Final Exam

15-814 Types and Programming Languages
Frank Pfenning

December 13, 2019

Name: Andrew ID:

Instructions

- This exam is closed-book, closed-notes.
- You have 180 minutes to complete the exam.
- There are 6 problems.
- For reference, on pages 12–17 there is an appendix with sections on the syntax, statics, and dynamics.

	Propositions as Types	Bisimulation	Isomorphisms	Parametricity	Sequentiality	Polymorphic Concurrency	
	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Prob 6	Total
Score							
Max	45	50	40	50	30	35	250

1 Propositions as Types (45 pts)

Consider the following natural deduction (where we have abbreviated A *true* by just A):

$$\dfrac{\dfrac{\dfrac{}{(A \supset B) \supset C}\, x \quad \dfrac{\dfrac{}{B}\, y}{A \supset B}\, \supset I^z}{C}\, \supset E}{\dfrac{C}{B \supset C}\, \supset I^y}{((A \supset B) \supset C) \supset (B \supset C)}\, \supset I^x$$

Task 1 (5 pts). Write out the type corresponding to the proposition whose truth is established by this deduction, using α for A, β for B, and γ for C.

Task 2 (5 pts). Write out the functional expression that can be extracted from the given natural deduction. Your variable names should correspond to the variable names introduced in the deduction.

Task 3 (5 pts). Is this expression linear? Indicate 'yes' or 'no', and in case it is not, list the variables that are not linear.

Task 4 (10 pts). Write a functional expression of type

$$((\alpha + \gamma) \,\&\, (\beta \to \gamma)) \to ((\alpha \to \beta) \to \gamma)$$

where α, β, and γ are type variables. You do not need to write a typing derivation.

Task 5 (5 pts). Is this expression linear? Indicate 'yes' or 'no' and in case it is not, list the variables that are not linear.

Task 6 (5 pts). Show the proposition of intuitionistic logic that corresponds to the type above, using A for α, B for β, and C for γ.

Task 7 (10 pts). Write a natural deduction that corresponds to your expression. The name of the variables introduced in $\supset I$ and $\vee E$ (in case you use them) should match the names of the corresponding variables in your functional expression. For reference, the inference rules of natural deduction are shown in the Appendix on page 12.

2 Bisimulation (50 pts)

In implementations of functional languages types are generally erased from expressions before they are evaluated. This can be justified if we agree that types cannot be observed in the outcome of computations.

Task 1 (5 pts). In our language, where do types occur in *values* in positions where we would expect to observe them? Give all such constructs.

Task 2 (15 pts). The intuition behind the erasure is that we replace type abstraction by a suitable λ-abstraction and every type τ by the unit element. To formalize this, we define a translation $(e)^*$ on expressions such that e is bisimilar to $(e)^*$. We show the cases for functions and one case regarding the quantifiers. Complete the definition.

$(x)^*$	$=$	x
$(\lambda x.\, e)^*$	$=$	$\lambda x.\, e^*$
$(e_1\, e_2)^*$	$=$	$e_1^*\, e_2^*$
$(\Lambda\alpha.\, e)^*$	$=$	
$(e\,[\tau])^*$	$=$	$e^*\,\langle\,\rangle$
$(\langle \tau, e\rangle)^*$	$=$	
$(\mathsf{case}\ e\ (\langle \alpha, x\rangle \Rightarrow e'))^*$	$=$	

Task 3 (10 pts). The translation $(e)^*$ is a rare example where the result is not necessarily well-typed even if e is. Give an example of a closed e that has some type τ (that is, $\cdot \vdash e : \tau$) but the translation has no type (that is, there is no type σ such that $\cdot \vdash e^* : \sigma$). You do not need to show a typing derivation for e, nor prove that there is none for $(e)^*$.

Task 4 (20 pts). Nevertheless, the translation defines a *bisimulation* R if we define $e \ R \ e^*$. This bisimulation is *strong* in the sense that a single step in e is simulated by a single step in e^* and vice versa. We consider one case in one direction of this proof. Complete the text.

> We prove that if $e \ R \ e^*$ and $e \mapsto e'$ then $e^* \mapsto e_0$ for some e_0 with $e' \ R \ e_0$. The proof proceeds by [ ].

Consider all the relevant cases in the proof when $e = e_1 \, [\tau]$ for some e_1 and τ. One of the cases will require a simple lemma regarding the behavior of $(-)^*$ which you should state explicitly at the end.

3 Isomorphisms (40 pts)

Task 1 (10 pts). Consider the following proposed isomorphism (where α does not occur in τ).

$$\tau \quad \stackrel{?}{\cong} \quad \forall\alpha.\,((\tau \to \alpha) \to \alpha)$$

Define functions *Forth* and *Back* between these types.

Forth	:	
Forth	=	
Back	:	
Back	=	

Task 2 (10 pts). Prove that $\mathit{Back} \circ \mathit{Forth} = \lambda x.\,x$ for your definitions of *Back* and *Forth*.

Task 3 (20 pts). Provide functions *Forth* and *Back* for the following proposed isomorphism (where, again α does not appear in τ).

$$\exists\beta.\,\tau \quad \stackrel{?}{\cong} \quad \forall\alpha.\,((\forall\beta.\,\tau \to \alpha) \to \alpha)$$

You do not need to prove that they form an isomorphism.

Forth	=	
Back	=	

4 Parametricity (50 pts)

Recall the type of natural numbers in binary representation

$$\mathit{bin} \cong (\texttt{B0} : \mathit{bin}) + (\texttt{B1} : \mathit{bin}) + (\texttt{E} : 1)$$

We consider a (drastically simplified) library `CRYPT` to encrypt and decrypt binary numbers.

```
CRYPT = {
  type cipher
  encrypt : bin -> cipher
  decrypt : cipher -> bin
}
```

A desired property is that $\mathit{decrypt}\ (\mathit{encrypt}\ v) \mapsto^* v$ for all values $v : \mathit{bin}$.

Task 1 (5 pts). Express this library signature as an existential type *Crypt*.

Task 2 (5 pts). Provide a reference implementation $\mathit{Id} : \mathit{Crypt}$ in which the "encrypted" form of a binary number x is just x itself.

Task 3 (10 pts). Provide an alternative implementation $\mathit{Flip} : \mathit{Crypt}$ where each bit in the encrypted value is the opposite of the bit in original value. For this purpose, you should first define a function $\mathit{flip} : \mathit{bin} \to \mathit{bin}$. Your function(s) should be precise regarding the uses of *fold* and *unfold*.

Task 4 (5 pts). State the definition of $v \sim v' \in [\exists \alpha.\, \tau]$

Task 5 (5 pts). Complete the statement of the following lemma:

Lemma. For any purely positive type τ^+ we have $v \sim v' \in [\tau^+]$ iff $\boxed{}$.

Task 6 (20 pts). Prove that *Id* $\sim$ *Flip* $\in$ [*Crypt*]. You will need to use the definition from Task 4 and the property from Task 5. If you need some properties of the *flip* function, please state them without proof.

5 Sequentiality (30 pts)

We have translated expressions e that are evaluated sequentially to processes P that compute concurrently. To simulate *sequential* evaluation we follow a particular *schedule* that completes computation of P before starting the computation of Q for a spawn $x \leftarrow P \ ; \ Q$. In this problem we formalize this computation strategy. A summary of the process language can be found in the Appendix on page 17.

We add a new kind of process expression called *sequential composition*

$$\text{Processes} \quad P \quad ::= \quad \ldots \mid x \Leftarrow P \ ; \ Q$$

and also a new form of semantic object

$$\mathsf{cont}\ c\ d\ Q$$

which waits for the cell c to be filled and then continues with the computation of process Q with destination d. One critical rule connecting the two is

$$\mathsf{proc}\ d\ (x \Leftarrow P \ ; \ Q) \mapsto \mathsf{proc}\ c\ ([c/x]P), \mathsf{cell}\ c\ \_, \mathsf{cont}\ c\ d\ ([c/x]Q) \qquad (c \text{ fresh})$$

Task 1 (5 pts). Write a rule to "wake up" a continuation object $\mathsf{cont}\ c\ d\ Q$ when the cell it is waiting on has been filled.

Task 2 (10 pts). Using the new process construct, rewrite the translations for *eager pairs* so they can only be evaluated *sequentially* in the process language, just as they are in the functional expression language. Make sure to annotate each destination that is written to (with $()^W$) or read from (with $()^R$).

$[\![\langle e_1, e_2 \rangle]\!]\ d =$

$[\![\mathsf{case}\ e\ (\langle x_1, x_2 \rangle \Rightarrow e')]\!]\ d =$

Task 3 (15 pts). Write the translations for *parallel pairs*, where $\langle\!\langle e_1, e_2 \rangle\!\rangle$ evaluates e_1 and e_2 in parallel but waits for *both* to finish before continuing with other computation. Specifically, $\langle\!\langle v_1, v_2 \rangle\!\rangle$ is a value only if both v_1 and v_2 are. Note that this behavior is different from concurrent pairs in the process language in which computation referencing the components of a concurrent pair can continue without either of them finishing. For example, we expect the following properties (where $\bot$ with $\bot \mapsto \bot$ is a convenient nonterminating expression):

$[\![\mathsf{case}\ \langle\!\langle v_1, v_2 \rangle\!\rangle\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow \langle\!\langle x_2, x_1 \rangle\!\rangle)]\!]\, d$	writes a pair $\langle d_2, d_1 \rangle$ to d
$[\![\mathsf{case}\ \langle\!\langle e_1, \bot \rangle\!\rangle\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow x_1)]\!]\, d$	never writes to d
$[\![\mathsf{case}\ \langle\!\langle \bot, e_2 \rangle\!\rangle\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow x_2)]\!]\, d$	never writes to d

Complete the following definition, again annotating each destination that is written to (with $()^W$) or read from (with $()^R$), for clarity. You should exploit sequential composition, rather than introducing any new values or continuations in the process language

$[\![\langle\!\langle e_1, e_2 \rangle\!\rangle]\!]\, d =$

$[\![\mathsf{case}\ e\ (\langle\!\langle x_1, x_2 \rangle\!\rangle \Rightarrow e')]\!]\, d =$

6 Polymorphic Concurrency (35 pts)

In this problem we explore an extension of our process language with universal and existential types.

Task 1 (5 pts). State briefly what we mean when we say that two types are *dual* in the process language.

Task 2 (5 pts). Fill in the following table with examples of dual types in our language already.

positive	negative
	$\tau \to \sigma$
$\tau + \sigma$	
$\exists\alpha.\,\tau$	$\forall\alpha.\,\tau$

Task 3 (25 pts). We add new *values* $\langle \tau, c \rangle$ and new *continuations* $(\langle \alpha, x \rangle \Rightarrow P)$ to the process language so we can define the translations of expressions for universal and existential types. Complete the following definitions. Take care to annotate addresses we write to or read from with $()^W$ and $()^R$ respectively so that the dynamics of the result is as clear as possible.

$$\langle \tau, d \rangle \triangleright (\langle \alpha, x \rangle \Rightarrow P) \quad = \quad \boxed{}$$

$$[\![\langle \tau, e \rangle]\!]\, d \quad = \quad \boxed{}$$

$$[\![\mathsf{case}\ e\ (\langle \alpha, x \rangle \Rightarrow e')]\!]\, d \quad = \quad \boxed{}$$

$$[\![\Lambda\alpha.\, e]\!]\, d \quad = \quad \boxed{}$$

$$[\![e\,[\tau]]\!]\, d \quad = \quad \boxed{}$$

Appendix: Natural Deduction

Propositions

$$\text{Propositions} \quad A \quad ::= \quad A \wedge B \mid \top \mid A \supset B \mid A \vee B \mid \bot$$

Rules of Natural Deduction

Introduction Rules	**Elimination Rules**
$\dfrac{A\ true \quad B\ true}{A \wedge B\ true}\ \wedge I$	$\dfrac{A \wedge B\ true}{A\ true}\ \wedge E_1 \quad \dfrac{A \wedge B\ true}{B\ true}\ \wedge E_2$
$\dfrac{}{\top\ true}\ \top I$	*no* $\top E$ *rule*
$\dfrac{\begin{array}{c}\overline{A\ true}\ x \\ \vdots \\ B\ true\end{array}}{A \supset B\ true}\ \supset I^x$	$\dfrac{A \supset B\ true \quad A\ true}{B\ true}\ \supset E$
$\dfrac{A\ true}{A \vee B\ true}\ \vee I_1 \quad \dfrac{B\ true}{A \vee B\ true}\ \vee I_2$	$\dfrac{A \vee B\ true \quad \begin{array}{c}\overline{A\ true}\ u \\ \vdots \\ C\ true\end{array} \quad \begin{array}{c}\overline{B\ true}\ w \\ \vdots \\ C\ true\end{array}}{C\ true}\ \vee E^{u,w}$
no $\bot I$ *rule*	$\dfrac{\bot\ true}{C\ true}\ \bot E$

Appendix: Expression Language Reference

Language

Types	τ	::=	$\alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2 \mid 1 \mid \sum_{i \in I}(i : \tau_i) \mid \&_{i \in I}(i : \tau_i) \mid \rho\alpha.\,\tau \mid \forall\alpha.\,\tau \mid \exists\alpha.\,\tau$	
Expressions	e	::=	x	(variables)
		$\mid$	$\lambda x.\,e \mid e_1\, e_2$	$(\to)$
		$\mid$	$\langle e_1, e_2\rangle \mid \mathsf{case}\ e\ (\langle x_1, x_2\rangle \Rightarrow e')$	$(\times)$
		$\mid$	$\langle\,\rangle \mid \mathsf{case}\ e\ (\langle\,\rangle \Rightarrow e')$	(1)
		$\mid$	$j \cdot e \mid \mathsf{case}\ e\ (i \cdot x_i \Rightarrow e_i)_{i \in I}$	$(\sum)$
		$\mid$	$\{\!|i \Rightarrow e_i|\!\}_{i \in I} \mid e \cdot j$	$(\&)$
		$\mid$	$\mathsf{fold}\ e \mid \mathsf{unfold}\ e$	(ρ)
		$\mid$	$f \mid \mathsf{fix}\ f.\,e$	(recursion)
		$\mid$	$\Lambda\alpha.\,e \mid e\,[\tau]$	$(\forall)$
		$\mid$	$\langle \tau, e\rangle \mid \mathsf{case}\ e\ (\langle \alpha, x\rangle \Rightarrow e')$	$(\exists)$
Contexts	Γ	::=	$x_1 : \tau_1, \ldots, x_n : \tau_n$ (all x_i distinct)	
Type Contexts	Δ	::=	$\alpha_1\ \mathit{tp}, \ldots, \alpha_n\ \mathit{tp}$ (all α_i distinct)	

Statics and Dynamics

Functions.

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

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

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

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

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

Products.

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

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

$$\frac{v_1\ \mathit{val} \quad v_2\ \mathit{val}}{\langle v_1, v_2\rangle\ \mathit{val}}\ \mathsf{val/pair}$$

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

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

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

Unit.

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

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

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

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

Sums.

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

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

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

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

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

Lazy Records.

$$\frac{\Delta\,;\,\Gamma \vdash e_i : \tau_i \quad \text{(for all } i \in I\text{)}}{\Delta\,;\,\Gamma \vdash \langle\!|\, i \Rightarrow e_i \,|\!\rangle_{i \in I} : \&_{i \in I}(i : \tau_i)}\ \mathsf{lrec} \qquad \frac{\Delta\,;\,\Gamma \vdash e : \&_{i \in I}(i : \tau_i) \quad (j \in I)}{\Delta\,;\,\Gamma \vdash e \cdot j : \tau_j}\ \mathsf{lproj}$$

$$\frac{}{\langle\!|\, i \Rightarrow e_i \,|\!\rangle_{i \in I}\ \mathit{val}}\ \mathsf{val/lrec}$$

$$\frac{e \mapsto e'}{e \cdot j \mapsto e' \cdot j}\ \mathsf{step/case/lrec}_0 \qquad \frac{}{\langle\!|\, i \Rightarrow e_i \,|\!\rangle_{i \in I} \cdot j \mapsto e_j}\ \mathsf{step/case/lrec}$$

Recursive Types.

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

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

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

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

Fixed Point Expressions.

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

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

Universal Quantification.

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

$$\frac{}{\Lambda\alpha.\,e\ \mathit{val}}\ \mathsf{val/tplam}$$

$$\frac{e \mapsto e'}{e\,[\tau] \mapsto e'\,[\tau]}\ \mathsf{step/tpapp}_0 \qquad \frac{}{(\Lambda\alpha.\,e)[\tau] \mapsto [\tau/\alpha]e}\ \mathsf{step/tpapp}$$

Existential Quantification.

$$\frac{\Delta \vdash \sigma\ tp \quad \Delta\ ;\ \Gamma \vdash e : [\sigma/\alpha]\tau}{\Delta\ ;\ \Gamma \vdash \langle \sigma, e\rangle : \exists \alpha.\, \tau}\ \mathsf{exists} \qquad \frac{\Delta\ ;\ \Gamma \vdash e : \exists \alpha.\, \tau \qquad \Delta, \alpha\ tp\ ;\ \Gamma, x : \tau \vdash e' : \tau'}{\Delta\ ;\ \Gamma \vdash \mathbf{case}\ e\ (\langle \alpha, x\rangle \Rightarrow e') : \tau'}\ \mathsf{case/exists}$$

$$\frac{v\ val}{\langle \sigma, v\rangle\ val}\ \mathsf{val/exists} \qquad \frac{e \mapsto e'}{\langle \sigma, e\rangle \mapsto \langle \sigma, e'\rangle}\ \mathsf{step/exists}_1$$

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

$$\frac{v\ val}{\mathbf{case}\ \langle \sigma, v\rangle\ (\langle \alpha, x\rangle \Rightarrow e) \mapsto [\sigma/\alpha, v/x]e}\ \mathsf{step/case/exists}$$

Appendix: Process Language Reference

Syntax

Processes	P	$::=$	$x \leftarrow P\,;\,Q$	allocate/spawn
		$\mid$	$x^W \leftarrow y^R$	copy
		$\mid$	$x^W.\langle\,\rangle$ $\mid \mathsf{case}\ x^R\ (\langle\,\rangle \Rightarrow P)$	(1)
		$\mid$	$x^W.\langle y, z\rangle$ $\mid \mathsf{case}\ x^R\ (\langle y, z\rangle \Rightarrow P)$	$(\times)$
		$\mid$	$x^W.j(y)$ $\mid \mathsf{case}\ x^R\ (i(y) \Rightarrow P_i)_{i\in I}$	$(+)$
		$\mid$	$x^W.\mathsf{fold}(y)$ $\mid \mathsf{case}\ x^R\ (\mathsf{fold}(y) \Rightarrow P)$	(ρ)
		$\mid$	$x^R.\langle y, z\rangle$ $\mid \mathsf{case}\ x^W\ (\langle y, z\rangle \Rightarrow P)$	$(\rightarrow)$
		$\mid$	$x^R.j(y)$ $\mid \mathsf{case}\ x^W\ (i(y) \Rightarrow P_i)_{i\in I}$	$(\&)$

Dynamics

Values	V	$::=$	$\langle\,\rangle \mid \langle c, d\rangle \mid j(c) \mid \mathsf{fold}(c)$
Continuations	K	$::=$	$(\langle\,\rangle \Rightarrow P) \mid (\langle y, z\rangle \Rightarrow P) \mid (i(y) \Rightarrow P_i)_{i\in I} \mid (\mathsf{fold}(y) \Rightarrow P)$
Cell Contents	W	$::=$	$V \mid K$
Configurations	$\mathcal{C}$	$::=$	$\cdot \mid \mathcal{C}, \mathsf{proc}\ d\ P, \mathsf{cell}\ d\ \_ \mid \mathcal{C}, !\mathsf{cell}\ c\ W$

Passing Values to Continuations

$$
\begin{array}{lcll}
\langle\,\rangle & \triangleright & (\langle\,\rangle \Rightarrow P) & = P \\
\langle d_1, d_2\rangle & \triangleright & (\langle x_1, x_2\rangle \Rightarrow P) & = [d_1/x_1, d_2/x_2]P \\
j(d) & \triangleright & (i(y) \Rightarrow P_i)_{i\in I} & = [d/y]P_j \\
\mathsf{fold}(d) & \triangleright & (\mathsf{fold}(y) \Rightarrow P) & = [d/y]P
\end{array}
$$

Computation Rules

$\mathsf{proc}\ d\ (x \leftarrow P\,;\,Q)$	$\mapsto$	$\mathsf{proc}\ c\ ([c/x]P), \mathsf{cell}\ c\ \_, \mathsf{proc}\ d\ ([c/x]Q)$	(alloc/spawn)
$!\mathsf{cell}\ c\ W, \mathsf{proc}\ d\ (d^W \leftarrow c^R), \mathsf{cell}\ d\ \_$	$\mapsto$	$!\mathsf{cell}\ d\ W$	(copy)
$\mathsf{proc}\ d\ (d^W.V), \mathsf{cell}\ d\ \_$	$\mapsto$	$!\mathsf{cell}\ d\ V$	$(1, \times, +, \rho)$
$!\mathsf{cell}\ c\ V, \mathsf{proc}\ d\ (\mathsf{case}\ c^R\ K)$	$\mapsto$	$\mathsf{proc}\ d\ (V \triangleright K)$	$(1, \times, +, \rho)$
$\mathsf{proc}\ d\ (\mathsf{case}\ d^W\ K), \mathsf{cell}\ d\ \_$	$\mapsto$	$!\mathsf{cell}\ d\ K$	$(\rightarrow, \&)$
$!\mathsf{cell}\ c\ K, \mathsf{proc}\ d\ (c^R.V)$	$\mapsto$	$\mathsf{proc}\ d\ (V \triangleright K)$	$(\rightarrow, \&)$