline{2n}$
double	$=$	$\lambda n.\, n\,(\mathit{succ} \circ \mathit{succ})\ \mathit{zero}$
mystery $\overline{n}$	$=_\beta$	$\overline{2^n}$
mystery	$=$	$\lambda n.\, n\ \mathit{double}\ (\mathit{succ}\ \mathit{zero})$

Next, we consider a Church-style representation of numbers in base 2, defined with

$$(a_n \cdots a_1 a_0)_2 = a_n 2^n + \ldots a_1 2^1 + a_0 2^0$$

where each a_i is either 0 or 1. We then define $\overline{\overline{0}} = b0$ and $\overline{\overline{1}} = b1$ and

$$\ulcorner (a_n \cdots a_1 a_0)_2 \urcorner = \lambda b1.\, \lambda b0.\, \lambda e.\, \overline{\overline{a_0}}\, (\overline{\overline{a_1}}\, \ldots\, (\overline{\overline{a_n}}\, e))$$

For example, $\ulcorner 6 \urcorner = \ulcorner (110)_2 \urcorner = \lambda b1.\, \lambda b0.\, \lambda e.\, b0\, (b1\, (b1\, e))$. As a special case, we represent the number 0 as shown below with zero binary digits.

Task 2 (20 pts). Complete the following definitions, where you may use any definitions in subsequent answers, including all the definitions from Task 1.

bzero $=_\beta$ $\ulcorner 0 \urcorner$

bzero $=$ $\lambda b1.\, \lambda b0.\, \lambda e.\, e$

btwo $=_\beta$ $\ulcorner 2 \urcorner = \ulcorner (10)_2 \urcorner$

btwo $=$ $\boxed{\lambda b1.\, \lambda b0.\, \lambda e.\, b0\, (b1\, e)}$

bdouble $\ulcorner x \urcorner$ $=_\beta$ $\ulcorner 2x \urcorner$

bdouble $=$ $\boxed{\lambda x.\, \lambda b1.\, \lambda b0.\, \lambda e.\, b0\, (x\, b1\, b0\, e)}$

bin2nat $\ulcorner x \urcorner$ $=_\beta$ $\overline{x}$

bin2nat $=$ $\boxed{\lambda x.\, x\ (\mathit{succ} \circ \mathit{double})\ \mathit{double}\ \mathit{zero}}$

bmystery $\ulcorner x \urcorner$ $=_\beta$ $\boxed{\mathit{true} \text{ if } x \text{ is even, } \mathit{false} \text{ otherwise}}$

bmystery $=$ $\lambda x.\, x\ (\lambda y.\, \mathit{false})\ (\lambda z.\, \mathit{true})\ \mathit{true}$

2 Type Isomorphism (30 pts)

Recall that two types τ and σ are *isomorphic* if we can supply a pair of functions $\mathit{Forth} : \tau \to \sigma$ and $\mathit{Back} : \sigma \to \tau$ such that $\mathit{Back} \circ \mathit{Forth}$ and $\mathit{Forth} \circ \mathit{Back}$ are both equal to the identity function. As in lectures and homework assignments, we take here an extensional point of view, that is, two functions are equal if applied to an arbitrary value v of the correct type they yield equal results.

We define

$$\begin{array}{lcl} 2 & = & (\mathtt{zero} : 1) + (\mathtt{one} : 1) \\ \mathit{bin} & = & \rho\alpha.\,(\mathtt{E} : 1) + (\mathtt{B1} : \alpha) + (\mathtt{B0} : \alpha) \end{array}$$

Task 1 (30 pts). Define functions *Forth* and *Back* witnessing the isomorphism of $2 \times \mathit{bin} + 1 \cong \mathit{bin}$, using the following labels:

$$(\mathtt{lft} : 2 \times \mathit{bin}) + (\mathtt{rgt} : 1) \cong \mathit{bin}$$

You may use general pattern matching in your definition. You do not need to prove that the functions form an isomorphism.

$$\mathit{Forth} \quad : \quad (\mathtt{lft} : 2 \times \mathit{bin}) + (\mathtt{rgt} : 1) \to \mathit{bin}$$

$$\begin{array}{lcl} \mathit{Forth} & = & \lambda s.\,\mathsf{case}\ s\ (\ \mathtt{lft} \cdot \langle \mathtt{zero} \cdot \langle\,\rangle, x\rangle \Rightarrow \mathsf{fold}\ (\mathtt{B0} \cdot x) \\ & & \quad \mid \mathtt{lft} \cdot \langle \mathtt{one} \cdot \langle\,\rangle, x\rangle \Rightarrow \mathsf{fold}\ (\mathtt{B1} \cdot x) \\ & & \quad \mid \mathtt{rgt} \cdot \langle\,\rangle \Rightarrow \mathsf{fold}\ (\mathtt{E} \cdot \langle\,\rangle)) \end{array}$$

$$\mathit{Back} \quad : \quad \mathit{bin} \to (\mathtt{lft} : 2 \times \mathit{bin}) + (\mathtt{rgt} : 1)$$

$$\begin{array}{lcl} \mathit{Back} & = & \lambda x.\,\mathsf{case}\ x\ (\ \mathsf{fold}\ (\mathtt{E} \cdot \langle\,\rangle) \Rightarrow \mathtt{rgt} \cdot \langle\,\rangle \\ & & \quad \mid \mathsf{fold}\ (\mathtt{B0} \cdot y) \Rightarrow \mathtt{lft} \cdot \langle \mathtt{zero} \cdot \langle\,\rangle, y\rangle \\ & & \quad \mid \mathsf{fold}\ (\mathtt{B1} \cdot y) \Rightarrow \mathtt{lft} \cdot \langle \mathtt{one} \cdot \langle\,\rangle, y\rangle) \end{array}$$

3 Fork/Join Parallelism (50 pts)

Fork/join parallelism is the idea that we can *fork* the parallel evaluation of two expression and then *join* these two threads when they have both finished.

We model this with a *parallel pair* $\tau_1 \odot \tau_2$. The new expressions are $\langle e_1 \parallel e_2 \rangle$ to construct a parallel pair and case e $(\langle x_1 \parallel x_2 \rangle \Rightarrow e')$ to decompose it.

The typing rules are not very interesting, because they work exactly like the typing of constructors and destructors of ordinary eager pairs. So we do not write them out.

Regarding the dynamics, here are several examples to illustrate the desired behavior. We write v for expressions with v *val* and $\bot = \mathsf{fix}\ f.\ f$.

$\langle e_1 \parallel \bot \rangle$		does not have a value
$\langle \bot \parallel e_2 \rangle$		does not have a value
$\langle v_1 \parallel v_2 \rangle$		is a value
$\langle (\lambda x\, x)\, v_1 \parallel (\lambda x.\, \lambda y.\, x)\, v_2\, v_3 \rangle$	$\mapsto^2$	$\langle v_1 \parallel v_2 \rangle$

When writing down the dynamics, make sure that preservation and progress continue to hold.

Task 1 (5 pts). Give the rule(s) for the e *val* judgment for the new expressions.

$$\frac{e_1\ \mathit{val} \quad e_2\ \mathit{val}}{\langle e_1 \parallel e_2 \rangle\ \mathit{val}}\ \mathsf{val/ppair}$$

Task 2 (20 pts). Give the rules for the $e \mapsto e'$ judgment for the new expressions.

$$\frac{e_1 \mapsto e_1' \quad e_2 \mapsto e_2'}{\langle e_1 \parallel e_2 \rangle \mapsto \langle e_1' \parallel e_2' \rangle}\ \mathsf{step/ppair}_0$$

$$\frac{e_1 \mapsto e_1' \quad v_2\ \mathit{val}}{\langle e_1 \parallel v_2 \rangle \mapsto \langle e_1' \parallel v_2 \rangle}\ \mathsf{step/ppair}_1 \qquad \frac{v_1\ \mathit{val} \quad e_2 \mapsto e_2'}{\langle v_1 \parallel e_2 \rangle \mapsto \langle v_1 \parallel e_2' \rangle}\ \mathsf{step/ppair}_2$$

$$\frac{e_0 \mapsto e_0'}{\mathsf{case}\ e_0\ (\langle x_1 \parallel x_2 \rangle \Rightarrow e_3) \mapsto \mathsf{case}\ e_0'\ (\langle x_1 \parallel x_2 \rangle \Rightarrow e_3)}\ \mathsf{step/case/ppair}_0$$

$$\frac{v_1\ \mathit{val} \quad v_2\ \mathit{val}}{\mathsf{case}\ \langle v_1 \parallel v_2 \rangle\ (\langle x_1 \parallel x_2 \rangle \Rightarrow e_3) \mapsto [v_1/x_1][v_2/x_2]e_3}\ \mathsf{step/case/ppair}$$

Task 3 (20 pts). Fill in the gaps in the statement and one case in the proof of preservation.

Theorem (Preservation)
If $\cdot \vdash e : \tau$ and $\boxed{e \mapsto e'}$ then $\boxed{\cdot \vdash e' : \tau}$.

Proof. By $\boxed{\text{rule induction on the derivation of } e \mapsto e'}$

Case: In the rule where two expressions step in parallel, we have

$$\frac{e_1 \mapsto e_1' \quad e_2 \mapsto e_2'}{\langle e_1 \parallel e_2 \rangle \mapsto \langle e_1' \parallel e_2' \rangle}\ \mathsf{step/ppair}_0$$

with $e = \langle e_1 \parallel e_2 \rangle$ and $e' = \langle e_1' \parallel e_2' \rangle$. Then

$\cdot \vdash \langle e_1 \parallel e_2 \rangle : \tau$	Assumption
$\cdot \vdash e_1 : \tau_1$ and $\cdot \vdash e_2 : \tau_2$ and $\tau = \tau_1 \odot \tau_2$ for some τ_1 and τ_2	By inversion
$\cdot \vdash e_1' : \tau_1$	By ind. hyp.
$\cdot \vdash e_2' : \tau_2$	By ind. hyp.
$\cdot \vdash \langle e_1' \parallel e_2' \rangle : \tau_1 \odot \tau_2$	By rule

Task 4 (5 pts). Complete the statement of the progress theorem and the global structure of the proof. You do not need to show any cases.

Theorem (Progress)
If $\boxed{\cdot \vdash e : \tau}$ then either $e \mapsto e'$ for some e' or e *val*.

Proof. By $\boxed{\text{rule induction on the derivation of } \cdot \vdash e : \tau}$

4 Small-Step Determinacy (30 pts)

As noted during the midterm review session, in the proof that our language from the appendix satisfies small-step determinacy we may need the following two lemmas. We write $e \not\mapsto$ if there is no e' such that $e \mapsto e'$.

Task 1 (5 pts).

Lemma A If $\cdot \vdash e : \tau$ and e *val* then $e \not\mapsto$.
Circle one: This lemma follows directly from the progress theorem.

YES / [NO]

If your answer is NO: the statement can be proved by [rule induction on the derivation of e *val*]

Task 2 (5 pts).

Lemma B If $\cdot \vdash e : \tau$ and $e \not\mapsto$ then e *val*.
Circle one: This lemma follows directly from the progress theorem.

[YES] / NO

If your answer is NO: the statement can be proved by [*does not apply*]

Task 3 (15 pts). Complete the following portion of the proof of small-step determinacy.

Theorem (Small-Step Determinacy). If $\cdot \vdash e : \tau$ and $e \mapsto e'$ and $e \mapsto e''$ then $e' = e''$.
Proof: By rule induction on the derivation of $e \mapsto e'$.

Case:

$$\frac{e_1 \mapsto e_1'}{\langle e_1, e_2 \rangle \mapsto \langle e_1', e_2 \rangle}\ \mathsf{step/pair}_1$$

where $e = \langle e_1, e_2 \rangle$ and $e' = \langle e_1', e_2 \rangle$.

$\cdot \vdash e_1 : \tau_1$ for some τ_1 By [inversion on $\cdot \vdash \langle e_1, e_2 \rangle : \tau$]

We then apply inversion on [$\langle e_1, e_2 \rangle \mapsto e''$] and obtain [two] subcase(s).

State and complete each subcase below.

First subcase: $\langle e_1, e_2 \rangle \mapsto \langle e_1'', e_2 \rangle$ by rule $\mathsf{step/pair}_1$ and $e_1 \mapsto e_1''$. Then $e_1' = e_1''$ by induction hypothesis and therefore $e' = \langle e_1', e_2 \rangle = \langle e_1'', e_2 \rangle = e''$

Second subcase: $\langle e_1, e_2 \rangle \mapsto \langle e_1, e_2' \rangle$ where e_1 *val* by rule $\mathsf{step/pair}_2$. But this is impossible by Lemma A above since $e_1 \mapsto e_1'$.

Task 4 (5 pts). Does your set of rules in Problem 3 on fork/join parallelism satisfy small-step determinacy? Circle one:

[YES] / NO

Appendix: Language Reference

Language

$$
\begin{array}{llll}
\text{Types} & \tau & ::= & \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \sum_{i \in I}(i : \tau_i) \mid \rho\alpha.\,\tau \\
\text{Expressions} & e & ::= & x \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (\text{variables}) \\
 & & \mid & \lambda x.\, e \mid e_1\, e_2 \qquad\qquad\qquad\qquad\qquad\qquad\quad (\to) \\
 & & \mid & \langle e_1, e_2 \rangle \mid \mathsf{case}\ e\ (\langle x_1, x_2 \rangle \Rightarrow e') \qquad\quad (\times) \\
 & & \mid & \langle\,\rangle \mid \mathsf{case}\ e\ (\langle\,\rangle \Rightarrow e') \qquad\qquad\qquad\quad (1) \\
 & & \mid & j \cdot e \mid \mathsf{case}\ e\ (i \cdot x_i \Rightarrow e_i)_{i \in I} \qquad\qquad\quad (\textstyle\sum) \\
 & & \mid & \mathsf{fold}\ e \mid \mathsf{unfold}\ e \qquad\qquad\qquad\qquad\quad (\rho) \\
 & & \mid & f \mid \mathsf{fix}\ f.\, e \qquad\qquad\qquad\qquad\qquad\qquad\quad (\text{recursion}) \\
\text{Contexts} & \Gamma & ::= & x_1 : \tau_1, \ldots, x_n : \tau_n \quad (\text{all } x_i \text{ distinct})
\end{array}
$$

Statics and Dynamics

Functions.

$$\frac{\Gamma, x_1 : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda x_1.\, e_2 : \tau_1 \to \tau_2}\ \mathsf{lam} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \mathsf{var}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau_1}\ \mathsf{app}$$

$$\frac{}{\lambda x.\, e\ \mathit{val}}\ \mathsf{val/lam}$$

$$\frac{e_1 \mapsto e_1'}{e_1\, e_2 \mapsto e_1'\, e_2}\ \mathsf{step/app}_1 \qquad \frac{v_1\ \mathit{val} \quad e_2 \mapsto e_2'}{v_1\, e_2 \mapsto v_1\, e_2'}\ \mathsf{step/app}_2$$

$$\frac{v_2\ \mathit{val}}{(\lambda x.\, e_1)\, v_2 \mapsto [v_2/x]e_1}\ \mathsf{beta}$$

Products.

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}\ \mathsf{pair}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e' : \tau'}{\Gamma \vdash \mathsf{case}\ e\ (\langle x_1, x_2 \rangle \Rightarrow e') : \tau'}\ \mathsf{case/pair}$$

$$\frac{e_1\ \mathit{val} \quad e_2\ \mathit{val}}{\langle e_1, e_2\rangle\ \mathit{val}}\ \mathsf{val/pair}$$

$$\frac{e_1 \mapsto e_1'}{\langle e_1, e_2\rangle \mapsto \langle e_1', e_2\rangle}\ \mathsf{step/pair}_1 \qquad \frac{e_1\ \mathit{val} \quad e_2 \mapsto e_2'}{\langle e_1, e_2\rangle \mapsto \langle e_1, e_2'\rangle}\ \mathsf{step/pair}_2$$

$$\frac{e_0 \mapsto e_0'}{\mathsf{case}\ e_0\ (\langle x_1, x_2\rangle \Rightarrow e_3) \mapsto \mathsf{case}\ e_0'\ (\langle x_1, x_2\rangle \Rightarrow e_3)}\ \mathsf{step/case/pair}_0$$

$$\frac{v_1\ \mathit{val} \quad v_2\ \mathit{val}}{\mathsf{case}\ \langle v_1, v_2\rangle\ (\langle x_1, x_2\rangle \Rightarrow e_3) \mapsto [v_1/x_1][v_2/x_2]e_3}\ \mathsf{step/case/pair}$$

Unit.

$$\frac{}{\Gamma \vdash \langle\,\rangle : 1}\ \mathsf{unit} \qquad \frac{\Gamma \vdash e_0 : 1 \quad \Gamma \vdash e' : \tau}{\Gamma \vdash \mathsf{case}\ e_0\ (\langle\,\rangle \Rightarrow e') : \tau}\ \mathsf{case/unit}$$

$$\frac{}{\langle\,\rangle\ \mathsf{val}}\ \mathsf{val/unit}$$

$$\frac{e_0 \mapsto e_0'}{\mathsf{case}\ e_0\ (\langle\,\rangle \Rightarrow e_1) \mapsto \mathsf{case}\ e_0'\ (\langle\,\rangle \Rightarrow e_1)}\ \mathsf{step/case/unit}_0$$

$$\frac{}{\mathsf{case}\ \langle\,\rangle\ (\langle\,\rangle \Rightarrow e_1) \mapsto e_1}\ \mathsf{step/case/unit}$$

Sums.

$$\frac{j \in I \quad \Gamma \vdash e : \tau_j}{\Gamma \vdash j \cdot e : \sum_{i\in I}(i : \tau_i)}\ \mathsf{sum} \qquad \frac{\Gamma \vdash e_0 : \sum_{i\in I}(i : \tau_i) \quad \Gamma, x_i : \tau_i \vdash e_i : \tau \quad \text{for all } i \in I}{\Gamma \vdash \mathsf{case}\ e_0\ (i \cdot x_i \Rightarrow e_i)_{i\in I} : \tau}\ \mathsf{case/sum}$$

$$\frac{e\ \mathit{val}}{j \cdot e\ \mathit{val}}\ \mathsf{val/sum}$$

$$\frac{e \mapsto e'}{j \cdot e \mapsto j \cdot e'}\ \mathsf{step/sum}$$

$$\frac{e_0 \mapsto e_0'}{\mathsf{case}\ e_0\ (i \cdot x_i \Rightarrow e_i)_{i\in I} \mapsto \mathsf{case}\ e_0'\ (i \cdot x_i \Rightarrow e_i)_{i\in I}}\ \mathsf{step/case/sum}_0$$

$$\frac{v\ \mathit{val}}{\mathsf{case}\ (j \cdot v)\ (i \cdot x_i \Rightarrow e_i)_{i\in I} \mapsto [v/x_j]e_j}\ \mathsf{step/case/sum}$$

Recursive Types.

$$\frac{\Gamma \vdash e : [\rho\alpha.\,\tau/\alpha]\tau}{\Gamma \vdash \mathsf{fold}\ e : \rho\alpha.\,\tau}\ \mathsf{fold} \qquad \frac{\Gamma \vdash e : \rho\alpha.\,\tau}{\Gamma \vdash \mathsf{unfold}\ e : [\rho\alpha.\,\tau/\alpha]\tau}\ \mathsf{unfold}$$

$$\frac{e\ \mathit{val}}{\mathsf{fold}\ e\ \mathit{val}}\ \mathsf{val/fold}$$

$$\frac{e \mapsto e'}{\mathsf{fold}\ e \mapsto \mathsf{fold}\ e'}\ \mathsf{step/fold}$$

$$\frac{e \mapsto e'}{\mathsf{unfold}\ e \mapsto \mathsf{unfold}\ e'}\ \mathsf{step/unfold}_0 \qquad \frac{v\ \mathit{val}}{\mathsf{unfold}\ (\mathsf{fold}\ v) \mapsto v}\ \mathsf{step/unfold}$$

Fixed Point Expressions.

$$\frac{\Gamma, f : \tau \vdash e : \tau}{\Gamma \vdash \mathsf{fix}\ f.\,e : \tau}\ \mathsf{fix}$$

$$\frac{}{\mathsf{fix}\ f.\,e \mapsto [\mathsf{fix}\ f.\,e/f]e}\ \mathsf{step/fix}$$