Midterm Exam

15-814 Types and Programming Languages
Frank Pfenning
October 17, 2019

Name: Andrew ID:

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.

<table>
<thead>
<tr>
<th></th>
<th>λ-Calculus</th>
<th>Type Isomorphisms</th>
<th>Fork/Join Parallelism</th>
<th>Small-Step Determinacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Score

Max 40 30 50 30 150
1  λ-Calculus (40 pts)

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

\[ \bar{n} = \lambda s. \lambda z. s (s \ldots (s z)) \]

\( n \) times

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

\[
\begin{align*}
\text{zero} & = \beta 0 \\
\text{zero} & = \lambda s. \lambda z. z \\
\text{succ } \bar{n} & = \beta \bar{n + 1} \\
\text{succ} & = \quad \quad \quad \\
\text{compose } f \circ g & \quad \text{function composition, usually written in infix notation as } f \circ g \\
\text{compose} & = \\
\text{double } \bar{n} & = \beta 2\bar{n} \\
\text{double} & = \\
\text{mystery } \bar{n} & = \beta \lambda n. n \, \text{double} \, (\text{succ } \text{zero}) \\
\text{mystery} & = \lambda n. n \, \text{double} \, (\text{succ } \text{zero})
\end{align*}
\]
Next, we consider a Church-style representation of numbers in base 2, defined with

\[
(a_n \cdots a_1 a_0)_2 = a_n 2^n + \cdots + a_1 2^1 + a_0 2^0
\]

where each \( a_i \) is either 0 or 1. We then define \( \bar{0} = b0 \) and \( \bar{1} = b1 \) and

\[
\bar{(a_n \cdots a_1 a_0)} = \lambda b_1. \lambda b_0. \lambda e. a_0 (a_1 \ldots (a_n e))
\]

For example, \( \bar{6} = \bar{(110)_2} = \lambda b_1. \lambda b_0. \lambda e. b_0 (b_1 e) \). 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.

\[
\begin{align*}
\text{bzero} &= \beta \bar{0}^\gamma \\
\text{bzero} &= \lambda b_1. \lambda b_0. \lambda e. \ e \\
\text{btwo} &= \beta \bar{2}^\gamma = \bar{(10)_2}^\gamma \\
\text{btwo} &= \\
\text{bdouble} &= \beta \bar{2x}^\gamma \\
\text{bdouble} &= \\
\text{bin2nat} &= \beta \bar{x} \\
\text{bin2nat} &= \\
\text{bmystery} &= \beta \lambda x. x (\lambda y. \text{false}) (\lambda z. \text{true}) \text{true}
\end{align*}
\]
2 Type Isomorphism (30 pts)

Recall that two types $\tau$ and $\sigma$ are isomorphic if we can supply a pair of functions $Forth : \tau \rightarrow \sigma$ and $Back : \sigma \rightarrow \tau$ such that $Back \circ Forth$ and $Forth \circ Back$ 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

\[
2 = (\text{zero} : 1) + (\text{one} : 1) \\
\text{bin} = \rho \alpha. (E : 1) + (B_1 : \alpha) + (B_0 : \alpha)
\]

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

\[
(lft : 2 \times \text{bin}) + (rgt : 1) \cong \text{bin}
\]

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

\[
Forth : (lft : 2 \times \text{bin}) + (rgt : 1) \rightarrow \text{bin}
\]

\[
Forth =
\]

\[
Back : \text{bin} \rightarrow (lft : 2 \times \text{bin}) + (rgt : 1)
\]

\[
Back =
\]
3 Fork/Join Parallelism (50 pts)

Fork/join parallelism is the idea that we can fork the parallel evaluation of two expressions 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 $e \langle (x_1 \parallel x_2) \Rightarrow e' \rangle$ 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 $v \text{ val}$ and $\perp = \text{fix}_f.f$.

$$\langle e_1 \parallel \perp \rangle \quad \text{does not have a value}$$
$$\langle \perp \parallel e_2 \rangle \quad \text{does not have a value}$$
$$\langle v_1 \parallel v_2 \rangle \quad \text{is a value}$$
$$\langle (\lambda x.x) v_1 \parallel (\lambda x.\lambda y.x) v_2 v_3 \rangle \rightarrow^2 \langle v_1 \parallel v_2 \rangle$$

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

**Task 1** (5 pts). Give the rule(s) for the $e \text{ val}$ judgment for the new expressions.

**Task 2** (20 pts). Give the rules for the $e \rightarrow e'$ judgment for the new expressions.
Task 3 (20 pts). Fill in the gaps in the statement and one case in the proof of preservation.

Theorem (Preservation)
If $\vdash e : \tau$ and $\ldots$ then $\ldots$.

Proof. By $\ldots$

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

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 $\ldots$ then either $e \rightarrow e'$ for some $e'$ or $e$ val.

Proof. By $\ldots$
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 $e \not\rightarrow$ if there is no $e'$ such that $e \leftrightarrow e'$.

Task 1 (5 pts).

**Lemma A** If $\cdot \vdash e : \tau$ and $e \text{ val}$ then $e \not\rightarrow$.

Circle one: This lemma follows directly from the progress theorem.

YES / NO

If your answer is NO: the statement can be proved by

Task 2 (5 pts).

**Lemma B** If $\cdot \vdash e : \tau$ and $e \not\rightarrow$ then $e \text{ val}$.

Circle one: This lemma follows directly from the progress theorem.

YES / NO

If your answer is NO: the statement can be proved by
**Task 3** (15 pts). Complete the following portion of the proof of small-step determinacy.

**Theorem (Small-Step Determinacy).** If $\vdash e : \tau$ and $e \rightarrow e'$ and $e \rightarrow e''$ then $e' = e''$.

**Proof:** By rule induction on the derivation of $e \rightarrow e'$.

Case:

\[
\frac{e_1 \rightarrow e_1'}{(e_1, e_2) \rightarrow (e_1', e_2)}_{\text{step/pair}_1}
\]

where $e = (e_1, e_2)$ and $e' = (e_1', e_2)$.

$\vdash e_1 : \tau_1$ for some $\tau_1$

By

We then apply inversion on and obtain subcase(s).

*State and complete each subcase below.*

---

**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

Types  \( \tau ::= \alpha | \tau_1 \rightarrow \tau_2 | \tau_1 \times \tau_2 | 1 | \sum_{i \in I} (i : \tau_i) | \rho \alpha. \tau \)

Expressions  \( e ::= x \) (variables)
   \( \lambda x. e \) \( \rightarrow \) (\( \rightarrow \))
   \( \langle e_1, e_2 \rangle \) \( \case \) \( (x_1, x_2) \Rightarrow e' \) (\( \times \))
   \( j \cdot e \) \( \case \) \( (i \cdot x_i) \Rightarrow e_i \) \( i \in I \) (\( \sum \))
   \( \text{fold } e \) \( \text{unfold } e \) (\( \rho \))
   \( f \) \( \text{fix } f. e \) (recursion)

Contexts  \( \Gamma ::= x_1 : \tau_1, \ldots, x_n : \tau_n \) (all \( x_i \) distinct)

Statics and Dynamics

Functions.

\[
\begin{align*}
\Gamma, x_1 : \tau_1 &\vdash e_2 : \tau_2 \\
\Gamma &\vdash \lambda x_1. e_2 : \tau_1 \rightarrow \tau_2 \\
\Gamma &\vdash x : \tau \in \Gamma \\
\Gamma &\vdash x : \tau
\end{align*}
\]

\[
\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau_1}
\]

\[
\begin{align*}
\lambda x. e &\text{ val/lam} \\
\lambda x. e &\text{ val/lam} \\
\lambda x. e &\text{ val/lam}
\end{align*}
\]

\[
\begin{align*}
e_1 \mapsto e_1' &\text{ step/app}_1 \\
e_1 e_2 \mapsto e'_1 e_2 &\text{ step/app}_2 \\
v_1 \text{ val} &\text{ step/app}_1 \\
v_1 e_2 \mapsto v_1 e'_2 &\text{ step/app}_2 \\
v_2 \text{ val} &\text{ beta}
\end{align*}
\]

Products.

\[
\begin{align*}
\Gamma &\vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \\
\Gamma &\vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2
\end{align*}
\]

\[
\begin{align*}
\Gamma &\vdash e : \tau_1 \times \tau_2 \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e' : \tau' \\
\Gamma &\vdash \text{case } e \ ((x_1, x_2) \Rightarrow e') : \tau'
\end{align*}
\]

9
\[
\begin{align*}
\frac{e_1 \text{ val } e_2 \text{ val}}{\langle e_1, e_2 \rangle \text{ val}} & \quad \text{val/\text{pair}} \\
\frac{e_1 \mapsto e'_1}{\langle e_1, e_2 \rangle \mapsto \langle e'_1, e_2 \rangle} & \quad \text{step/\text{pair}}_1 \\
\frac{e_1 \text{ val } e_2 \mapsto e'_2}{\langle e_1, e_2 \rangle \mapsto \langle e_1, e'_2 \rangle} & \quad \text{step/\text{pair}}_2 \\
\frac{e_0 \mapsto e'_0}{\text{case } e_0 (\langle x_1, x_2 \rangle \Rightarrow e_3) \mapsto \text{case } e'_0 (\langle x_1, x_2 \rangle \Rightarrow e_3)} & \quad \text{step/case/\text{pair}}_0 \\
\frac{v_1 \text{ val } v_2 \text{ val}}{\text{case } \langle v_1, v_2 \rangle (\langle x_1, x_2 \rangle \Rightarrow e_3) \mapsto [v_1/x_1][v_2/x_2] e_3} & \quad \text{step/case/\text{pair}}
\end{align*}
\]

**Unit.**

\[
\frac{\Gamma \vdash \langle \rangle : \text{unit}}{\langle \rangle \text{ val/\text{unit}}} \\
\frac{\Gamma \vdash e_0 : 1 \quad \Gamma \vdash e' : \tau}{\Gamma \vdash \text{case } e_0 (\langle \rangle \Rightarrow e') : \tau} & \quad \text{case/\text{unit}}
\]

\[
\frac{\langle \rangle \text{ val/\text{unit}}} {e_0 \mapsto e'_0} \\
\frac{\text{case } e_0 (\langle \rangle \Rightarrow e_1) \mapsto \text{case } e'_0 (\langle \rangle \Rightarrow e_1)} {\text{step/case/\text{unit}}_0}
\]

\[
\frac{\text{case } \langle \rangle (\langle \rangle \Rightarrow e_1) \mapsto e_1} {\text{step/case/\text{unit}}}
\]

**Sums.**

\[
\begin{align*}
\frac{j \in I \quad \Gamma \vdash e : \tau_j}{\Gamma \vdash j \cdot e : \sum_{i \in I} (i : \tau_i)} & \quad \text{sum} \\
\frac{\Gamma \vdash e_0 : \sum_{i \in I} (i : \tau_i) \quad \Gamma, x_i : \tau_i \vdash e_i : \tau \quad \text{for all } i \in I}{\Gamma \vdash \text{case } e_0 (i \cdot x_i \Rightarrow e_i)_{i \in I} : \tau} & \quad \text{case/\text{sum}}
\end{align*}
\]

\[
\frac{e \text{ val}}{j \cdot e \text{ val}} & \quad \text{val/\text{sum}}
\]

\[
\frac{e \mapsto e'}{j \cdot e \mapsto j \cdot e'} & \quad \text{step/\text{sum}}
\]

\[
\frac{e_0 \mapsto e'_0}{\text{case } e_0 (i \cdot x_i \Rightarrow e_i)_{i \in I} \mapsto \text{case } e'_0 (i \cdot x_i \Rightarrow e_i)_{i \in I}} & \quad \text{step/case/\text{sum}}_0
\]

\[
\frac{v \text{ val}}{\text{case } (j \cdot v) (i \cdot x_i \Rightarrow e_i)_{i \in I} \mapsto [v/x_j] e_j} & \quad \text{step/case/\text{sum}}
\]
Recursive Types.

\[
\begin{align*}
\Gamma & \vdash e : [\rho_\alpha. \tau/\alpha] \tau \\
\Gamma & \vdash \text{fold } e : \rho_\alpha. \tau \\
\Gamma & \vdash \text{unfold } e : [\rho_\alpha. \tau/\alpha] \tau \\
\end{align*}
\]

\[
\begin{array}{c}
ed \text{ val} \\
\text{fold } e \text{ val} \\
\hline
e \mapsto e' \\
fold e \mapsto fold e' \\
\hline
e \mapsto e' \\
\text{unfold } e \mapsto \text{unfold } e' \\
\hline
v \text{ val} \\
\text{unfold } (\text{fold } v) \mapsto v \\
\end{array}
\]

Fixed Point Expressions.

\[
\begin{align*}
\Gamma, f : \tau & \vdash e : \tau \\
\Gamma & \vdash \text{fix } f. e : \tau \\
\hline
\text{fix } f. e \mapsto [\text{fix } f. e/f] e \\
\end{align*}
\]