Midterm Exam

15-814 Types and Programming Languages
Frank Pfenning

October 12, 2021

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 8–11 there is an appendix with the syntax, statics, and dynamics for our call-by-value language.

	λ-Calculus	Polymorphism	Parallel Pairs	Corecursive Types	
	Prob 1	Prob 2	Prob 3	Prob 4	Total
Score	**25**	**35**	**40**	**50**	**150**
Max	25	35	40	50	150

1 λ-Calculus (25 points)

Consider the following combinators as defined in the λ-calculus (remembering the convention that $a\,b\,c$ stands for $(a\,b)\,c$):

$$\begin{array}{lcl} I & = & \lambda x.\, x \\ K & = & \lambda x.\, \lambda y.\, x \\ K^* & = & \lambda x.\, \lambda y.\, y \\ C & = & \lambda x.\, \lambda y.\, \lambda z.\, x\, z\, y \\ B & = & \lambda x.\, \lambda y.\, \lambda z.\, x\, (y\, z) \\ W & = & \lambda x.\, \lambda y.\, x\, y\, y \end{array}$$

Task 1 (10 pts). In each of the following problems you are asked to find a definition of the combinator on the left **only in terms of applications constructed from the given combinators** if one exists. In particular, you are not allowed to use λ-abstractions. We filled in the first column for you as an example.

I	$=$	$K^*\,K$	using only K and K^*
K^*	$=$	$K\,I$	using only I and K
K	$=$	$C\,K^*$	using only K^* and C
K^*	$=$	$C\,K$	using only K and C
I	$=$	$K^*\,K^*$	using only K^* and C
I	$=$	$W\,K$	using only K and W

Task 2 (15 pts). A simple type τ for a λ-calculus expression e is *most general* if any type of e can be obtained by substitution of types for type variables. Complete the following table with **most general types** (we have filled in the first one for you as an example):

I	:	$\alpha \to \alpha$
K	:	$\alpha \to (\beta \to \alpha)$
K^*	:	$\alpha \to (\beta \to \beta)$
C	:	$(\alpha \to \beta \to \gamma) \to (\beta \to \alpha \to \gamma)$
B	:	$(\alpha \to \beta) \to (\gamma \to \alpha) \to (\gamma \to \beta)$
W	:	$(\alpha \to \alpha \to \beta) \to (\alpha \to \beta)$

2 Polymorphism (35 points)

Bit strings can be defined in the polymorphic λ-calculus with the type

$$\mathit{bits} \quad = \quad \forall\alpha.\,(\alpha \to \alpha) \to (\alpha \to \alpha) \to \alpha \to \alpha$$

We use the representation function $\ulcorner x \urcorner$ for bit strings x. For example,

$$\begin{array}{lcl} \mathit{ex} & : & \mathit{bits} \\ \mathit{ex} & = & \Lambda\alpha.\lambda b_0.\,\lambda b_1.\,\lambda e.\,b_0\,(b_1\,(b_1\,(b_0\,(b_1\,e)))) \quad = \quad \ulcorner 01101 \urcorner \end{array}$$

Task 1 (15 pts). Complete the following definitions

$\mathit{b0} \quad : \quad \mathit{bits} \to \mathit{bits}$ where $\mathit{b0}\ \ulcorner x \urcorner = \ulcorner 0x \urcorner$

$\mathit{b0} \quad = \quad \boxed{\lambda x.\,\Lambda\alpha.\,\lambda b_0.\,\lambda b_1.\,\lambda e.\,b_0\ (x\,[\alpha]\,b_0\ b_1\ e)}$

$\mathit{b1} \quad : \quad \mathit{bits} \to \mathit{bits}$ where $\mathit{b1}\ \ulcorner x \urcorner = \ulcorner 1x \urcorner$

$\mathit{b1} \quad = \quad \boxed{\lambda x.\,\Lambda\alpha.\,\lambda b_0.\,\lambda b_1.\,\lambda e.\,b_1\ (x\,[\alpha]\,b_0\ b_1\ e)}$

$\mathit{e} \quad = \quad \mathit{bits}$ where $\mathit{e} = \ulcorner \cdot \urcorner$

$\mathit{e} \quad = \quad \boxed{\Lambda\alpha.\lambda b_0.\,\lambda b_1.\,\lambda e.\,e}$

Task 2 (10 pts). Complete the following definition of the functions that flips every bit in the input. You may use the definitions from Task 1 and make auxiliary definitions if necessary.

$\mathit{flip} \quad : \quad \mathit{bits} \to \mathit{bits}$

$\mathit{flip} \quad = \quad \boxed{\lambda x.\,x\,[\mathit{bits}]\,\mathit{b1}\ \mathit{b0}\ \mathit{e}}$

Task 3 (10 pts). Complete the following definition of the parity function, where *parity* $\ulcorner x \urcorner$ returns $\ulcorner 0 \urcorner$ if x has an even number of 1s and $\ulcorner 1 \urcorner$ if x has an odd number of 1s. You may use the definitions from Tasks 1 and 2 and make auxiliary definitions if necessary.

$\mathit{parity} \quad : \quad \mathit{bits} \to \mathit{bits}$

$\mathit{parity} \quad = \quad \boxed{\lambda x.\,x\,[\mathit{bits}]\,(\lambda y.\,y)\,\mathit{flip}\ (\mathit{b0}\ \mathit{e})}$

3 Parallel Pairs (40 points)

We add a new type constructor of *parallel pairs* $\tau_1 \parallel \tau_2$ to our call-by-value language with the following constructs.

$$\begin{array}{lrcl} \text{Types} & \tau & ::= & \cdots \mid \tau_1 \parallel \tau_2 \\ \text{Expressions} & e & ::= & \cdots \mid \langle\!\langle e_1, e_2 \rangle\!\rangle \mid \mathsf{case}\ e\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow e') \end{array}$$

Parallel pairs are "super-eager" in that they can step both components at the same time whenever possible. For example, writing v for values and $\bot = \mathsf{fix}\ f.\ f$ we would have

$$\begin{array}{lcl} \langle\!\langle e_1, \bot \rangle\!\rangle & & \text{does not have a value} \\ \langle\!\langle \bot, e_2 \rangle\!\rangle & & \text{does not have a value} \\ \langle\!\langle v_1, v_2 \rangle\!\rangle & & \text{is a value} \\ \langle\!\langle (\lambda x\, x)\, v_1, (\lambda x.\, \lambda y.\, x)\, v_2\, v_3 \rangle\!\rangle & \mapsto^2 & \langle\!\langle v_1, v_2 \rangle\!\rangle \end{array}$$

We provide the typing rules and one critical stepping rule.

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle\!\langle e_1, e_2 \rangle\!\rangle : \tau_1 \parallel \tau_2}\ \mathsf{tp/ppair} \qquad \frac{\Gamma \vdash e : \tau_1 \parallel \tau_2 \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e' : \tau'}{\Gamma \vdash \mathsf{case}\ e\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow e') : \tau'}\ \mathsf{tp/casepp}$$

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

Task 1 (20 pts). Complete the set of rules for $e \mapsto e'$ involving the constructors and destructors for type $\tau_1 \parallel \tau_2$. You should ensure that the following theorems continue to hold: **Preservation**, **Progress**, **Finality of Values**, and **Determinacy**. See the Appendix for a statement of these theorems. Please number your rules (1), (2), ... so you can reference them in the answers to the next set of questions if necessary.

$$\frac{e_1 \mapsto e_1' \quad v_2\ \mathit{val}}{\langle\!\langle e_1, v_2 \rangle\!\rangle \mapsto \langle\!\langle e_1', v_2 \rangle\!\rangle}\ \mathsf{step/ppair}_0 \qquad \frac{v_1\ \mathit{val} \quad e_2 \mapsto e_2'}{\langle\!\langle v_1, e_2 \rangle\!\rangle \mapsto \langle\!\langle v_1, e_2' \rangle\!\rangle}\ \mathsf{step/ppair}_1$$

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

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

In the next few tasks we ask you to **add** or **delete exactly one stepping rule** (leaving everything else unchanged) so that one of the four key properties is now violated while the others remain true. Either write out the rule to be added, or indicate the rule to be deleted. When neither is possible,

just write "*impossible*". You do not need to justify this answer further. If needed, refer to the rules in Task 1 by the number you assigned to them.

Task 2 (5 pts). Add or delete one rule such that **preservation** fails, while progress, finality of values, and determinacy continue to hold.

That's impossible. First, we observe that deleting a rule would not disturb preservation.

Second, we cannot add a rule to violate preservation while maintaining progress, finality of values, and determinacy. By progress, every expression $e : \tau$ is either a value or reduces. Adding a rule to reduce a value violates finality of values. Adding a rule to reduce an expression to an alternate result would break determinacy.

Task 3 (5 pts). Add or delete one rule such that **progress** fails, while preservation, determinacy, and finality of values continue to hold.

We remove $\mathsf{step}/\mathsf{ppair}_2$ (but actually removing any stepping rule works).

Task 4 (5 pts). Add or delete one rule such that **determinacy** fails, while preservation, progress, and finality of values continue to hold.

We add a rule

$$\frac{e_1 \mapsto e_1'}{\langle\!\langle e_1, e_2 \rangle\!\rangle \mapsto \langle\!\langle e_1', e_2 \rangle\!\rangle}$$

Task 5 (5 pts). Add or delete one rule such that **finality of values** fails, while preservation, progress, and determinacy continue to hold.

This is impossible. If we add a rule to reduce a value $\langle\!\langle v_1, v_2 \rangle\!\rangle$ then determinacy will fail for $\mathsf{case}\ \langle\!\langle v_1, v_2 \rangle\!\rangle\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow e)$.

4 Corecursive Types (50 points)

A corecursive type $\nu\alpha.\,\tau$ is similar to a recursive type $\mu\alpha.\,\tau$ except that it is lazy. We have a constructor roll and a destructor unroll, corresponding to fold and unfold, respectively. We specify:

$$\frac{}{\mathsf{roll}\ e\ \textit{value}}\ \mathsf{val/roll} \qquad \frac{}{\mathsf{unroll}\ (\mathsf{roll}\ e) \mapsto e}\ \mathsf{step/unroll/roll} \qquad \frac{e \mapsto e'}{\mathsf{unroll}\ e \mapsto \mathsf{unroll}\ e'}\ \mathsf{step/unroll}_1$$

$$\frac{\Gamma \vdash e : [\nu\alpha.\,\tau/\alpha]\tau}{\Gamma \vdash \mathsf{roll}\ e : \nu\alpha.\,\tau}\ \mathsf{tp/roll} \qquad \frac{\Gamma \vdash e : \nu\alpha.\,\tau}{\Gamma \vdash \mathsf{unroll}\ e : [\nu\alpha.\,\tau/\alpha]\tau}\ \mathsf{tp/unroll}$$

Task 1 (5 pts). The preservation theorem states that if $\cdot \vdash e : \tau$ and $e \mapsto e'$ then $\cdot \vdash e' : \tau$. The proof of this theorem is by induction. State by which form of induction and on which expression or judgment.

By rule induction on the derivation of $e \mapsto e'$

Task 2 (15 pts). Provide **all new cases** in the proof of the preservation theorem pertaining to corecursive types.

Case:

$$\frac{}{\mathsf{unroll}\ (\mathsf{roll}\ e_1) \mapsto e_1}\ \mathsf{step/unroll/roll}$$

where $e = \mathsf{unroll}\ (\mathsf{roll}\ e_1)$ and $e' = e_1$.

$\cdot \vdash e : \tau$	Given
$\cdot \vdash \mathsf{unroll}\ (\mathsf{roll}\ e_1) : \tau$	By equality
$\cdot \vdash \mathsf{roll}\ e_1 : \nu\alpha.\,\sigma$ for some σ with $\tau = [\nu\alpha.\,\sigma/\alpha]\sigma$	By inversion (rule tp/unroll)
$\cdot \vdash e_1 : [\nu\alpha.\,\sigma/\alpha]\sigma$	By inversion (rule tp/roll)
$\cdot \vdash e' : \tau$	By equality

Case:

$$\frac{e_1 \mapsto e_1'}{\mathsf{unroll}\ e_1 \mapsto \mathsf{unroll}\ e_1'}\ \mathsf{step/unroll}_1$$

where $e = \mathsf{unroll}\ e_1$ and $e' = \mathsf{unroll}\ e_1'$.

$\cdot \vdash e : \tau$	Given
$\cdot \vdash \mathsf{unroll}\ e_1 : \tau$	By equality
$\cdot \vdash e_1 : \nu\alpha.\,\sigma$ for some σ with $\tau = [\nu\alpha.\,\sigma/\alpha]\sigma$	By inversion (rule tp/unroll)
$\cdot \vdash e_1' : \nu\alpha.\,\sigma$	By induction hypothesis
$\cdot \vdash \mathsf{unroll}\ e_1' : [\nu\alpha.\,\sigma/\alpha]\sigma$	By rule tp/unroll
$\cdot \vdash e' : \tau$	By equality

For reference, we fix the usual recursive type of natural numbers.

$$\begin{array}{lcl} \mathit{nat} & = & \mu\alpha.\,(\mathbf{zero} : 1) + (\mathbf{succ} : \alpha) \\ \mathit{zero} & : & \mathit{nat} \\ \mathit{zero} & = & \mathsf{fold}\ (\mathbf{zero} \cdot \langle\,\rangle) \\ \\ \mathit{succ} & : & \mathit{nat} \to \mathit{nat} \\ \mathit{succ} & = & \lambda n.\,\mathsf{fold}\ (\mathbf{succ} \cdot n) \end{array}$$

Now consider the type *stream* of potentially infinite streams of natural numbers. As an example, we provide a definition of a potentially infinite stream of zeros (written informally as $0, 0, 0, \ldots$).

$$\begin{array}{lcl} \mathit{stream} & = & \nu\alpha.\,\mathit{nat} \times \alpha \\ \\ \mathit{zeros} & : & \mathit{stream} \\ \mathit{zeros} & = & \mathsf{fix}\, s.\,\mathsf{roll}\,\langle \mathit{zero}, s\rangle \end{array}$$

Task 2 (5 pts). Show the value v such that $\mathit{zeros} \mapsto^* v$ or indicate it doesn't terminate. You may freely use the definitions above.

$$v = \mathsf{roll}\,\langle \mathit{zero}, \mathit{zeros}\rangle$$

Task 3 (10 pts). Complete the definition of *count*, where *count* k should produce the stream $k, k+1, k+2, \ldots$. You may freely use the definitions above.

$$\mathit{count} \quad : \quad \mathit{nat} \to \mathit{stream}$$

$$\mathit{count} = \mathsf{fix}\, f.\,\lambda k.\,\mathsf{roll}\,\langle k, f\ (\mathit{succ}\ k)\rangle$$

Task 4 (15 pts). Complete the definition of *nth*, where *nth* n s returns k_n in the stream $s = k_0, k_1, k_2, \ldots$. You may freely use the definitions above.

$$\mathit{nth} \quad : \quad \mathit{nat} \to \mathit{stream} \to \mathit{nat}$$

$$\mathit{nth} = \mathsf{fix}\, f.\,\lambda n.\,\lambda s.\,\mathsf{case}\ (\mathsf{unroll}\ s)\ (\langle k, s'\rangle \Rightarrow \mathsf{case}\ (\mathsf{unfold}\ n)\ (\mathbf{zero} \cdot \_ \Rightarrow k \mid \mathbf{succ} \cdot n' \Rightarrow f\ n'\ s'))$$

Appendix: Language Reference

Abstract Syntax

Types	τ	$::=$	$\tau_1 \to \tau_2 \mid \forall\alpha.\,\tau \mid \alpha \mid \tau_1 \times \tau_2 \mid 1 \mid \sum_{i\in I}(i : \tau_i) \mid \mu\alpha.\,\tau$	
Terms	e	$::=$	$x \mid \lambda x.\,e \mid e_1\,e_2$	$(\to)$
		$\mid$	$\Lambda\alpha.\,e \mid e\,[\tau]$	$(\forall)$
		$\mid$	$\langle e_1, e_2\rangle \mid \mathsf{case}\ e\ (\langle x_1, x_2\rangle \Rightarrow e')$	$(\times)$
		$\mid$	$\langle\,\rangle \mid \mathsf{case}\ e\ (\langle\,\rangle \Rightarrow e')$	(1)
		$\mid$	$k \cdot e \mid \mathsf{case}\ e\ (i \cdot x_i \Rightarrow e_i)_{i\in I}$	$(+)$
		$\mid$	$\mathsf{case}\ e\ (\,)$	(0)
		$\mid$	$\mathsf{fold}\ e \mid \mathsf{unfold}\ e$	(μ)
		$\mid$	$\mathsf{fix}\, f.\, e \mid f$	
Contexts	Γ	$::=$	$\cdot \mid \Gamma, \alpha\ \mathit{type} \mid \Gamma, x : \tau$	(all variables distinct)

Judgments

$\Gamma\ \mathit{ctx}$	Γ is a valid context	
$\Gamma \vdash \tau\ \mathit{type}$	τ is a valid type	presupposes $\Gamma\ \mathit{ctx}$
$\Gamma \vdash e : \tau$	expression e has type τ	presupposes $\Gamma\ \mathit{ctx}$, ensures $\Gamma \vdash \tau\ \mathit{type}$
$e\ \mathit{value}$	expression e is a value	presupposes $\cdot \vdash e : \tau$ for some τ
$e \mapsto e'$	expression e steps to e'	presupposes $\cdot \vdash e : \tau$ for some τ

Theorems

Preservation. If $\cdot \vdash e : \tau$ and $e \mapsto e'$ then $\cdot \vdash e' : \tau$.

Progress. For every expression $\cdot \vdash e : \tau$ either $e \mapsto e'$ for some e' or e *value*.

Finality of Values. If $\cdot \vdash e : \tau$ and e *value* then there is no e' with $e \mapsto e'$.

Determinacy. If $\cdot \vdash e : \tau$ and $e \mapsto e_1$ and $e \mapsto e_2$ then $e_1 = e_2$.

Canonical Forms. Assume $\cdot \vdash e : \tau$ and e *value*.

(i) If $\tau = \tau_1 \to \tau_2$ then $e = \lambda x.\, e_2$ for some e_2

(ii) If $\tau = \forall\alpha.\,\tau'$ then $e = \Lambda\alpha.\, e'$ for some e'

(iii) If $\tau = \tau_1 \times \tau_2$ then $e = \langle e_1, e_2\rangle$ for some e_1 *value* and e_2 *value*

(iv) If $\tau = 1$ then $e = \langle\,\rangle$

(v) If $\tau = \sum_{i\in I}(i : \tau_i)$ then $e = k \cdot e_k$ for some $k \in I$ and e_k *value*.

(vi) If $\tau = \mu\alpha.\,\tau'$ then $e = \mathsf{fold}\ e'$ for some e' *value*

Contexts Γ

$$\frac{}{(\cdot)\ \mathit{ctx}}\ \mathsf{ctx/emp} \qquad \frac{\Gamma\ \mathit{ctx}}{(\Gamma, \alpha\ \mathit{type})\ \mathit{ctx}}\ \mathsf{ctx/tpvar} \qquad \frac{\Gamma\ \mathit{ctx} \quad \Gamma \vdash \tau\ \mathit{type}}{(\Gamma, x : \tau)\ \mathit{ctx}}\ \mathsf{ctx/var}$$

Functions $\tau_1 \to \tau_2$

$$\frac{\Gamma \vdash \tau_1\ \mathit{type} \quad \Gamma \vdash \tau_2\ \mathit{type}}{\Gamma \vdash \tau_1 \to \tau_2\ \mathit{type}}\ \mathsf{tp/arrow}$$

$$\frac{\Gamma \vdash \tau_1\ \mathit{type} \quad \Gamma, x_1 : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda x_1{:}\tau_1.\, e_2 : \tau_1 \to \tau_2}\ \mathsf{tp/lam} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \mathsf{tp/var} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau_1}\ \mathsf{tp/app}$$

$$\frac{}{\lambda x.\, e\ \mathit{value}}\ \mathsf{val/lam} \qquad \frac{e_2\ \mathit{value}}{(\lambda x.\, e_1)\, e_2 \mapsto [e_2/x]e_1}\ \mathsf{step/app/lam}$$

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

Polymorphic Types $\forall \alpha.\, \tau$

$$\frac{\alpha\ \mathit{type} \in \Gamma}{\Gamma \vdash \alpha\ \mathit{type}}\ \mathsf{tp/tpvar} \qquad \frac{\Gamma, \alpha\ \mathit{type} \vdash \tau\ \mathit{type}}{\Gamma \vdash \forall \alpha.\, \tau\ \mathit{type}}\ \mathsf{tp/forall}$$

$$\frac{\Gamma, \alpha\ \mathit{type} \vdash e : \tau}{\Gamma \vdash \Lambda \alpha.\, e : \forall \alpha.\, \tau}\ \mathsf{tp/tplam} \qquad \frac{\Gamma \vdash e : \forall \alpha.\, \tau \quad \Gamma \vdash \sigma\ \mathit{type}}{\Gamma \vdash e\,[\sigma] : [\sigma/\alpha]\tau}\ \mathsf{tp/tpapp}$$

$$\frac{}{\Lambda \alpha.\, e\ \mathit{value}}\ \mathsf{val/tplam} \qquad \frac{}{(\Lambda \alpha.\, e)\,[\tau] \mapsto [\tau/\alpha]e}\ \mathsf{step/tpapp/tplam} \qquad \frac{e \mapsto e'}{e\,[\tau] \mapsto e'\,[\tau]}\ \mathsf{step/tpapp}$$

Pairs $\tau_1 \times \tau_2$

$$\frac{\Gamma \vdash \tau_1 \; \textit{type} \quad \Gamma \vdash \tau_2 \; \textit{type}}{\Gamma \vdash \tau_1 \times \tau_2 \; \textit{type}} \; \mathsf{tp/prod}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \; \mathsf{tp/pair} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2 \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e' : \tau'}{\Gamma \vdash \mathsf{case}\ e\ (\langle x_1, x_2 \rangle \Rightarrow e') : \tau'} \; \mathsf{tp/casep}$$

$$\frac{e_1 \; \textit{value} \quad e_2 \; \textit{value}}{\langle e_1, e_2 \rangle \; \textit{value}} \; \mathsf{val/pair} \qquad \frac{v_1 \; \textit{value} \quad v_2 \; \textit{value}}{\mathsf{case}\ \langle v_1, v_2 \rangle\ (\langle x_1, x_2 \rangle \Rightarrow e_3) \mapsto [v_1/x_1][v_2/x_2]e_3} \; \mathsf{step/casep/pair}$$

$$\frac{e_1 \mapsto e_1'}{\langle e_1, e_2 \rangle \mapsto \langle e_1', e_2 \rangle} \; \mathsf{step/pair}_1 \qquad \frac{v_1 \; \textit{value} \quad e_2 \mapsto e_2'}{\langle v_1, e_2 \rangle \mapsto \langle v_1, e_2' \rangle} \; \mathsf{step/pair}_2$$

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

Unit 1

$$\frac{}{\Gamma \vdash 1 \; \textit{type}} \; \mathsf{tp/one} \qquad \frac{}{\Gamma \vdash \langle\,\rangle : 1} \; \mathsf{tp/unit} \qquad \frac{\Gamma \vdash e : 1 \quad \Gamma \vdash e' : \tau'}{\Gamma \vdash \mathsf{case}\ e\ (\langle\,\rangle \Rightarrow e') : \tau'} \; \mathsf{tp/caseu}$$

$$\frac{}{\langle\,\rangle \; \textit{value}} \; \mathsf{val/unit} \qquad \frac{}{\mathsf{case}\ \langle\,\rangle\ (\langle\,\rangle \Rightarrow e) \mapsto e} \; \mathsf{step/caseu/unit}$$

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

Sums $\sum_{i \in I}(i : \tau_i)$

$$\frac{\Gamma \vdash \tau_i \; \textit{type} \quad (\text{for all } i \in I)}{\Gamma \vdash \sum_{i \in I}(i : \tau_i) \; \textit{type}} \; \mathsf{tp/sum}$$

$$\frac{(k \in I) \quad \Gamma \vdash e_k : \tau_k \quad \Gamma \vdash \sum_{i \in I}(i : \tau_i) \; \textit{type}}{\Gamma \vdash k \cdot e_k : \sum_{i \in I}(i : \tau_i)} \; \mathsf{tp/tag}$$

$$\frac{\Gamma \vdash e : \sum_{i \in I}(i : \tau_i) \quad \Gamma, x_i : \tau_i \vdash e_i : \sigma \quad (\text{for all } i \in I)}{\Gamma \vdash \mathsf{case}\ e\ (i \cdot x_i \Rightarrow e_i)_{i \in I} : \sigma} \; \mathsf{tp/cases}$$

$$\frac{e \; \textit{value}}{k \cdot e \; \textit{value}} \; \mathsf{val/tag} \qquad \frac{v_k \; \textit{value}}{\mathsf{case}\ k \cdot v_k\ (i \cdot x_i \Rightarrow e_i)_{i \in I} \mapsto [v_k/x_k]e_k} \; \mathsf{step/cases/tag}$$

$$\frac{e_1 \mapsto e_1'}{k \cdot e_1 \mapsto k \cdot e_1'} \; \mathsf{step/tag} \qquad \frac{e_0 \mapsto e_0'}{\mathsf{case}\ e_0\ (i \cdot x_i \Rightarrow e_i)_{i \in I} \mapsto \mathsf{case}\ e_0'\ (i \cdot x_i \Rightarrow e_i)_{i \in I}} \; \mathsf{step/cases}_0$$

Recursive Types $\mu\alpha.\,\tau$

$$\frac{\Gamma, \alpha \; \textit{type} \vdash \tau \; \textit{type}}{\Gamma \vdash \mu\alpha.\,\tau \; \textit{type}} \; \mathsf{tp/mu}$$

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

$$\frac{e \; \textit{value}}{\mathsf{fold}\ e \; \textit{value}} \; \mathsf{val/fold} \qquad \frac{v \; \textit{value}}{\mathsf{unfold}\ (\mathsf{fold}\ v) \mapsto v} \; \mathsf{step/unfold/fold}$$

$$\frac{e \mapsto e'}{\mathsf{fold}\ e \mapsto \mathsf{fold}\ e'} \; \mathsf{step/fold} \qquad \frac{e \mapsto e'}{\mathsf{unfold}\ e \mapsto \mathsf{unfold}\ e'} \; \mathsf{step/unfold}_0$$

Recursion

$$\frac{\Gamma, f : \tau \vdash e : \tau}{\Gamma \vdash \mathsf{fix}\ f{:}\tau.\, e : \tau} \; \mathsf{tp/fix} \qquad \frac{}{\mathsf{fix}\ f.\, e \mapsto [\mathsf{fix}\ f.\, e/f]e} \; \mathsf{step/fix}$$