

CHAPTER 6

Higher Order, Typed Functional Languages

- Types

τ over *Types* $\tau ::= int \mid \tau_1 \times \tau_2 \mid \tau_1 \rightarrow \tau_2$

- Integers

n, m, \dots over $N = \{\dots, -2, -1, 0, 1, 2, \dots\}$

- Arithmetic operators

$op \in \{+, -, \times\}$

- Variables

x, y, f, v, w, \dots over **Var**

1. Syntax

1.1. Eager Language.

- Terms

t over *Terms* $t ::= n \mid x \mid t_1 \mathbf{op} t_2 \mid$
 $\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \mid$
 $(t_1, t_2) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid$
 $(t_1 t_2) \mid \lambda x. t \mid$
 $\mathbf{rec} y. (\lambda x. t)$

1.2. Lazy Language.

- Terms

t over *Terms* $t ::= n \mid x \mid t_1 \mathbf{op} t_2 \mid$
 $\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \mid$
 $(t_1, t_2) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid$
 $(t_1 t_2) \mid \lambda x. t \mid$
 $\mathbf{rec} x. t$

We will also feel free to denote the function application as $t_1(t_2)$ or $t_1 t_2$, instead of $(t_1 t_2)$.

1.3. Typing Rules.

Both the Eager and the Lazy languages are typed languages. For all terms t and types τ , by $t : \tau$ we denote that the term t has type τ .

Now we list the rules which give a type to each term of the Eager language and the Lazy language (actually, we will have two variants of the Lazy language: the Lazy1 language and the Lazy2 language with the same operational semantics and two distinct denotational semantics).

One can show that every term can be given exactly one type.

$x : \tau$	(the type τ is known from the identifier of the variable x)	
$n : int$	for each integer $n \in \mathbb{N}$	
$\frac{t_1 : int \quad t_2 : int}{t_1 \text{ op } t_2 : int}$		
$\frac{t_0 : int \quad t_1 : \tau \quad t_2 : \tau}{\text{if } t_0 \text{ then } t_1 \text{ else } t_2 : \tau}$		
$\frac{t_1 : \tau_1 \quad t_2 : \tau_2}{(t_1, t_2) : \tau_1 \times \tau_2}$	$\frac{t : \tau_1 \times \tau_2}{\mathbf{fst}(t) : \tau_1}$	$\frac{t : \tau_1 \times \tau_2}{\mathbf{snd}(t) : \tau_2}$
$\frac{x : \tau_1 \quad t : \tau_2}{\lambda x. t : \tau_1 \rightarrow \tau_2}$	$\frac{t_1 : \tau_1 \rightarrow \tau_2 \quad t_2 : \tau_1}{(t_1 t_2) : \tau_2}$	
$\frac{y : \tau \quad \lambda x. t : \tau}{\mathbf{rec } y. (\lambda x. t) : \tau}$	(for the Eager language) τ is of the form $\tau_1 \rightarrow \tau_2$	
$\frac{x : \tau \quad t : \tau}{\mathbf{rec } x. t : \tau}$	(for the Lazy language)	

2. Operational Semantics

The operational semantics rules are structural, that is, they are based on the structure of the syntactic terms.

A *context* $C[-]$ is a term without a subterm, that is, a term ‘with a hole’. In the following table we list some contexts with a missing subterm. $C[t]$ denotes the context $C[-]$ where we have inserted the term t in the place of the missing subterm.

	context $C[-]$	missing subterm t	complete term $C[t]$
(i)	$(t [-])$	t_1	$(t t_1)$
(ii)	$[-] \mathbf{op} t_2$	t_1	$t_1 \mathbf{op} t_2$
(iii)	$\mathbf{if} [-] \mathbf{then} t_1 \mathbf{else} t_2$	t_0	$\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2$
(iv)	$[-]$	t	t

Note that in Case (iv) the missing subterm is the whole term. In this case the context is called the *empty context*.

2.1. Eager Operational Semantics.

- Canonical forms

c over *Canonical forms* $c ::= n \mid (c_1, c_2) \mid \lambda x.t$
where $\lambda x.t$ is a closed term.

Take a *closed* term t of type τ .

The eager operational semantics relation $t \rightarrow^e c$ is defined as follows. The superscript e stands for *eager*. For simplicity, we write $t \rightarrow c$, instead of $t \rightarrow^e c$. If $t \rightarrow^e c$ we also say that the eager operational evaluation of t yields the canonical form c .

$c \rightarrow c$

$$\frac{t_1 \rightarrow c_1 \quad t_2 \rightarrow c_2}{t_1 \mathbf{op} t_2 \rightarrow c}$$

where $c = c_1 \mathit{op} c_2$ and op is the semantic operation corresponding to \mathbf{op}

$$\frac{t_0 \rightarrow 0 \quad t_1 \rightarrow c_1}{\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \rightarrow c_1}$$

$$\frac{t_0 \rightarrow n \quad t_2 \rightarrow c_2 \quad n \neq 0}{\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \rightarrow c_2}$$

$$\frac{t_1 \rightarrow c_1 \quad t_2 \rightarrow c_2}{(t_1, t_2) \rightarrow (c_1, c_2)}$$

$$\frac{t \rightarrow (c_1, c_2)}{\mathbf{fst}(t) \rightarrow c_1}$$

$$\frac{t \rightarrow (c_1, c_2)}{\mathbf{snd}(t) \rightarrow c_2}$$

$$\frac{t_1 \rightarrow \lambda x.t \quad t_2 \rightarrow c_2 \quad t[c_2/x] \rightarrow c}{(t_1 t_2) \rightarrow c}$$

$$\mathbf{rec} y.(\lambda x.t) \rightarrow \lambda x.(t[\mathbf{rec} y.(\lambda x.t)/y])$$

NOTE 2.1. The eager operational semantics rules define a so called *big-step semantics* in the sense that for every term t_1 and t_2 , if $t_1 \rightarrow^e t_2$ then t_2 is a canonical value for the eager operational semantics.

If $t_1 \rightarrow^e t_2$ we will say that t_2 is the *eager operational value* of t_1 . □

NOTE 2.2. In the definition of the eager operational semantics we do *not* have a context rule of the form:

$$\frac{t \rightarrow c}{C[t] \rightarrow C[c]} \text{ for every context } C[-]. \quad \square$$

2.2. Lazy Operational Semantics.

- Canonical forms

c over *Canonical forms* $c ::= n \mid (t_1, t_2) \mid \lambda x.t$

where t_1 , t_2 , and $\lambda x.t$ are closed terms.

Take a *closed* term t of type τ .

The lazy operational semantics relation $t \rightarrow^\ell c$ is defined as follows. The superscript ℓ stands for *lazy*. For simplicity we write $t \rightarrow c$, instead of $t \rightarrow^\ell c$. If $t \rightarrow^\ell c$ we also say that the lazy operational evaluation of t yields the canonical form c .

$$c \rightarrow c$$

$$\frac{t_1 \rightarrow c_1 \quad t_2 \rightarrow c_2}{t_1 \mathbf{op} t_2 \rightarrow c}$$

where $c = c_1 \mathit{op} c_2$ and op is the semantic operation corresponding to **op**

$$\frac{t_0 \rightarrow 0 \quad t_1 \rightarrow c_1}{\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \rightarrow c_1}$$

$$\frac{t_0 \rightarrow n \quad t_2 \rightarrow c_2 \quad n \neq 0}{\mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \rightarrow c_2}$$

$$\frac{t \rightarrow (t_1, t_2) \quad t_1 \rightarrow c_1}{\mathbf{fst}(t) \rightarrow c_1}$$

$$\frac{t \rightarrow (t_1, t_2) \quad t_2 \rightarrow c_2}{\mathbf{snd}(t) \rightarrow c_2}$$

$$\frac{t_1 \rightarrow \lambda x.t \quad t[t_2/x] \rightarrow c}{(t_1 t_2) \rightarrow c}$$

$$\frac{t[(\mathbf{rec} x.t)/x] \rightarrow c}{\mathbf{rec} x.t \rightarrow c}$$

NOTE 2.3. In the definition of the lazy operational semantics we do *not* have a context rule of the form:

$$\frac{t \rightarrow c}{C[t] \rightarrow C[c]} \text{ for every context } C[-]. \quad \square$$

NOTE 2.4. In the lazy operational semantics there are no rules for pairs with conclusion of the form $(t_1, t_2) \rightarrow (c_1, c_2)$ because for all terms t_1 and t_2 , (t_1, t_2) is a canonical form. \square

NOTE 2.5. The lazy operational semantics rules define a big-step semantics in the sense that for every term t_1 and t_2 , if $t_1 \rightarrow^\ell t_2$ then t_2 is a canonical value for the lazy operational semantics.

If $t_1 \rightarrow^\ell t_2$ we will say that t_2 is the *lazy operational value* of t_1 . \square

NOTE 2.6. Both in the eager operational semantics and lazy operational semantics the evaluation of the operations **op**'s is done by evaluating its arguments first. Analogously, the evaluation of the **if** t_0 **then** t_1 **else** t_2 is done by evaluating the condition t_0 first and then either the arm t_1 or the arm t_2 , according to the value of t_0 . \square

NOTE 2.7. We have the following difference between the eager operational semantics and the lazy operational semantics in the case of pairs.

(i) In the eager semantics the canonical form of a pair is the pair of the canonical forms of the two components. For any pair t , in order to evaluate **fst**(t) we need to evaluate the canonical form of both the first and the second component of t . Analogously for the the evaluation of second component **snd**(t).

(ii) In the lazy semantics any pair (t_1, t_2) is already in canonical form, and for any pair t in order to evaluate **fst**(t) we need to evaluate only the canonical form c_1 of t_1 . Analogously for the evaluation of the second component **snd**(t).

We also have the following difference between the eager operational semantics and the lazy operational semantics in the case of function application.

(i) In the eager semantics in order to evaluate the function application $(t_1 t_2)$, we first evaluate t_1 and t_2 to their canonical forms, say c_1 and c_2 , respectively, and then we apply c_1 to c_2 .

(ii) In the lazy semantics we evaluate t_1 to its canonical form, say c_1 , and then we apply c_1 to t_2 .

Other differences between the eager operational semantics and the lazy operational semantics are in the syntax and semantics of the **rec** construct which are evident from the rules we have given above. \square

NOTE 2.8. The eager operational rule for the **rec**, that is,

$$\mathbf{rec} y.(\lambda x.t) \rightarrow \lambda x.(t[\mathbf{rec} y.(\lambda x.t)/y])$$

follows from the usual semantics of recursion via unfolding. Indeed, the rule one expects via unfolding is

$$\frac{(\lambda x.t)[(\mathbf{rec} y.(\lambda x.t))/x] \rightarrow c}{\mathbf{rec} y.(\lambda x.t) \rightarrow c}$$

and it reduces to $\mathbf{rec} y.(\lambda x.t) \rightarrow \lambda x.(t[\mathbf{rec} y.(\lambda x.t)/y])$ because:

(i) $(\lambda x.t)[(\mathbf{rec} y.(\lambda x.t))/x]$ is equal to $\lambda x.(t[(\mathbf{rec} y.(\lambda x.t))/x])$ (recall that x and y are distinct variables having distinct types), and

(ii) $\lambda x.(t[(\mathbf{rec} y.(\lambda x.t))/x]) \rightarrow^e \lambda x.(t[\mathbf{rec} y.(\lambda x.t)/y])$ because any term of the form $\lambda x.t$ is a canonical form in the Eager language. \square

2.3. Operational Evaluation in Linear Form.

The deduction, that is, a proof tree which justifies the operational evaluation of a given term, will often be written *in linear form*. This form is obtained by visiting the proof tree in a preorder way. Thus, for instance, instead of writing:

$$\frac{\frac{a \rightarrow a' \quad b \rightarrow b'}{a + b \rightarrow c} \quad d \rightarrow d'}{(a + b) + d \rightarrow e}$$

we will write:

$$(a+b)+d \rightarrow (a'+b)+d \rightarrow (a'+b')+d \rightarrow c+d \rightarrow c+d' \rightarrow e.$$

Analogously, instead of writing:

$$\frac{t_1 \rightarrow \lambda x.t \quad t_2 \rightarrow c_2 \quad t[c_2/x] \rightarrow c}{(t_1 t_2) \rightarrow c}$$

we will write:

$$(t_1 t_2) \rightarrow ((\lambda x.t) t_2) \rightarrow ((\lambda x.t) c_2) \rightarrow t[c_2/x] \rightarrow c.$$

In the case of the lazy operational semantics, instead of writing:

$$\frac{t[\mathbf{rec} x.t/x] \rightarrow c}{\mathbf{rec} x.t \rightarrow c}$$

we will write:

$$\mathbf{rec} x.t \rightarrow t[\mathbf{rec} x.t/x] \rightarrow c.$$

REMARK 2.9. When the evaluation of a term t is written in linear form, the term u occurring in a pair of the form $t \rightarrow u$, need not be a canonical form. \square

DEFINITION 2.10. [**Eager Canonical Form of a Term**] We will say that a term t has the *eager canonical form* c iff its eager operational evaluation in linear form is a sequence of the form: $t \rightarrow^e \dots \rightarrow^e c$ and c is a term in eager canonical form.

DEFINITION 2.11. [**Lazy Canonical Form of a Term**] We will say that a term t has the *lazy canonical form* c iff its lazy operational evaluation in linear form is a sequence of the form: $t \rightarrow^\ell \dots \rightarrow^\ell c$ and c is a term in lazy canonical form.

2.4. Eager Operational Semantics in Action: The Factorial Function.

Let us now show the evaluation, according to the eager operational semantics, of (*fact 2*), where *fact* is the factorial function.

In order to evaluate (*fact 2*) we have to consider the initial term:

$$((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1)))2).$$

Here is the evaluation of (*fact 2*) according to the eager operational semantics.

$$\begin{aligned} & ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1)))2) \\ & \rightarrow ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) 2) \\ & \rightarrow \mathbf{if} 2 \mathbf{then} 1 \mathbf{else} 2 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (2-1)) \\ & \rightarrow 2 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (2-1)) \tag{e2} \\ & \rightarrow 2 \times ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) (2-1)) \\ & \rightarrow 2 \times ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) 1) \\ & \rightarrow 2 \times (\mathbf{if} 1 \mathbf{then} 1 \mathbf{else} 1 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (1-1))) \end{aligned}$$

$$\begin{aligned}
&\rightarrow 2 \times (1 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (1-1))) & (e1) \\
&\rightarrow 2 \times (1 \times ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times \\
&\quad ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) (1-1))) \\
&\rightarrow 2 \times (1 \times ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) 0)) \\
&\rightarrow 2 \times (1 \times (\mathbf{if} 0 \mathbf{then} 1 \mathbf{else} 0 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (0-1)))) \\
&\rightarrow 2 \times (1 \times 1) & (e0) \\
&\rightarrow 2 \times 1 \rightarrow 2.
\end{aligned}$$

Note that the derivation between term (e1) and term (e0) is similar to the derivation between term (e2) and term (e1).

2.5. Lazy Operational Semantics in Action: The Factorial Function.

Here is the evaluation of $fact(2)$ according to the lazy operational semantics.

$$\begin{aligned}
&((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1)))2) \\
&\rightarrow ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) 2) \\
&\rightarrow \mathbf{if} 2 \mathbf{then} 1 \mathbf{else} 2 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (2-1)) \\
&\rightarrow 2 \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (2-1)) & (\ell2) \\
&\rightarrow 2 \times ((\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1))) (2-1)) \\
&\rightarrow 2 \times (\mathbf{if} 2-1 \mathbf{then} 1 \mathbf{else} (2-1) \times (((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) ((2-1)-1)))) \\
&\rightarrow 2 \times (\mathbf{if} 1 \mathbf{then} 1 \mathbf{else} (2-1) \times (((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) ((2-1)-1)))) \\
&\rightarrow 2 \times ((2-1) \times (((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) ((2-1)-1)))) \\
&\rightarrow 2 \times (1 \times (((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) ((2-1)-1)))) & (\ell1) \\
&\rightarrow 2 \times (1 \times (\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times \\
&\quad ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (x-1)) ((2-1)-1))) \\
&\rightarrow 2 \times (1 \times (\mathbf{if} ((2-1)-1) \mathbf{then} 1 \mathbf{else} ((2-1)-1) \times \\
&\quad ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (((2-1)-1)-1)))) \\
&\rightarrow 2 \times (1 \times (\mathbf{if} (1-1) \mathbf{then} 1 \mathbf{else} ((2-1)-1) \times \\
&\quad ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (((2-1)-1)-1)))) \\
&\rightarrow 2 \times (1 \times (\mathbf{if} 0 \mathbf{then} 1 \mathbf{else} ((2-1)-1) \times \\
&\quad ((\mathbf{rec} f.(\lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} x \times f(x-1))) (((2-1)-1)-1)))) \\
&\rightarrow 2 \times (1 \times 1) & (\ell0) \\
&\rightarrow 2 \times 1 \rightarrow 2
\end{aligned}$$

Note that the derivation between term ($\ell1$) and term ($\ell0$) is similar to the derivation between term ($\ell2$) and term ($\ell1$).

2.6. Adding the let-in construct: the Operational Semantics.

Let us assume that we have one more syntactic construct:

let $x \leftarrow t_1$ **in** t

with the following typing rule:

$$\frac{x : \tau_1 \quad t_1 : \tau_1 \quad t : \tau}{\mathbf{let} x \leftarrow t_1 \mathbf{in} t : \tau}$$

and the following eager operational semantics rule:

$$\frac{t_1 \rightarrow c_1 \quad t[c_1/x] \rightarrow c}{\mathbf{let } x \Leftarrow t_1 \mathbf{ in } t \rightarrow c}$$

and the following lazy operational semantics:

$$\frac{t[t_1/x] \rightarrow c}{\mathbf{let } x \Leftarrow t_1 \mathbf{ in } t \rightarrow c}$$

The construct $\mathbf{let } x \Leftarrow t_1 \mathbf{ in } t$ has the same eager operational semantics as $((\lambda x.t) t_1)$. Indeed:

$$\frac{t_1 \rightarrow c_1 \quad t[c_1/x] \rightarrow c}{\mathbf{let } x \Leftarrow t_1 \mathbf{ in } t \rightarrow c} \quad \text{and} \quad \frac{\lambda x.t \rightarrow \lambda x.t \quad t_1 \rightarrow c_1 \quad t[c_1/x] \rightarrow c}{((\lambda x.t) t_1) \rightarrow c}$$

and, obviously, $\lambda x.t \rightarrow \lambda x.t$ always holds because $\lambda x.t$ is a canonical form.

The construct $\mathbf{let } x \Leftarrow t_1 \mathbf{ in } t$ has the same lazy operational semantics as $((\lambda x.t) t_1)$. Indeed, we have that:

$$\frac{t[t_1/x] \rightarrow c}{\mathbf{let } x \Leftarrow t_1 \mathbf{ in } t \rightarrow c} \quad \text{and} \quad \frac{\lambda x.t \rightarrow \lambda x.t \quad t[t_1/x] \rightarrow c}{((\lambda x.t) t_1) \rightarrow c}$$

and, obviously, $\lambda x.t \rightarrow \lambda x.t$ always holds because $\lambda x.t$ is a canonical form.

3. Eager, Lazy1, and Lazy2 Denotational Semantics

Take a closed term t of type τ . For all terms t and types τ , $t : \tau$ denotes that the type of t is τ .

The domain of values of type τ is the cpo V_τ .

The canonical form c of a term t of the Eager, Lazy1, and Lazy2 languages is indicated in Table 1. The corresponding semantic domains are indicated in Table 2 on the next page.

Type τ	Eager language	Lazy1 language and Lazy2 language
$\tau = int$	n	n
$= \tau_1 \times \tau_2$	(c_1, c_2)	(t_1, t_2)
$= \tau_1 \rightarrow \tau_2$	$\lambda x.t$ closed	$\lambda x.t$ closed

TABLE 1. The canonical forms for the Eager, Lazy1, and Lazy2 languages.

Type τ	Eager Semantics	Lazy1 Semantics	Lazy2 Semantics
	$\rho \in \mathbf{Var} \rightarrow \bigcup_{\tau} V_{\tau}$	$\rho \in \mathbf{Var} \rightarrow \bigcup_{\tau} (V_{\tau})_{\perp}$	$\rho \in \mathbf{Var} \rightarrow \bigcup_{\tau} V_{\tau}$
$\tau = int$ $= \tau_1 \times \tau_2$ $= \tau_1 \rightarrow \tau_2$	$V_{\tau} = N$ $= V_{\tau_1} \times V_{\tau_2}$ $= [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]$	$V_{\tau} = N$ $= (V_{\tau_1})_{\perp} \times (V_{\tau_2})_{\perp}$ $= [(V_{\tau_1})_{\perp} \rightarrow (V_{\tau_2})_{\perp}]$	$V_{\tau} = N_{\perp}$ $= V_{\tau_1} \times V_{\tau_2}$ $= [V_{\tau_1} \rightarrow V_{\tau_2}]$
	$\llbracket t \rrbracket^e \rho \in (V_{\tau})_{\perp}$	$\llbracket t \rrbracket^{l1} \rho \in (V_{\tau})_{\perp}$	$\llbracket t \rrbracket^{l2} \rho \in V_{\tau}$

TABLE 2. The semantic domains for the Eager, Lazy1, and Lazy2 languages.

Eager Denotational Semantics. For simplicity, we write $\llbracket _ \rrbracket$, instead of $\llbracket _ \rrbracket^e$.	
$\llbracket n \rrbracket \rho = \lfloor n \rfloor$	
$\llbracket x \rrbracket \rho = \lfloor \rho(x) \rfloor$	
$\llbracket t_1 \mathbf{op} t_2 \rrbracket \rho = \llbracket t_1 \rrbracket \rho \mathit{op}_{\perp} \llbracket t_2 \rrbracket \rho$	
$\llbracket \mathbf{if} t_0 \mathbf{then} t_1 \mathbf{else} t_2 \rrbracket \rho = \mathit{Cond}(\llbracket t_0 \rrbracket \rho, \llbracket t_1 \rrbracket \rho, \llbracket t_2 \rrbracket \rho)$	
$\llbracket (t_1, t_2) \rrbracket \rho = \mathit{let} v_1 \leftarrow \llbracket t_1 \rrbracket \rho, v_2 \leftarrow \llbracket t_2 \rrbracket \rho \cdot \lfloor (v_1, v_2) \rfloor$	
$\llbracket \mathbf{fst}(t) \rrbracket \rho = \mathit{let} v \leftarrow \llbracket t \rrbracket \rho \cdot \lfloor \pi_1(v) \rfloor$	
$\llbracket \mathbf{snd}(t) \rrbracket \rho = \mathit{let} v \leftarrow \llbracket t \rrbracket \rho \cdot \lfloor \pi_2(v) \rfloor$	
$\llbracket t_1 t_2 \rrbracket \rho = \mathit{let} \varphi \leftarrow \llbracket t_1 \rrbracket \rho, v \leftarrow \llbracket t_2 \rrbracket \rho \cdot \varphi(v)$	
$\llbracket \lambda x.t \rrbracket \rho = \lfloor \lambda v. \llbracket t \rrbracket \rho [v/x] \rfloor$	
$\llbracket \mathbf{rec} y.(\lambda x.t) \rrbracket \rho = \lfloor \mathit{fix} (\lambda f. \lambda v. \llbracket t \rrbracket \rho [v/x, f/y]) \rfloor$	(recGlynn)
$= \mathit{fix} (\lambda f. \llbracket \lambda x.t \rrbracket \rho [\mathit{down}(f)/y])$	(recAlberto)

NOTE 3.1. The right hand side of the semantic equation for $\llbracket \lambda x.t \rrbracket \rho$ cannot be simplified to $\lfloor \lambda x. \llbracket t \rrbracket \rho \rfloor$. If we make this wrong simplification, in fact, the eager semantics value of the application $(\lambda x.x) 4$ in the environment ρ such that $\rho(x) = 1$, is $\lfloor 1 \rfloor$, instead of the expected value $\lfloor 4 \rfloor$. Indeed,

$$\begin{aligned}
\llbracket (\lambda x.x) 4 \rrbracket \rho &= \\
&= (\lambda x. (\llbracket x \rrbracket \rho)) 4 = \\
&= (\lambda x. \lfloor 1 \rfloor) 4 = \lfloor 1 \rfloor.
\end{aligned}$$

Note also that in the expression $(\lambda x. (\llbracket x \rrbracket \rho)) 4$, the binder λx , which is a piece of semantics, binds the variable x in $\llbracket x \rrbracket$, which is a piece of syntax, and this is an indication that the expression $\lambda x. (\llbracket x \rrbracket \rho)$ is not correct. \square

NOTE 3.2. The *fix* operator in the expression (**recGlynn**) is different from the *fix* operator in the expression (**recAlberto**) because they have different types (see below). Actually, there is a *fix* operator of type $[[V_\tau \rightarrow V_\tau] \rightarrow V_\tau]$ for each type τ . \square

NOTE 3.3. Instead of $fix(\lambda f.e)$, we will also write $\mu f.e$. By definition of the minimal fixpoint $fix(\lambda f.e)$ of the function $\lambda f.e$, we have that: $\mu f.e = (\lambda f.e)(\mu f.e)$. \square

NOTE 3.4. op_\perp is the strict semantic function in $[N_\perp \times N_\perp \rightarrow N_\perp]$ corresponding to **op**. Thus, for all x and y ,

$$x \text{ op}_\perp y = \begin{cases} \perp & \text{if } x = \perp \text{ or } y = \perp \\ [x' + y'] & \text{if } x = [x'] \text{ and } y = [y'] \end{cases} \quad \square$$

NOTE 3.5. The denotational semantics for the Eager language satisfies the following context rule: for all context $C[_]$ and for all typable, closed terms t_1 and t_2 such that $C[t_1]$ and $C[t_2]$ are typable, closed terms,

$$\text{if } [[t_1]] \rho = [[t_2]] \rho \text{ then } [[C[t_1]]] \rho = [[C[t_2]]] \rho. \quad \square$$

NOTE 3.6. Let us assume that in the term **rec** $y.(\lambda x.t)$ we have that:

- (i) $y : \tau_1 \rightarrow \tau_2$
- (ii) $x : \tau_1$
- (iii) $t : \tau_2$

Then in Equations (**recGlynn**) and (**recAlberto**) we have that:

$$\begin{aligned} [[\text{rec } y.(\lambda x.t)]] \rho &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]_\perp \\ \rho &\in \mathbf{Var} \rightarrow V_{\tau_1} \cup [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp] \end{aligned}$$

In Equation (**recGlynn**) we also have that:

$$\begin{aligned} f &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp] \\ v &\in V_{\tau_1} \\ [[t]] \rho &\in (V_{\tau_2})_\perp \\ \text{fix} &\in [[V_{\tau_1} \rightarrow (V_{\tau_2})_\perp] \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]] \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp] \end{aligned}$$

In Equation (**recAlberto**) we also have that:

$$\begin{aligned} f &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]_\perp \\ [[\lambda x.t]] \rho &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]_\perp \\ \text{down}(f) &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp] \\ \text{fix} &\in [[V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]_\perp \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]_\perp] \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_\perp]_\perp \quad \square \end{aligned}$$

Now we show that, indeed, the two definitions of $[[\text{rec } y.(\lambda x.t)]] \rho$ for the eager semantics are equivalent, that is, they define the same semantic value. Thus, we have to show that:

$$[[\text{fix}(\lambda f.d)]] = \text{fix}(\lambda f.[d[\text{down}(f)/f]]) \quad (G)$$

where: (i) d is of the form $\lambda x.t$ for some $x \in V_{\tau_1}$ and some term $t \in (V_{\tau_2})_{\perp}$, and (ii) $d[\text{down}(f)/f]$ is the expression d where each occurrence of f has been replaced by $\text{down}(f)$.

Thus, on the left hand side of (G):

$$\begin{aligned} f &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}] \\ d &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}] \\ \text{fix} &\in [[V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}] \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]] \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}] \end{aligned}$$

while on the right hand side:

$$\begin{aligned} f &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp} \\ d &\in [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}] \quad (\text{not } [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp}) \\ \text{fix} &\in [[V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp} \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp}] \rightarrow [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp} \end{aligned}$$

Thus, in (G) the two occurrences of fix denote two different operators because their types are different.

We first show that:

$$\lfloor \text{fix}(\lambda f.d) \rfloor \sqsubseteq \text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor) \quad (G1)$$

and then we show that:

$$\lfloor \text{fix}(\lambda f.d) \rfloor \sqsupseteq \text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor) \quad (G2)$$

Proof of (G1). Since $\lfloor _ \rfloor$ is continuous $\lfloor \text{fix}(\lambda f.d) \rfloor = \text{fix} \lfloor (\lambda f.d) \rfloor$. Thus, we have to show that:

$$\bigsqcup_{i \in \omega} \lfloor (\lambda f.d)^i(\perp) \rfloor \sqsubseteq \bigsqcup_{i \in \omega} (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)$$

and we will prove this by showing that

$$\text{for all } i \geq 0, \lfloor (\lambda f.d)^i(\perp) \rfloor \sqsubseteq \bigsqcup_{i \in \omega} (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp).$$

The proof by induction on i .

(Basis) We have to show that $\lfloor \perp \rfloor \sqsubseteq \bigsqcup_{i \in \omega} (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)$.

This holds because $\lfloor d[\text{down}(f)/f] \rfloor \neq \perp \in [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp}$, and thus,

$$\lfloor \perp \rfloor \sqsubseteq (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)(\perp).$$

To help the reader to better understand this inequality let us consider the types of its subexpressions. Let V_{τ} denote the cpo $[V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]$. Then, on the left hand side $\perp \in V_{\tau}$ and on the right hand side the first two occurrences of f are in $(V_{\tau})_{\perp}$ while the third one is in V_{τ} and $\perp \in (V_{\tau})_{\perp}$.

(Step) Assume that $\lfloor (\lambda f.d)^i(\perp) \rfloor \sqsubseteq \text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)$. We have to show that:

$$\lfloor (\lambda f.d)((\lambda f.d)^i(\perp)) \rfloor \sqsubseteq \text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor).$$

Indeed, for the left hand side we have that:

$$\begin{aligned} \lfloor (\lambda f.d)((\lambda f.d)^i(\perp)) \rfloor &= \\ &= \lfloor d[(\lambda f.d)^i(\perp)/f] \rfloor = \{ \text{by } \text{down}(\lfloor x \rfloor) = x \} = \\ &= \lfloor d[\text{down}(\lfloor (\lambda f.d)^i(\perp) \rfloor)/f] \rfloor. \end{aligned} \quad (G1.1)$$

For the right hand side we have that:

$$\begin{aligned} \text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor) &= \\ &= (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor) (\text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)) = \end{aligned}$$

$$= \lfloor d[\text{down}(\text{fix}(\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)) / f] \rfloor. \quad (G1.2)$$

Now we have that (G1.1) \sqsubseteq (G1.2), by induction hypothesis, and monotonicity of *down*, substitution, and $\lfloor _ \rfloor$. This completes the proof of (G1).

Proof of (G2). We have to show that:

$$\lfloor \bigsqcup_{i \in \omega} (\lambda f. d)^i(\perp) \rfloor \sqsupseteq \bigsqcup_{i \in \omega} (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)$$

and we will prove this by showing that

$$\text{for all } i \geq 0, \lfloor \bigsqcup_{i \in \omega} (\lambda f. d)^i(\perp) \rfloor \sqsupseteq (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp).$$

The proof by induction on i .

(*Basis*) Obviously, we have that: $\lfloor \bigsqcup_{i \in \omega} (\lambda f. d)^i(\perp) \rfloor \sqsupseteq \perp \in [V_{\tau_1} \rightarrow (V_{\tau_2})_{\perp}]_{\perp}$.

(*Step*) Assume that $\lfloor \text{fix}(\lambda f. d) \rfloor \sqsupseteq (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)$. We have to show that:

$$\lfloor \text{fix}(\lambda f. d) \rfloor \sqsupseteq (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor) ((\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)).$$

Indeed, for the left hand side we have that:

$$\begin{aligned} \lfloor d[\text{fix}(\lambda f. d)/f] \rfloor &= \{ \text{by } \text{down}(\lfloor x \rfloor) = x \} = \\ &= \lfloor d[\text{down}(\lfloor \text{fix}(\lambda f. d) \rfloor) / f] \rfloor. \end{aligned} \quad (G2.1)$$

For the right hand side we have that:

$$\begin{aligned} (\lambda f. \lfloor d[\text{down}(f)/f] \rfloor) ((\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)) &= \\ = \lfloor d[\text{down}((\lambda f. \lfloor d[\text{down}(f)/f] \rfloor)^i(\perp)) / f] \rfloor \end{aligned} \quad (G2.2)$$

Now we have that (G2.1) \sqsupseteq (G2.2) by induction hypothesis, and monotonicity of *down*, substitution, and $\lfloor _ \rfloor$. This completes the proof of (G2).

3.1. Computing the Factorial by Eager Denotational Semantics.

Now let us see in action the eager denotational semantics and let us compute the value of $\llbracket (\text{rec fact. } \lambda x. \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1)) (2) \rrbracket$. We have:

$$\begin{aligned} \llbracket (\text{rec fact. } \lambda x. \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1)) (2) \rrbracket \rho &= \\ = \text{let } \varphi &\Leftarrow \llbracket \text{rec fact. } \lambda x. \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho, v \Leftarrow \llbracket 2 \rrbracket \rho. \varphi(v) = \\ = \text{let } \varphi &\Leftarrow \lfloor \mu\psi. \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}, v/x] \rfloor, v \Leftarrow \llbracket 2 \rrbracket \rho. \varphi(v) = \\ &= (\mu\psi. \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}, v/x]) (2) \end{aligned} \quad (E2)$$

Now we may proceed by computing by mathematical induction the minimal fixpoint $\mu\psi. \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}, v/x]$. A different way of proceeding is to use the following equation, which holds in the eager semantics:

$$\mu x. t = (\lambda x. t)(\mu x. t) \quad (\text{FixUnfold})$$

This equation characterizes the fixpoint $\mu x. t$ and allows us to unfold the fixpoint as many times as required for its evaluation for any given argument. Now we clarify this sentence by performing the unfolding of the fixpoint in the case of our fixpoint

$$\mu\psi. \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}, v/x]$$

with the argument 2.

From (E2) by applying (*FixUnfold*), we get:

$$\begin{aligned} (\lambda\psi. \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}, v/x]) \\ (\mu\psi. \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}, v/x]) (2) = \\ = \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \end{aligned}$$

$$\begin{aligned}
& \rho[(\mu\psi.\lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}, v/x])/\mathit{fact}, 2/x] = \\
& = \mathit{Cond}(\llbracket x \rrbracket \rho[\dots, 2/x], \llbracket 1 \rrbracket \rho[\dots, 2/x], \llbracket x \times \mathit{fact}(x-1) \rrbracket \rho[\dots, 2/x]) = \\
& = \mathit{Cond}(\llbracket 2 \rrbracket, \llbracket 1 \rrbracket \rho[\dots], \llbracket x \times \mathit{fact}(x-1) \rrbracket \rho[\dots]) = \\
& = \llbracket x \times \mathit{fact}(x-1) \rrbracket \rho[(\mu\psi.\lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \\
& \qquad \qquad \qquad \rho[\psi/\mathit{fact}, v/x])/\mathit{fact}, 2/x] = \\
& = \llbracket x \rrbracket \rho[\dots, 2/x] \times_{\perp} \llbracket \mathit{fact}(x-1) \rrbracket \rho[\dots, 2/x] = \\
& = \llbracket 2 \rrbracket \times_{\perp} (\mathit{let} \ \varphi \Leftarrow \llbracket \mathit{fact} \rrbracket \rho[\dots, 2/x], v \Leftarrow \llbracket x-1 \rrbracket \rho[\dots, 2/x].\varphi(v)) = \\
& = \llbracket 2 \rrbracket \times_{\perp} (\mathit{let} \ \varphi \Leftarrow \llbracket \mu\psi.\lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}, v/x] \rrbracket, \\
& \qquad \qquad \qquad v \Leftarrow \llbracket x \rrbracket \rho[\dots, 2/x] \perp \llbracket 1 \rrbracket \rho[\dots, 2/x].\varphi(v)) = \\
& = \llbracket 2 \rrbracket \times_{\perp} ((\mu\psi.\lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}, v/x]) \\
& \qquad \qquad \qquad (\mathit{down}(\llbracket 2 \rrbracket \perp \llbracket 1 \rrbracket))) = \\
& = \llbracket 2 \rrbracket \times_{\perp} ((\mu\psi.\lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}, v/x]) (1)) \quad (E1)
\end{aligned}$$

Now if we compare expressions (E1) and (E2), we can see that, by performing from (E1) a sequence of evaluation steps analogous to the sequence which leads from (E2) to (E1), we eventually get:

$$\llbracket 2 \rrbracket \times_{\perp} (\llbracket 1 \rrbracket \times_{\perp} ((\mu\psi.\lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}, v/x]) (0)))$$

Then, after a few more evaluation steps we get:

$$\llbracket 2 \rrbracket \times_{\perp} (\llbracket 1 \rrbracket \times_{\perp} \llbracket 1 \rrbracket)$$

which is equal to $\llbracket 2 \rrbracket$, as expected.

3.2. Two Equivalent Expressions in the Eager Denotational Semantics.

In this section we will show that the following equality holds in the eager denotational semantics:

$$\llbracket ((\lambda f.(fx)) (\mathbf{rec} f.\lambda x.t)) \rrbracket \rho = \llbracket ((\mathbf{rec} f.\lambda x.t) x) \rrbracket \rho \quad (\mathit{RecDef})$$

For the left hand side of (*RecDef*) we have:

$$\begin{aligned}
& \llbracket ((\lambda f.(fx)) (\mathbf{rec} f.\lambda x.t)) \rrbracket \rho = \\
& = \mathit{let} \ \varphi \Leftarrow \llbracket \lambda f.(fx) \rrbracket \rho, v \Leftarrow \llbracket \mathbf{rec} f.(\lambda x.t) \rrbracket \rho. \varphi(v) = \\
& = \mathit{let} \ \varphi \Leftarrow \llbracket \lambda g. \llbracket fx \rrbracket \rho[g/f] \rrbracket, v \Leftarrow \llbracket \mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x] \rrbracket. \varphi(v) = \\
& = (\lambda g. \llbracket fx \rrbracket \rho[g/f]) (\mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x]) = \\
& = \llbracket fx \rrbracket \rho[(\mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x])/f] = \\
& = \mathit{let} \ \varphi \Leftarrow \llbracket \mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x] \rrbracket, v \Leftarrow \llbracket \rho(x) \rrbracket. \varphi(v) = \\
& = (\mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x]) \rho(x)
\end{aligned}$$

For the right hand side of (*RecDef*) we have:

$$\begin{aligned}
& \llbracket ((\mathbf{rec} f.(\lambda x.t)) x) \rrbracket \rho = \\
& = \mathit{let} \ \varphi \Leftarrow \llbracket \mathbf{rec} f.(\lambda x.t) \rrbracket \rho, v \Leftarrow \llbracket x \rrbracket \rho. \varphi(v) = \\
& = \mathit{let} \ \varphi \Leftarrow \llbracket \mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x] \rrbracket, v \Leftarrow \llbracket \rho(x) \rrbracket. \varphi(v) = \\
& = (\mu g.\lambda v. \llbracket t \rrbracket \rho[g/f, v/x]) \rho(x)
\end{aligned}$$

Thus, Equation (*RecDef*) has been proved. \square

NOTE 3.7. (i) Equation (*RecDef*) is a consequence of the β -rule in lambda calculus. However, the β -rule does *not* hold, in general, in the eager denotational semantics as we will see later.

(ii) As a consequence of the above Equation (*RecDef*) we have, for instance, that:

$$\begin{aligned} & \llbracket (\lambda \text{fact}. \text{fact}(2)) (\mathbf{rec} \text{fact}. \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \text{fact}(x - 1)) \rrbracket \rho = \\ & = \llbracket (\mathbf{rec} \text{fact}. \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \text{fact}(x - 1))(2) \rrbracket \rho. \end{aligned}$$

Lazy1 Denotational Semantics. For simplicity, we write $\llbracket _ \rrbracket$, instead of $\llbracket _ \rrbracket^{l1}$.

$$\begin{aligned} \llbracket n \rrbracket \rho &= \lfloor n \rfloor \\ \llbracket x \rrbracket \rho &= \rho(x) \\ \llbracket t_1 \mathbf{op} t_2 \rrbracket \rho &= \llbracket t_1 \rrbracket \rho \ \text{op}_\perp \ \llbracket t_2 \rrbracket \rho \\ \llbracket \mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \rrbracket \rho &= \text{Cond}(\llbracket t_0 \rrbracket \rho, \llbracket t_1 \rrbracket \rho, \llbracket t_2 \rrbracket \rho) \\ \llbracket (t_1, t_2) \rrbracket \rho &= \lfloor (\llbracket t_1 \rrbracket \rho, \llbracket t_2 \rrbracket \rho) \rfloor \\ \llbracket \mathbf{fst}(t) \rrbracket \rho &= \text{let } v \Leftarrow \llbracket t \rrbracket \rho \cdot \pi_1(v) \\ \llbracket \mathbf{snd}(t) \rrbracket \rho &= \text{let } v \Leftarrow \llbracket t \rrbracket \rho \cdot \pi_2(v) \\ \llbracket t_1 t_2 \rrbracket \rho &= \text{let } \varphi \Leftarrow \llbracket t_1 \rrbracket \rho \cdot \varphi(\llbracket t_2 \rrbracket \rho) \\ \llbracket \lambda x. t \rrbracket \rho &= \lfloor \lambda v. \llbracket t \rrbracket \rho [v/x] \rfloor \\ \llbracket \mathbf{rec} \ x. t \rrbracket \rho &= \text{fix}(\lambda f. \llbracket t \rrbracket \rho [f/x]) \end{aligned}$$

Lazy2 Denotational Semantics. For simplicity, we write $\llbracket _ \rrbracket$, instead of $\llbracket _ \rrbracket^{l2}$.

$$\begin{aligned} \llbracket n \rrbracket \rho &= \lfloor n \rfloor \\ \llbracket x \rrbracket \rho &= \rho(x) \\ \llbracket t_1 \mathbf{op} t_2 \rrbracket \rho &= \llbracket t_1 \rrbracket \rho \ \text{op}_\perp \ \llbracket t_2 \rrbracket \rho \\ \llbracket \mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2 \rrbracket \rho &= \text{Cond}(\llbracket t_0 \rrbracket \rho, \llbracket t_1 \rrbracket \rho, \llbracket t_2 \rrbracket \rho) \\ \llbracket (t_1, t_2) \rrbracket \rho &= (\llbracket t_1 \rrbracket \rho, \llbracket t_2 \rrbracket \rho) \\ \llbracket \mathbf{fst}(t) \rrbracket \rho &= \pi_1(\llbracket t \rrbracket \rho) \\ \llbracket \mathbf{snd}(t) \rrbracket \rho &= \pi_2(\llbracket t \rrbracket \rho) \\ \llbracket t_1 t_2 \rrbracket \rho &= (\llbracket t_1 \rrbracket \rho)(\llbracket t_2 \rrbracket \rho) \\ \llbracket \lambda x. t \rrbracket \rho &= \lambda v. \llbracket t \rrbracket \rho [v/x] \\ \llbracket \mathbf{rec} \ x. t \rrbracket \rho &= \text{fix}(\lambda f. \llbracket t \rrbracket \rho [f/x]) \end{aligned}$$

NOTE 3.8. As in the case of the denotational semantics for the Eager language, the denotational semantics for the Lazy1 language and Lazy2 language satisfy the

following context rule: for all context $C[_]$ and for all typable, closed terms t_1 and t_2 such that $C[t_1]$ and $C[t_2]$ are typable, closed terms,

$$\text{if } \llbracket t_1 \rrbracket \rho = \llbracket t_2 \rrbracket \rho \text{ then } \llbracket C[t_1] \rrbracket \rho = \llbracket C[t_2] \rrbracket \rho. \quad \square$$

3.3. Computing the Factorial by Lazy1 Denotational Semantics.

Now let us see in action the lazy1 denotational semantics and let us compute the value of $\llbracket (\text{rec fact. } \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1)) (2) \rrbracket$. We have:

$$\begin{aligned} & \llbracket (\text{rec fact. } \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1)) (2) \rrbracket \rho = \\ & = \text{let } \varphi \Leftarrow \llbracket \text{rec fact. } \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho. \varphi(\llbracket 2 \rrbracket \rho) = \\ & = \text{let } \varphi \Leftarrow (\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) . \varphi(\llbracket 2 \rrbracket) = \\ & = (\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) (\llbracket 2 \rrbracket) \end{aligned} \quad (L1.2)$$

Now we proceed by using the equation which holds in the lazy1 semantics:

$$\mu x. t = (\lambda x. t)(\mu x. t) \quad (\text{FixUnfold})$$

Thus, from (L1.2) by applying (*FixUnfold*), we get:

$$\begin{aligned} & (\lambda\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) \\ & \quad (\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) (\llbracket 2 \rrbracket) = \\ & = \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \\ & \quad \rho[(\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) / \text{fact}] (\llbracket 2 \rrbracket) = \\ & = \llbracket \lambda v. \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \\ & \quad \rho[(\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) / \text{fact}, v/x] \rrbracket (\llbracket 2 \rrbracket) = \\ & = \llbracket \text{if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \\ & \quad \rho[(\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) / \text{fact}, \llbracket 2 \rrbracket / x] = \\ & = \text{Cond}(\llbracket x \rrbracket \rho[\dots / \text{fact}, \llbracket 2 \rrbracket / x], \llbracket 1 \rrbracket \rho[\dots / \text{fact}, \llbracket 2 \rrbracket / x], \\ & \quad \llbracket x \times \text{fact}(x - 1) \rrbracket \rho[\dots / \text{fact}, \llbracket 2 \rrbracket / x]) = \\ & = \text{Cond}(\llbracket 2 \rrbracket, \llbracket 1 \rrbracket, \\ & \quad \llbracket 2 \rrbracket \times_{\perp} (\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) (\llbracket 2 \rrbracket -_{\perp} \llbracket 1 \rrbracket)) = \\ & = \llbracket 2 \rrbracket \times_{\perp} (\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) (\llbracket 1 \rrbracket) = \end{aligned} \quad (L1.1)$$

Now if we compare the expressions (L1.1) and (L1.2), we can see that, by performing from (L1.1) a sequence of evaluation steps analogous to the sequence which leads from (L1.2) to (L1.1), we eventually get:

$$\llbracket 2 \rrbracket \times_{\perp} (\llbracket 1 \rrbracket \times_{\perp} ((\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) (\llbracket 0 \rrbracket)))$$

Then, after a few more evaluation steps we get:

$$\llbracket 2 \rrbracket \times_{\perp} (\llbracket 1 \rrbracket \times_{\perp} \llbracket 1 \rrbracket) = \llbracket 2 \rrbracket, \text{ as desired.}$$

3.4. Computing the Factorial by Lazy2 Denotational Semantics.

Now let us see in action the lazy2 denotational semantics and let us compute the value of $\llbracket (\text{rec fact. } \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1)) (2) \rrbracket$. We have:

$$\begin{aligned} & \llbracket (\text{rec fact. } \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1)) (2) \rrbracket \rho = \\ & = (\llbracket \text{rec fact. } \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho) (\llbracket 2 \rrbracket \rho) = \\ & = (\mu\psi. \llbracket \lambda x. \text{ if } x \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) \rrbracket \rho[\psi/\text{fact}]) (\llbracket 2 \rrbracket) \end{aligned} \quad (L2.2)$$

Now we proceed by using the equation which holds in the lazy2 semantics:

$$\mu x.t = (\lambda x.t)(\mu x.t) \quad (\text{FixUnfold})$$

Thus, from (L2.2) by applying (*FixUnfold*), we get:

$$\begin{aligned} & (\lambda \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) \\ & \quad (\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) (\lfloor 2 \rfloor) = \\ & = \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \\ & \quad \rho[(\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) / \mathit{fact}] (\lfloor 2 \rfloor) = \\ & = \lambda v. \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \\ & \quad \rho[(\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) / \mathit{fact}, v/x] (\lfloor 2 \rfloor). \end{aligned}$$

Note that with respect to the evaluation of the lazy1 semantics, in the above expression there is no lifting. The evaluation proceeds as in the case of the lazy1 semantics and we then get:

$$\begin{aligned} & \llbracket \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \\ & \quad \rho[(\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) / \mathit{fact}, \lfloor 2 \rfloor / x] = \\ & = \mathit{Cond}(\llbracket x \rrbracket \rho[\dots / \mathit{fact}, \lfloor 2 \rfloor / x], \llbracket 1 \rrbracket \rho[\dots / \mathit{fact}, \lfloor 2 \rfloor / x], \\ & \quad \llbracket x \times \mathit{fact}(x-1) \rrbracket \rho[\dots / \mathit{fact}, \lfloor 2 \rfloor / x]) = \\ & = \mathit{Cond}(\lfloor 2 \rfloor, \lfloor 1 \rfloor, \\ & \quad \lfloor 2 \rfloor \times_{\perp} (\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) (\lfloor 2 \rfloor -_{\perp} \lfloor 1 \rfloor)) = \\ & = \lfloor 2 \rfloor \times_{\perp} (\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) (\lfloor 1 \rfloor) = \quad (L2.1) \end{aligned}$$

Now if we compare the expressions (L2.1) and (L2.2), we can see that, by performing from (L2.1) a sequence of evaluation steps analogous to the sequence which leads from (L2.2) to (L2.1), we eventually get:

$$\lfloor 2 \rfloor \times_{\perp} (\lfloor 1 \rfloor \times_{\perp} ((\mu \psi. \llbracket \lambda x. \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ x \times \mathit{fact}(x-1) \rrbracket \rho[\psi/\mathit{fact}]) (\lfloor 0 \rfloor)))$$

Then, after a few more evaluation steps we get:

$$\lfloor 2 \rfloor \times_{\perp} (\lfloor 1 \rfloor \times_{\perp} \lfloor 1 \rfloor) = \lfloor 2 \rfloor, \text{ as expected.}$$

3.5. Adding the let-in construct: the Denotational Semantics.

Let us assume that we have the syntactic construct:

$$\mathbf{let} \ x \Leftarrow t_1 \ \mathbf{in} \ t$$

we have introduced in Section 2.6. We want to have the equivalence of the denotational semantics of $\mathbf{let} \ x \Leftarrow t_1 \ \mathbf{in} \ t$ and the denotational semantics of $((\lambda x.t)t_1)$. Thus, in the eager denotational semantics we should have:

$$\llbracket \mathbf{let} \ x \Leftarrow t_1 \ \mathbf{in} \ t \rrbracket \rho = \llbracket t \rrbracket \rho[\mathit{down}(\llbracket t_1 \rrbracket \rho) / x]$$

and in the lazy1 and lazy2 denotational semantics we should have:

$$\llbracket \mathbf{let} \ x \Leftarrow t_1 \ \mathbf{in} \ t \rrbracket \rho = \llbracket t \rrbracket \rho[\llbracket t_1 \rrbracket \rho / x].$$

4. The alpha rule

- The α -rule **does not hold** in the eager operational semantics. The same for the lazy operational semantics.

Indeed, both for the eager operational semantics and the lazy operational semantics $\lambda x.t$ and $\lambda y.t[y/x]$ are different canonical forms. (As usual, $t[y/x]$ denotes the term t where each free occurrence of the variable x is replaced by y .)

- The α -rule **holds** in the eager denotational semantics, in the lazy1 denotational semantics, and in the lazy2 denotational semantics.

Let us consider the case of the eager denotational semantics. The cases of the lazy1 and the lazy2 denotational semantics are similar. We have that:

$$\begin{aligned} \llbracket \lambda x.t \rrbracket \rho &= \llbracket \lambda v. \llbracket t \rrbracket \rho[v/x] \rrbracket \quad \text{and} \\ \llbracket \lambda y.t[y/x] \rrbracket \rho &= \llbracket \lambda v. \llbracket t[y/x] \rrbracket \rho[v/y] \rrbracket \quad \text{and} \end{aligned}$$

the right hand sides of the above two equations are equal.

5. The beta rule

- The β -rule **does not hold** in the eager operational semantics.

It is *not* the case that:

$$((\lambda x.t) e) \rightarrow c \quad \text{iff} \quad t[e/x] \rightarrow c$$

Indeed, take $t = 1$ and $e = ((\mathbf{rec} y. \lambda x.(yx)) 5)$. We have the following Points (i) and (ii).

(i) No canonical form c exists such that $(\lambda x.1)e \rightarrow c$, because no canonical form c_2 exists such that $e \rightarrow c_2$. Indeed, since e is an application, by the following rules of the eager operational semantics (see the table on page 77)

$$\frac{t_1 \rightarrow \lambda x.t \quad t_2 \rightarrow c_2 \quad t[c_2/x] \rightarrow c}{(t_1 t_2) \rightarrow c} \quad (app)$$

$$\mathbf{rec} y. (\lambda x.t) \rightarrow \lambda x. (t[\mathbf{rec} y. (\lambda x.t)/y]) \quad (rec)$$

we have that:

there exists a canonical form c_2 such that $e \rightarrow c_2$ iff

{by (rec)}

there exists a canonical form c such that $\lambda x. ((\mathbf{rec} y. \lambda \tilde{x}. (y \tilde{x})) x) 5 \rightarrow c$ iff

{by (app)}

there exists a canonical form c such that $((\mathbf{rec} y. \lambda \tilde{x}. (y \tilde{x})) 5) \rightarrow c$

Now, since e is the term $((\mathbf{rec} y. \lambda \tilde{x}. (y \tilde{x})) 5)$, there is no proof that shows that there exists a canonical form c_2 such that $e \rightarrow c_2$.

(ii) We have that $t[e/x] \rightarrow 1[e/x] \rightarrow 1$ and 1 is a canonical form.

- The β -rule **holds** in the lazy operational semantics.

It is the case that:

$$((\lambda x.t) e) \rightarrow c \quad \text{iff} \quad t[e/x] \rightarrow c$$

because we have the following derived rule for the lazy operational semantics (see the table on page 78) when t_1 is $\lambda x.t$ and t_2 is e :

$$\frac{t[e/x] \rightarrow c}{((\lambda x.t) e) \rightarrow c}$$

- *The β -rule **does not hold** in the eager denotational semantics.*

We have that:

$$\llbracket (\lambda x.t) e \rrbracket \rho \neq \llbracket t \rrbracket \rho [\llbracket e \rrbracket \rho / x]$$

Indeed,

$$\begin{aligned} \llbracket (\lambda x.t) e \rrbracket \rho &= \\ &= \text{let } \varphi \Leftarrow [\lambda \tilde{v}. \llbracket t \rrbracket \rho [\tilde{v}/x]], v \Leftarrow \llbracket e \rrbracket \rho. \varphi(v) = \\ &= \text{let } v \Leftarrow \llbracket e \rrbracket \rho. (\lambda \tilde{v}. \llbracket t \rrbracket \rho [\tilde{v}/x])(v) = \\ &= \text{let } v \Leftarrow \llbracket e \rrbracket \rho. \llbracket t \rrbracket \rho [v/x] \end{aligned}$$

which is different from $\llbracket t \rrbracket \rho [\llbracket e \rrbracket \rho / x]$ whenever $\llbracket e \rrbracket \rho = \perp$.

Indeed, take $t=1$ and $e=((\mathbf{rec} y.\lambda x.(yx)) 5)$.

We have that $y : \text{int} \rightarrow \text{int}$ and $x : \text{int}$.

We have that

$$\llbracket (\lambda x.1) ((\mathbf{rec} y.\lambda x.(yx)) 5) \rrbracket \rho \neq \llbracket 1 \rrbracket \rho [\llbracket ((\mathbf{rec} y.\lambda x.(yx)) 5) \rrbracket \rho / x]. \quad (\dagger)$$

left-hand-side of $(\dagger) =$

$$\begin{aligned} &= \text{let } v \Leftarrow \llbracket (\mathbf{rec} y.\lambda x.(yx)) 5 \rrbracket \rho. \llbracket 1 \rrbracket \rho [v/x] = \\ &= \text{let } v \Leftarrow (\text{let } \varphi \Leftarrow \llbracket \mathbf{rec} y.\lambda x.(yx) \rrbracket \rho, v \Leftarrow \llbracket 5 \rrbracket \rho. \varphi(v)). \llbracket 1 \rrbracket \rho [v/x] = \\ &\quad \{\text{by Remark 5.1 below}\} \\ &= \text{let } v \Leftarrow (\lambda x.\perp) 5. \llbracket 1 \rrbracket \rho [v/x] = \\ &= \perp \in N_{\perp}, \quad \text{and} \end{aligned}$$

right-hand-side of $(\dagger) =$

$$= \llbracket 1 \rrbracket \rho [\llbracket (\mathbf{rec} y.\lambda x.(yx)) 5 \rrbracket \rho / x] = \llbracket 1 \rrbracket .$$

REMARK 5.1. Here is the proof that $\llbracket \mathbf{rec} y.\lambda x.(yx) \rrbracket \rho = [\lambda x.\perp]$.

$$\llbracket \mathbf{rec} y.\lambda x.(yx) \rrbracket \rho \in [N \rightarrow N_{\perp}]_{\perp}.$$

We have that:

$$\begin{aligned} \llbracket \mathbf{rec} y.\lambda x.(yx) \rrbracket \rho &= \\ &= [\mu \varphi.\lambda v. \text{let } y' \Leftarrow \llbracket y \rrbracket \rho [v/x, \varphi/y], x' \Leftarrow \llbracket x \rrbracket \rho [v/x, \varphi/y]. y'(x')] = \\ &= [\mu \varphi.\lambda v.\varphi(v)] = [\text{fix}(\lambda \varphi.\lambda v.\varphi(v))] = \{\text{see below}\} = \\ &= [\lambda x.\perp] \in [N \rightarrow N_{\perp}]_{\perp}. \end{aligned}$$

Indeed, we have that $\text{fix}(\lambda \varphi.\lambda v.\varphi(v)) = \lambda x.\perp$, with $\lambda x.\perp \in [N \rightarrow N_{\perp}]$, because we have that:

$$\begin{aligned} \bigsqcup_{n \geq 0} (\lambda \varphi.\lambda v.\varphi(v))^n(\perp) &= \bigsqcup_{n \geq 0} \tau^n(\perp) \quad \text{for } \tau = \lambda \varphi.\lambda v.\varphi(v) \text{ with } \perp \in [N \rightarrow N_{\perp}]_{\perp}. \\ \tau^0(\perp) &= \lambda x.\perp \in [N \rightarrow N_{\perp}] \quad \text{and} \\ \tau^1(\perp) &= \lambda v.((\lambda x.\perp)(v)) \text{ which is } \lambda v.\perp \in [N \rightarrow N_{\perp}]. \end{aligned}$$

where $\perp \in [N \rightarrow N_{\perp}]_{\perp}$ on the left hand sides, and $\perp \in N_{\perp}$ on the right hand sides.

Thus, $\bigsqcup_{n \geq 0} (\lambda \varphi. \lambda v. \varphi(v))^n(\perp) = \lambda v. \perp \in [N \rightarrow N_\perp]$. \square

- The β -rule **holds** in the lazy1 denotational semantics.

We have that:

$$\begin{aligned} \llbracket (\lambda x. t) e \rrbracket \rho &= \\ &= \text{let } \varphi \leftarrow \llbracket \lambda d. \llbracket t \rrbracket \rho[d/x] \rrbracket \cdot \varphi(\llbracket e \rrbracket \rho) = \\ &= (\lambda d. \llbracket t \rrbracket \rho[d/x]) (\llbracket e \rrbracket \rho) = \\ &= \llbracket t \rrbracket \rho[\llbracket e \rrbracket \rho/x]. \end{aligned}$$

- The β -rule **holds** in the lazy2 denotational semantics.

We have that:

$$\llbracket (\lambda x. t) e \rrbracket \rho = (\lambda d. \llbracket t \rrbracket \rho[d/x]) (\llbracket e \rrbracket \rho) = \llbracket t \rrbracket \rho[\llbracket e \rrbracket \rho/x].$$

6. The eta rule

- The η -rule **does not hold** in the eager operational semantics.

Let us consider the term $\lambda x. (fx)$ where x is not free in f . This term is a canonical form and it may be the case that there is no canonical form c such that $f \rightarrow c$.

Consider, for instance, $f = ((\mathbf{rec} y. \lambda x. (yx)) 5)$. Indeed,

$$f \rightarrow c \text{ iff } \lambda x. ((\mathbf{rec} y. \lambda \tilde{x}. (y \tilde{x})) x) 5 \rightarrow c \text{ iff } ((\mathbf{rec} y. \lambda \tilde{x}. (y \tilde{x})) 5) \rightarrow c$$

and thus, there is no finite proof which shows that there exists a canonical form c such that $f \rightarrow c$.

- The η -rule **does not hold** in the lazy operational semantics.

The proof is as in the case of the eager operational semantics, but consider the term $f = \mathbf{rec} w. w$, instead of $f = ((\mathbf{rec} y. \lambda x. (yx)) 5)$.

- The η -rule **does not hold** in the eager denotational semantics.

Consider $x : \sigma \quad f : \sigma \rightarrow \tau \quad \lambda x. (fx) : \sigma \rightarrow \tau$ with x not free in f .

We have:

$$\begin{aligned} \llbracket f \rrbracket \rho, \llbracket \lambda x. (fx) \rrbracket \rho &\in [V_\sigma \rightarrow (V_\tau)_\perp]_\perp \\ x : \sigma \quad \rho(x) \in V_\sigma \quad \llbracket x \rrbracket \rho &\in (V_\sigma)_\perp. \end{aligned}$$

Assume $\llbracket f \rrbracket \rho = \perp$ with $\perp \in [V_\sigma \rightarrow (V_\tau)_\perp]_\perp$.

We have:

$$\begin{aligned} \llbracket \lambda x. (fx) \rrbracket \rho &= \\ &= \llbracket \lambda w. \text{let } \varphi \leftarrow \llbracket f \rrbracket \rho[w/x], v \leftarrow \llbracket x \rrbracket \rho[w/x] \cdot \varphi(v) \rrbracket = \\ &= \llbracket \lambda w. \text{let } \varphi \leftarrow \llbracket f \rrbracket \rho, v \leftarrow \llbracket w \rrbracket \cdot \varphi(v) \rrbracket = \\ &= \llbracket \lambda w. \text{let } \varphi \leftarrow \llbracket f \rrbracket \rho \cdot \varphi(w) \rrbracket \quad (\text{where } w \in V_\sigma \text{ and } \varphi \in [V_\sigma \rightarrow (V_\tau)_\perp]) = \quad (*) \\ &= \{\text{because } \llbracket f \rrbracket \rho = \perp\} = \\ &= \llbracket \lambda w. \perp \rrbracket \end{aligned}$$

which is different from $\perp \in [V_\sigma \rightarrow (V_\tau)_\perp]_\perp$.

Note that if we assume that $\llbracket f \rrbracket \rho \neq \perp$ with $\perp \in [V_\sigma \rightarrow (V_\tau)_\perp]_\perp$ then the η -rule holds in the eager denotational semantics. Indeed,

$$\begin{aligned} \llbracket \lambda x.(fx) \rrbracket \rho &= \{\text{see (*) above}\} = \\ &= \llbracket \lambda w. \text{let } \varphi \Leftarrow \llbracket f \rrbracket \rho \bullet \varphi(w) \rrbracket = \{\text{evaluating the } \textit{let} \text{ expression}\} = \\ &= \llbracket \lambda w. (\text{down}(\llbracket f \rrbracket \rho) w) \rrbracket = \{\text{the } \eta\text{-rule holds in mathematics}\} = \\ &= \llbracket \text{down}(\llbracket f \rrbracket \rho) \rrbracket = \{\text{since } \llbracket f \rrbracket \rho \neq \perp\} = \\ &= \llbracket f \rrbracket \rho. \end{aligned}$$

- The η -rule **does not hold** in the lazy1 denotational semantics.

Consider $x : \sigma \quad f : \sigma \rightarrow \tau \quad \lambda x.(fx) : \sigma \rightarrow \tau$ with x not free in f .

We have:

$$\begin{aligned} \llbracket f \rrbracket \rho, \llbracket \lambda x.(fx) \rrbracket \rho &\in [(V_\sigma)_\perp \rightarrow (V_\tau)_\perp]_\perp \\ x : \sigma \quad \rho(x) \in V_\sigma \quad \llbracket x \rrbracket \rho &\in (V_\sigma)_\perp. \end{aligned}$$

Assume $\llbracket f \rrbracket \rho = \perp$ with $\perp \in [(V_\sigma)_\perp \rightarrow (V_\tau)_\perp]_\perp$.

We have:

$$\begin{aligned} \llbracket \lambda x.(fx) \rrbracket \rho &= \\ &= \llbracket \lambda d. \llbracket fx \rrbracket \rho[d/x] \rrbracket \quad (\text{where } d \in (V_\sigma)_\perp \text{ and } \varphi \in [V_\sigma \rightarrow (V_\tau)_\perp]) = \\ &= \llbracket \lambda d. \text{let } \varphi \Leftarrow \llbracket f \rrbracket \rho[d/x] \bullet \varphi(\llbracket x \rrbracket \rho[d/x]) \rrbracket \quad (\text{where } \varphi \in [(V_\sigma)_\perp \rightarrow (V_\tau)_\perp]) = \\ &= \llbracket \lambda d. \text{let } \varphi \Leftarrow \llbracket f \rrbracket \rho \bullet \varphi(d) \rrbracket = \tag{**} \\ &= \llbracket \lambda d. \perp \rrbracket, \text{ with } \llbracket \lambda d. \perp \rrbracket \in [(V_\sigma)_\perp \rightarrow (V_\tau)_\perp]_\perp \end{aligned}$$

which is different from \perp of $[(V_\sigma)_\perp \rightarrow (V_\tau)_\perp]_\perp$.

Note that if we assume that $\llbracket f \rrbracket \rho \neq \perp$ with $\perp \in [(V_\sigma)_\perp \rightarrow (V_\tau)_\perp]_\perp$ then the η -rule holds in the lazy1 denotational semantics. Indeed,

$$\begin{aligned} \llbracket \lambda x.(fx) \rrbracket \rho &= \{\text{see (**) above}\} = \\ &= \llbracket \lambda d. \text{let } \varphi \Leftarrow \llbracket f \rrbracket \rho \bullet \varphi(d) \rrbracket = \{\text{evaluating the } \textit{let} \text{ expression}\} = \\ &= \llbracket \lambda d. (\text{down}(\llbracket f \rrbracket \rho) d) \rrbracket = \{\text{the } \eta\text{-rule holds in mathematics}\} = \\ &= \llbracket \text{down}(\llbracket f \rrbracket \rho) \rrbracket = \{\text{because } \llbracket f \rrbracket \rho \neq \perp\} = \\ &= \llbracket f \rrbracket \rho. \end{aligned}$$

- The η -rule **holds** in the lazy2 denotational semantics.

Consider $x : \sigma \quad f : \sigma \rightarrow \tau \quad \lambda x.(fx) : \sigma \rightarrow \tau$ with x not free in f .

We have:

$$\begin{aligned} \llbracket f \rrbracket \rho, \llbracket \lambda x.(fx) \rrbracket \rho &\in [V_\sigma \rightarrow V_\tau] \\ x : \sigma \quad \rho(x) \in V_\sigma \quad \llbracket x \rrbracket \rho &\in V_\sigma. \end{aligned}$$

We have:

$$\begin{aligned} \llbracket \lambda x.(fx) \rrbracket \rho &= \\ &= \lambda d. (\llbracket fx \rrbracket \rho[d/x]) = \\ &= \lambda d. ((\llbracket f \rrbracket \rho[d/x]) (\llbracket x \rrbracket \rho[d/x])) = \\ &= \lambda d. ((\llbracket f \rrbracket \rho) d) = \end{aligned}$$

$$\begin{aligned}
&= \{\text{the } \eta\text{-rule holds in mathematics}\} = \\
&= \llbracket f \rrbracket \rho.
\end{aligned}$$

7. The fixpoint operators

In the Eager language, given the terms $x : \alpha$, $t : \beta$, $\lambda x.t : \tau$, and $y : \tau$, with $\tau = \alpha \rightarrow \beta$, there exists a term $R : (\tau \rightarrow \tau) \rightarrow \tau$ such that:

$$\llbracket R(\lambda y.\lambda x.t) \rrbracket \rho = \llbracket \mathbf{rec} y.(\lambda x.t) \rrbracket \rho \quad (\mathbf{E})$$

We will show that the term R can be taken to be $\mathbf{rec} w.\lambda f.\lambda x.((f(wf)) x)$ where $w : (\tau \rightarrow \tau) \rightarrow \tau$, $f : \tau \rightarrow \tau$, and $x : \alpha$, with $\tau = \alpha \rightarrow \beta$.

Other choices for R are possible.

In the Lazy1 language and the Lazy2 language, given the terms $y : \tau$, and $t : \tau$, there exists a term $R : (\tau \rightarrow \tau) \rightarrow \tau$ such that:

$$\llbracket R(\lambda y.t) \rrbracket \rho = \llbracket \mathbf{rec} y.t \rrbracket \rho \quad (\mathbf{L})$$

We will show that in the Lazy1 language and the Lazy2 language the term R can be taken to be $\mathbf{rec} w.\lambda f.(f(wf))$ where $w : (\tau \rightarrow \tau) \rightarrow \tau$ and $f : \tau \rightarrow \tau$.

Other choices for R are possible.

In the Eager language we have that:

$$\llbracket R \rrbracket \rho = \lfloor \lambda \varphi. \lfloor \mathit{fix}(\mathit{down} \circ \varphi) \rfloor \rfloor \in [[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp]_\perp \quad (\mathbf{RE})$$

$$\begin{aligned}
&\text{with } \varphi \in [V_\tau \rightarrow (V_\tau)_\perp], \quad \mathit{fix} \in [[V_\tau \rightarrow V_\tau] \rightarrow V_\tau], \text{ and} \\
&\mathit{down} \in [(V_\tau)_\perp \rightarrow V_\tau].
\end{aligned}$$

In the Lazy1 language we have that:

$$\llbracket R \rrbracket \rho = \lfloor \lambda \varphi. \mathit{fix}(\mathit{down} \varphi) \rfloor \in [[(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp]_\perp \quad (\mathbf{RL1})$$

$$\begin{aligned}
&\text{with } \varphi \in [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp, \quad \mathit{fix} \in [[(V_\tau)_\perp \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp], \text{ and} \\
&\mathit{down} \in [[(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]].
\end{aligned}$$

In the Lazy2 language we have that:

$$\llbracket R \rrbracket \rho = \mathit{fix} \in [[V_\tau \rightarrow V_\tau] \rightarrow V_\tau] \quad (\mathbf{RL2})$$

In the case of the Lazy2 language we require that the cpo V_τ is a cpo with the bottom element. This hypothesis is required, in particular, in the proof of **(RL2)**, where we need the bottom element of the cpo $[[V_\tau \rightarrow V_\tau] \rightarrow V_\tau]$ (see page 97).

The terms R are called *fixpoint operators* because for any given term $F : \tau \rightarrow \tau$, in the Eager language we have that:

$$\llbracket RF \rrbracket \rho = \llbracket F(RF) \rrbracket \rho \in (V_\tau)_\perp \quad \text{if } \mathit{down}(\llbracket F \rrbracket \rho) \neq (\lambda x.\perp) \in [V_\tau \rightarrow (V_\tau)_\perp] \quad (\mathbf{EF})$$

in the Lazy1 language we have that:

$$\llbracket RF \rrbracket \rho = \llbracket F(RF) \rrbracket \rho \in (V_\tau)_\perp \quad (\mathbf{LF1})$$

in the Lazy2 language we have that:

$$\llbracket RF \rrbracket \rho = \llbracket F(RF) \rrbracket \rho \in V_\tau \quad (\mathbf{LF2})$$

In **(EF)** the condition $\text{down}(\llbracket F \rrbracket \rho) \neq (\lambda x. \perp) \in [V_\tau \rightarrow (V_\tau)_\perp]$ is equivalent to the conjunction of the following two conditions:

- (i) $\llbracket F \rrbracket \rho \neq \perp \in [V_\tau \rightarrow (V_\tau)_\perp]_\perp$ where \perp is the bottom element in $[V_\tau \rightarrow (V_\tau)_\perp]_\perp$, and
- (ii) $\llbracket F \rrbracket \rho \neq (\lambda x. \perp) \in [V_\tau \rightarrow (V_\tau)_\perp]$ where $x \in V_\tau$ and \perp is the bottom element in $(V_\tau)_\perp$.

First we show **(RE)**, **(RL1)**, and **(RL2)**.

Proof of (RE). We have to show that $\llbracket R \rrbracket \rho = \lfloor \lambda \varphi. \lfloor \text{fix}(\text{down} \circ \varphi) \rfloor \rfloor$.

We have that $\llbracket R \rrbracket \rho \in [[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp]_\perp$.

Since R is **rec** $w. \lambda f. \lambda x. ((f(wf)) x)$, we have that:

$$\llbracket R \rrbracket \rho = \lfloor \text{fix}(\lambda u. \lambda \varphi. \lfloor \text{down}(\text{let } v \leftarrow u(\varphi) \bullet (\varphi v)) \rfloor) \rfloor$$

with $\varphi \in [V_\tau \rightarrow (V_\tau)_\perp]$, $u \in A$, and $\text{fix} \in [[A \rightarrow A] \rightarrow A]$,

where A stands for $[[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp]$.

Thus,

$$\begin{aligned} \llbracket R \rrbracket \rho &= \lfloor \bigsqcup_{n \in \omega} \chi^n(\perp) \rfloor \text{ with } \chi = \lambda u. \lambda \varphi. \lfloor \text{down}(\text{let } v \leftarrow u(\varphi) \bullet (\varphi v)) \rfloor \text{ and} \\ \chi &\in [[[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp] \rightarrow [[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp]]. \end{aligned}$$

We have that:

$$\begin{aligned} \chi^0(\perp) &= \perp \quad \text{with both } \perp \text{'s belonging to } [[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp] \\ &= \lambda \varphi. \perp \quad \text{with } \perp \in (V_\tau)_\perp \\ \chi^1(\perp) &= \lambda \varphi. \lfloor \text{down}(\text{let } v \leftarrow \perp \bullet (\varphi v)) \rfloor = \\ &= \lambda \varphi. \lfloor \text{down}(\varphi \perp) \rfloor = \\ &= \lambda \varphi. \lfloor (\text{down} \circ \varphi) \perp \rfloor \end{aligned}$$

where on the left hand side $\perp \in [[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp]$ and on the right hand sides $\perp \in (V_\tau)_\perp$.

$$\begin{aligned} \chi^2(\perp) &= \lambda \varphi. \lfloor \text{down}(\text{let } v \leftarrow \lambda \tilde{\varphi}. \lfloor (\text{down} \circ \tilde{\varphi}) \perp \rfloor (\varphi) \bullet (\varphi v)) \rfloor = \\ &= \lambda \varphi. \lfloor \text{down}(\text{let } v \leftarrow \lfloor (\text{down} \circ \varphi) \perp \rfloor \bullet (\varphi v)) \rfloor = \\ &= \lambda \varphi. \lfloor \text{down}(\varphi((\text{down} \circ \varphi) \perp)) \rfloor = \\ &= \lambda \varphi. \lfloor (\text{down} \circ \varphi)((\text{down} \circ \varphi) \perp) \rfloor = \\ &= \lambda \varphi. \lfloor (\text{down} \circ \varphi)^2(\perp) \rfloor \end{aligned}$$

where on the left hand side $\perp \in [V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp$ and on the right hand sides $\perp \in (V_\tau)_\perp$,

and, by induction, one can show that for all $n \geq 0$, we have that:

$$\chi^n(\perp) = \lambda \varphi. \lfloor (\text{down} \circ \varphi)^n \perp \rfloor$$

where on the left hand side $\perp \in [V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp$ and on the right hand side $\perp \in (V_\tau)_\perp$. Thus,

$$\begin{aligned}
\llbracket R \rrbracket \rho &= \llbracket \bigsqcup_{n \in \omega} \chi^n(\perp) \rrbracket = \\
&= \llbracket \bigsqcup_{n \in \omega} \lambda\varphi. \llbracket ((\text{down} \circ \varphi)^n \perp) \rrbracket \rrbracket = \{\text{lub's of functions are computed pointwise}\} = \\
&= \llbracket \lambda\varphi. \llbracket \bigsqcup_{n \in \omega} \llbracket ((\text{down} \circ \varphi)^n \perp) \rrbracket \rrbracket \rrbracket = \{\text{by continuity of } \llbracket _ \rrbracket \} = \\
&= \llbracket \lambda\varphi. \llbracket \bigsqcup_{n \in \omega} (\text{down} \circ \varphi)^n \perp \rrbracket \rrbracket = \{\text{by definition of } \text{fix} = \lambda f. \bigsqcup_{n \in \omega} f^n(\perp)\} = \\
&= \llbracket \lambda\varphi. \llbracket \text{fix}(\text{down} \circ \varphi) \rrbracket \rrbracket. \quad \square
\end{aligned}$$

Proof of (RL1). We have to show that $\llbracket R \rrbracket \rho = \llbracket \lambda\varphi. \text{fix}(\text{down} \varphi) \rrbracket$.

We have that $\llbracket R \rrbracket \rho \in \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp$.

Since R is $\text{rec } w. \lambda f. (f(wf))$, we have that:

$$\llbracket R \rrbracket \rho = \text{fix}(\lambda u. \llbracket \lambda\varphi. (\text{down}(\varphi))((\text{down}(u))\varphi) \rrbracket)$$

with $\varphi \in \llbracket (V_\tau)_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp$ and $u \in \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp$.

(Note that the two *down* operators are different because they have different types.)

Thus,

$$\begin{aligned}
\llbracket R \rrbracket \rho &= \bigsqcup_{n \in \omega} \chi^n(\perp) \text{ with } \chi = \lambda u. \llbracket \lambda\varphi. (\text{down}(\varphi))((\text{down}(u))\varphi) \rrbracket \text{ and} \\
\chi &\in \llbracket \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp \rightarrow \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp \rrbracket.
\end{aligned}$$

We have that:

$$\begin{aligned}
\chi^0(\perp) &= \perp \text{ with both } \perp \text{'s belonging to } \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp \\
\chi^1(\perp) &= \llbracket \lambda\varphi. (\text{down}(\varphi)\perp) \rrbracket
\end{aligned}$$

where on the left hand side $\perp \in \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp$ and on the right hand sides $\perp \in (V_\tau)_\perp$.

$$\begin{aligned}
\chi^2(\perp) &= \llbracket \lambda\varphi. (\text{down}(\varphi))((\text{down}(\llbracket \lambda\tilde{\varphi}. (\text{down}(\tilde{\varphi})\perp) \rrbracket))\varphi) \rrbracket = \\
&= \llbracket \lambda\varphi. (\text{down}(\varphi))((\lambda\tilde{\varphi}. (\text{down}(\tilde{\varphi})\perp))\varphi) \rrbracket = \\
&= \llbracket \lambda\varphi. (\text{down}(\varphi))((\text{down}(\varphi)\perp)) \rrbracket = \\
&= \llbracket \lambda\varphi. (\text{down}(\varphi))^2(\perp) \rrbracket
\end{aligned}$$

where on the left hand side $\perp \in \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp$ and on the right hand sides $\perp \in (V_\tau)_\perp$. By induction on n we have that for all $n \geq 0$,

$$\chi^n(\perp) = \llbracket \lambda\varphi. (\text{down}(\varphi))^n(\perp) \rrbracket$$

where on the left hand side $\perp \in \llbracket [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp \rrbracket_\perp$ and on the right hand side $\perp \in (V_\tau)_\perp$. Thus,

$$\begin{aligned}
\llbracket R \rrbracket \rho &= \bigsqcup_{n \in \omega} \chi^n(\perp) = \\
&= \bigsqcup_{n \in \omega} \llbracket \lambda\varphi. (\text{down}(\varphi))^n(\perp) \rrbracket = \{\text{by continuity of } \llbracket _ \rrbracket \} = \\
&= \llbracket \bigsqcup_{n \in \omega} \lambda\varphi. (\text{down}(\varphi))^n(\perp) \rrbracket = \{\text{lub's of functions are computed pointwise}\} = \\
&= \llbracket \lambda\varphi. \bigsqcup_{n \in \omega} (\text{down}(\varphi))^n(\perp) \rrbracket = \{\text{by definition of } \text{fix} = \lambda f. \bigsqcup_{n \in \omega} f^n(\perp)\} = \\
&= \llbracket \lambda\varphi. \text{fix}(\text{down}(\varphi)) \rrbracket. \quad \square
\end{aligned}$$

Proof of (RL2). We have to show that $\llbracket R \rrbracket \rho = \text{fix}$, where $R : (\tau \rightarrow \tau) \rightarrow \tau$.

We have that $\llbracket R \rrbracket \rho \in \llbracket [V_\tau \rightarrow V_\tau] \rightarrow V_\tau \rrbracket$.

Since R is $\mathbf{rec} w.\lambda f.(f(wf))$, we have that:

$$\begin{aligned} \llbracket R \rrbracket \rho &= \llbracket \mathbf{rec} w.\lambda f.(f(wf)) \rrbracket \rho = \\ &= \mathit{fix}(\lambda \tilde{w}.\llbracket \lambda f.(f(wf)) \rrbracket \rho[\tilde{w}/w]) = \\ &= \mathit{fix}(\lambda \tilde{w}.\lambda \tilde{f}.\llbracket f(wf) \rrbracket \rho[\tilde{w}/w, \tilde{f}/f]) = \{\text{by renaming of bound variables}\} = \\ &= \mathit{fix}(\lambda w.\lambda f.f(wf)). \end{aligned}$$

Thus,

$$\begin{aligned} \llbracket R \rrbracket \rho &= \bigsqcup_{n \in \omega} \chi^n(\perp) \text{ with } \chi = \lambda w.\lambda f.f(wf) \text{ and} \\ \chi &\in \llbracket [V_\tau \rightarrow V_\tau] \rightarrow V_\tau \rrbracket \rightarrow \llbracket [V_\tau \rightarrow V_\tau] \rightarrow V_\tau \rrbracket. \end{aligned}$$

We have that:

$$\begin{aligned} \chi^0(\perp) &= \perp \\ \chi^1(\perp) &= (\lambda w.\lambda f.f(wf))(\perp) = \lambda f.\perp \\ \chi^2(\perp) &= (\lambda w.\lambda f.f(wf))(\lambda f.\perp) = \lambda f.f(\perp) \\ \chi^3(\perp) &= (\lambda w.\lambda f.f(wf))(\lambda f.f(\perp)) = \lambda f.f(f(\perp)) \end{aligned}$$

where all \perp 's belong to $\perp \in \llbracket [V_\tau \rightarrow V_\tau] \rightarrow V_\tau \rrbracket$. By induction on n we have that for all $n \geq 0$, $\chi^n(\perp) = \lambda f.f^n(\perp)$. Thus,

$$\llbracket R \rrbracket \rho = \bigsqcup_{n \in \omega} \chi^n(\perp) = \bigsqcup_{n \in \omega} \lambda f.f^n(\perp) = \mathit{fix}. \quad \square$$

Proof of (E). For the Eager language we have to show that

$$\llbracket R(\lambda y.\lambda x.t) \rrbracket \rho = \llbracket \mathbf{rec} y.\lambda x.t \rrbracket \rho \quad (\mathbf{E})$$

with $R = \mathbf{rec} w.\lambda f.\lambda x.((f(wf))x)$ and $R : (\tau \rightarrow \tau) \rightarrow \tau$. By **(RE)** and the definition of the eager denotational semantics of $\mathbf{rec} y.\lambda x.t$, we have to show that

let $\tilde{\varphi} \Leftarrow \llbracket \lambda \varphi. \llbracket \mathit{fix}(\mathit{down} \circ \varphi) \rrbracket \rrbracket$, $v \Leftarrow \llbracket \lambda y.\lambda x.t \rrbracket \rho$. $\tilde{\varphi}(v) = \llbracket \mathit{fix}(\lambda u.\lambda v.\llbracket t \rrbracket \rho[u/y, v/x]) \rrbracket$, that is,

$$\text{let } v \Leftarrow \llbracket \lambda u.\llbracket \lambda x.t \rrbracket \rho[u/y] \rrbracket \cdot (\lambda \varphi. \llbracket \mathit{fix}(\mathit{down} \circ \varphi) \rrbracket)(v) = \llbracket \mathit{fix}(\lambda u.\lambda v.\llbracket t \rrbracket \rho[u/y, v/x]) \rrbracket.$$

Now, this last equality holds because:

$$\begin{aligned} \text{let } v \Leftarrow \llbracket \lambda u.\llbracket \lambda x.t \rrbracket \rho[u/y] \rrbracket \cdot (\lambda \varphi. \llbracket \mathit{fix}(\mathit{down} \circ \varphi) \rrbracket)(v) &= \\ &= \lambda \varphi. \llbracket \mathit{fix}(\mathit{down} \circ \varphi) \rrbracket(\lambda u.\llbracket \lambda x.t \rrbracket \rho[u/y] \rrbracket) = \\ &= \llbracket \mathit{fix}(\mathit{down} \circ (\lambda u.\llbracket \lambda x.t \rrbracket \rho[u/y] \rrbracket)) \rrbracket = \\ &= \llbracket \mathit{fix}(\mathit{down} \circ (\lambda u.\llbracket \lambda v.\llbracket t \rrbracket \rho[u/y, v/x] \rrbracket)) \rrbracket = \{\text{see } (\dagger 3) \text{ below}\} = \\ &= \llbracket \mathit{fix}(\lambda u.\lambda v.\llbracket t \rrbracket \rho[u/y, v/x]) \rrbracket. \end{aligned}$$

Thus, the proof is completed if the following equality holds between elements of the cpo $\llbracket [V_\tau \rightarrow V_\tau] \rrbracket$:

$$\mathit{down} \circ (\lambda u.\llbracket \lambda v.\llbracket t \rrbracket \rho[u/y, v/x] \rrbracket) = \lambda u.\lambda v.\llbracket t \rrbracket \rho[u/y, v/x] \quad (\dagger 3)$$

where $u : \tau$, $v : \alpha$, and $t : \beta$, with $\tau = \alpha \rightarrow \beta$. Indeed, for all r and s , we have that:

$$\begin{aligned} (\mathit{down} \circ (\lambda u.\llbracket r \rrbracket))s &= \mathit{down}((\lambda u.\llbracket r \rrbracket)s) = \mathit{down}\llbracket r[s/u] \rrbracket = r[s/u] = \\ &= (\lambda u.r)s \end{aligned}$$

Then, by instantiating this equality $(\mathit{down} \circ (\lambda u.\llbracket r \rrbracket))s = (\lambda u.r)s$ for r equal to the term $\lambda v.\llbracket t \rrbracket \rho[u/y, v/x]$, we get the desired equality $(\dagger 3)$. \square

Proof of (L) for the Lazy1 language. We have to show that

$$\llbracket R (\lambda y.t) \rrbracket \rho = \llbracket \mathbf{rec} y.t \rrbracket \rho \quad (\mathbf{L})$$

with $R = \mathbf{rec} w.\lambda f.(f(wf))$ and $R : (\tau \rightarrow \tau) \rightarrow \tau$. By **(RL1)** and the definition of the lazy1 denotational semantics of $\mathbf{rec} y.t$, we have to show that:

$$\text{let } \tilde{\varphi} \Leftarrow \llbracket \lambda \varphi. \text{fix}(\text{down } \varphi) \rrbracket \cdot \tilde{\varphi}(\llbracket \lambda y.t \rrbracket \rho) = \text{fix}(\lambda u. \llbracket t \rrbracket \rho[u/y]).$$

This equality holds because:

$$\begin{aligned} \text{let } \tilde{\varphi} \Leftarrow \llbracket \lambda \varphi. \text{fix}(\text{down } \varphi) \rrbracket \cdot \tilde{\varphi}(\llbracket \lambda y.t \rrbracket \rho) &= \{\text{by definition of the } \text{let} \text{ construct}\} = \\ &= \text{fix}(\text{down}(\llbracket \lambda y.t \rrbracket \rho)) = \{\text{by definition of } \llbracket \lambda y.t \rrbracket \rho\} = \\ &= \text{fix}(\text{down} \llbracket \lambda u. \llbracket t \rrbracket \rho[u/y] \rrbracket) = \{\text{by } \text{down}[x] = x\} = \\ &= \text{fix}(\lambda u. \llbracket t \rrbracket \rho[u/y]). \end{aligned} \quad \square$$

Proof of (L) for the Lazy2 language. We have to show that:

$$\llbracket R (\lambda y.t) \rrbracket \rho = \llbracket \mathbf{rec} y.t \rrbracket \rho \quad (\mathbf{L})$$

with $R = \mathbf{rec} w.\lambda f.(f(wf))$ and $R : (\tau \rightarrow \tau) \rightarrow \tau$. By **(RL2)** and the definition of the lazy2 denotational semantics of $\mathbf{rec} y.t$, we have to show that:

$$\text{fix}(\llbracket \lambda y.t \rrbracket \rho) = \text{fix}(\lambda u. \llbracket t \rrbracket \rho[u/y]).$$

This equality holds in the lazy2 denotational semantics because:

$$\llbracket \lambda y.t \rrbracket \rho = \lambda u. \llbracket t \rrbracket \rho[u/y]. \quad \square$$

Proof of (EF). Consider $F : \tau \rightarrow \tau$ and $R : (\tau \rightarrow \tau) \rightarrow \tau$. For the Eager language we have that:

$$\begin{aligned} \llbracket F \rrbracket \rho &\in [V_\tau \rightarrow (V_\tau)_\perp]_\perp \\ \llbracket R \rrbracket \rho &\in [[V_\tau \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp]_\perp \\ \text{fix} &\in [[V_\tau \rightarrow V_\tau] \rightarrow V_\tau] \\ \varphi &\in [V_\tau \rightarrow (V_\tau)_\perp] \\ \text{down} \circ \varphi &\in [V_\tau \rightarrow V_\tau] \\ \llbracket R \rrbracket \rho &= \llbracket \lambda \varphi. \llbracket \text{fix}(\text{down} \circ \varphi) \rrbracket \rrbracket \end{aligned}$$

We have to show that:

$$\llbracket RF \rrbracket \rho = \llbracket F(RF) \rrbracket \rho \quad \text{if } \text{down}(\llbracket F \rrbracket \rho) \neq (\lambda x.\perp) \in [V_\tau \rightarrow (V_\tau)_\perp]. \quad (\mathbf{EF})$$

For the left hand side of **(EF)** we have that:

$$\begin{aligned} \llbracket RF \rrbracket \rho &= \text{let } r \Leftarrow \llbracket R \rrbracket \rho, v \Leftarrow \llbracket F \rrbracket \rho \cdot r(v) = \\ &\quad \{\text{by definition of the } \text{let} \text{ construct}\} \\ &= (\text{down}(\llbracket R \rrbracket \rho)) (\text{down}(\llbracket F \rrbracket \rho)) = \\ &\quad \{\text{by definition of } \llbracket R \rrbracket \rho\} \\ &= \llbracket \text{fix}(\text{down} \circ (\text{down}(\llbracket F \rrbracket \rho))) \rrbracket = \\ &\quad \{\text{by definition of } \text{fix} \text{ and } \circ\} \\ &= \llbracket \text{down}((\text{down}(\llbracket F \rrbracket \rho))(\text{fix}(\text{down} \circ (\text{down}(\llbracket F \rrbracket \rho)))))) \rrbracket = \quad (\ddagger 1) \\ &\quad \{\text{by } \llbracket \text{down}(x) \rrbracket = x \text{ for } x \neq \perp \in (V_\tau)_\perp\} \\ &= (\text{down}(\llbracket F \rrbracket \rho))(\text{fix}(\text{down} \circ (\text{down}(\llbracket F \rrbracket \rho)))) \quad (\ddagger 2) \end{aligned}$$

This last step from expression $(\ddagger 1)$ to expression $(\ddagger 2)$ is justified by the fact that the argument of the leftmost down in $(\ddagger 1)$ is different from $\perp \in (V_\tau)_\perp$ because, by hypothesis, $\text{down}(\llbracket F \rrbracket \rho) \neq (\lambda x.\perp) \in [V_\tau \rightarrow (V_\tau)_\perp]$.

For the right hand side of **(EF)** we have that:

$$\begin{aligned}
\llbracket F(RF) \rrbracket \rho &= \text{let } f \Leftarrow \llbracket F \rrbracket \rho, x \Leftarrow (\text{let } r \Leftarrow \llbracket R \rrbracket \rho, v \Leftarrow \llbracket F \rrbracket \rho \cdot r(v)) \cdot f(x) = \\
&\quad \{\text{by definition of the } \text{let} \text{ construct}\} \\
&= (\text{down}(\llbracket F \rrbracket \rho)) (\text{down}((\text{down}(\llbracket R \rrbracket \rho)) (\text{down}(\llbracket F \rrbracket \rho)))) = \\
&\quad \{\text{by definition of } \llbracket R \rrbracket \rho\} \\
&= (\text{down}(\llbracket F \rrbracket \rho)) (\text{down}(\text{down} \lfloor \lambda \varphi. \lfloor \text{fix}(\text{down} \circ \varphi) \rfloor \rfloor) (\text{down}(\llbracket F \rrbracket \rho)))) = \\
&\quad \{\text{by } \text{down}(\lfloor x \rfloor) = x \text{ and function application}\} \\
&= (\text{down}(\llbracket F \rrbracket \rho)) (\text{down}(\lfloor \text{fix}(\text{down} \circ (\text{down}(\llbracket F \rrbracket \rho))) \rfloor)) = \\
&\quad \{\text{by } \text{down}(\lfloor x \rfloor) = x\} \\
&= (\text{down}(\llbracket F \rrbracket \rho)) (\text{fix}(\text{down} \circ (\text{down}(\llbracket F \rrbracket \rho)))). \tag{\dagger 3}
\end{aligned}$$

Since $(\dagger 2) = (\dagger 3)$ the proof is completed. \square

Proof of (LF1). Consider $F : \tau \rightarrow \tau$ and $R : (\tau \rightarrow \tau) \rightarrow \tau$. For the Lazy1 language we have that:

$$\begin{aligned}
\llbracket F \rrbracket \rho &\in [(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \\
\llbracket R \rrbracket \rho &\in [[(V_\tau)_\perp \rightarrow (V_\tau)_\perp]_\perp \rightarrow (V_\tau)_\perp]_\perp \\
\text{fix} &\in [[(V_\tau)_\perp \rightarrow (V_\tau)_\perp] \rightarrow (V_\tau)_\perp] \\
\llbracket R \rrbracket \rho &= \text{fix}
\end{aligned}$$

We have to show that:

$$\llbracket RF \rrbracket \rho = \llbracket F(RF) \rrbracket \rho \in (V_\tau)_\perp.$$

This equality holds because we have that:

$$\begin{aligned}
\llbracket RF \rrbracket \rho &= \text{let } \varphi \Leftarrow \llbracket R \rrbracket \rho \cdot \varphi(\llbracket F \rrbracket \rho) = \\
&= \text{let } \tilde{\varphi} \Leftarrow \lfloor \lambda \varphi. \text{fix}(\text{down } \varphi) \rfloor \cdot \tilde{\varphi}(\llbracket F \rrbracket \rho) = \\
&= (\lambda \varphi. \text{fix}(\text{down } \varphi))(\llbracket F \rrbracket \rho) = \\
&= \text{fix}(\text{down}(\llbracket F \rrbracket \rho)), \tag{\dagger 4} \\
\llbracket F(RF) \rrbracket \rho &= \text{let } \varphi \Leftarrow \llbracket F \rrbracket \rho \cdot \varphi(\llbracket RF \rrbracket \rho) = \{\text{see } (\dagger 4)\} = \\
&= \text{let } \varphi \Leftarrow \llbracket F \rrbracket \rho \cdot \varphi(\text{fix}(\text{down}(\llbracket F \rrbracket \rho))) = \\
&= (\text{down}(\llbracket F \rrbracket \rho)) (\text{fix}(\text{down}(\llbracket F \rrbracket \rho))), \text{ and}
\end{aligned}$$

for all x , $\text{fix } x = x(\text{fix } x)$. \square

Proof of (LF2). Consider $F : \tau \rightarrow \tau$ and $R : (\tau \rightarrow \tau) \rightarrow \tau$. For the Lazy2 language we have that:

$$\begin{aligned}
\llbracket F \rrbracket \rho &\in [V_\tau \rightarrow V_\tau] \\
\llbracket R \rrbracket \rho &\in [[V_\tau \rightarrow V_\tau] \rightarrow V_\tau] \\
\text{fix} &\in [[V_\tau \rightarrow V_\tau] \rightarrow V_\tau] \\
\llbracket R \rrbracket \rho &= \text{fix}
\end{aligned}$$

Since in the Lazy2 language we have that $\llbracket t_1 t_2 \rrbracket \rho = (\llbracket t_1 \rrbracket \rho)(\llbracket t_2 \rrbracket \rho)$, we have to show that $\llbracket RF \rrbracket \rho = \llbracket F(RF) \rrbracket \rho$, that is,

$$\text{fix}(\llbracket F \rrbracket \rho) = (\llbracket F \rrbracket \rho) (\text{fix}(\llbracket F \rrbracket \rho))$$

Indeed, this equality holds because for any $x \in [V_\tau \rightarrow V_\tau]$ we have that $\text{fix } x = x(\text{fix } x)$. \square

7.1. Operational Semantics of Fixpoint Operators.

7.1.1. Eager Operational Semantics.

Let R denote the fixpoint operator $\mathbf{rec} y.(\lambda f.\lambda x.((f(yf))x))$ of the eager operational semantics.

We show that in the eager operational semantics for all terms $F : \tau \rightarrow \tau$,

- (i) if F has no canonical form then both RF and $F(RF)$ have no canonical form, and
- (ii) if F has canonical form then RF has canonical form, while $F(RF)$ may or may not have canonical form.

Point (i) is immediate because in the eager operational semantics the evaluation of an application $(t_1 t_2)$ requires that both subterms t_1 and t_2 have canonical forms.

In order to show Point (ii) let us consider the term $F = \lambda u.t$ which is a canonical form. We have that:

$$\begin{aligned} RF &\equiv (\mathbf{rec} y.(\lambda f.\lambda x.((f(yf))x))) (\lambda u.t) \rightarrow \\ &\rightarrow (\lambda f.\lambda x.((f(yf))x) [\mathbf{rec} y.(\lambda g\lambda v.((g(yg))v)) / y]) (\lambda u.t) \rightarrow \\ &\rightarrow \lambda x.(((\lambda u.t)(y(\lambda u.t)))x) [\mathbf{rec} y.(\lambda g\lambda v.((g(yg))v))] / y \end{aligned} \quad (\alpha)$$

which is a canonical form. Let us call it α . We also have that:

$$F(RF) \equiv (\lambda u.t) ((\mathbf{rec} y.(\lambda f.\lambda x.((f(yf))x))) (\lambda u.t)) \rightarrow \dots \rightarrow (\lambda u.t) \alpha \rightarrow t[\alpha/u]$$

If in F we take t to be 1 then $RF \rightarrow \alpha[1/t]$ and $F(RF) \rightarrow 1$. Thus, we get the two distinct canonical forms $\alpha[1/t]$ and 1. We have that RF and $F(RF)$ have the same canonical form if in F we take t to be u . In that case, in fact, the terms α and $t[\alpha/u]$ are both equal to $\alpha[u/t]$.

If in F we take t to be $((\mathbf{rec} y.(\lambda f.(f(yf)))) (\lambda v.t_1))$ for some term t_1 then RF has the canonical form α with $((\mathbf{rec} y.(\lambda f.(f(yf)))) (\lambda v.t_1))$ instead of t , and $F(RF)$ has no canonical form in the eager operational semantics. Indeed, in the eager operational semantics the application:

$$(\mathbf{rec} y.(\lambda f.(f(yf)))) (\lambda v.t_1)$$

where $\mathbf{rec} y.(\lambda f.(f(yf)))$ is the fixpoint operator of the lazy operational semantics, has no canonical form.

7.1.2. Lazy Operational Semantics.

We show that in the lazy operational semantics for all terms $F : \tau \rightarrow \tau$ and every canonical form c , $RF \rightarrow c$ iff $F(RF) \rightarrow c$, where R denotes the fixpoint operator $\mathbf{rec} y.\lambda f.(f(yf))$ of the lazy operational semantics. Indeed, we have that:

$$RF \rightarrow c$$

{by definition of R }

$$\text{iff } (\mathbf{rec} y.\lambda f.(f(yf))) F \rightarrow c$$

{by operational rule for \mathbf{rec} }

$$\text{iff } \lambda f.(f((\mathbf{rec} y.\lambda g.(g(yg)))f)) F \rightarrow c$$

{by operational rule for function application}

$$\text{iff } F((\mathbf{rec} y.(\lambda g.g(yg)))F) \rightarrow c$$

{by definition of R }

$$\text{iff } F(RF) \rightarrow c.$$

8. Adequacy

Let us introduce the following definition.

DEFINITION 8.1. Given a cpo V_τ , a predicate $p_\tau : V_\tau \rightarrow \{true, false\}$ is said to be *canonically consistent* iff for all terms t of type τ and all environments ρ ,

$$p_\tau(\llbracket t \rrbracket^{\ell_2} \rho) = true \text{ iff} \\ \text{there exists a canonical form } c \text{ of type } \tau \text{ such that } \llbracket t \rrbracket^{\ell_2} \rho = \llbracket c \rrbracket^{\ell_2} \rho.$$

We have that for each type τ , there exists a unique canonically consistent predicate from V_τ to $\{true, false\}$ which we call $halt_\tau$.

We also need the following notations. For any closed, typable term t of type τ ,

- (i) $t \Downarrow^e$ (pronounced, t converges in the *eager operational* semantics) iff there exists a canonical form c such that $t \rightarrow^e c$, where \rightarrow^e denotes the eager operational semantics relation.
- (ii) $t \Downarrow^\ell$ (pronounced, t converges in the *lazy operational* semantics) iff there exists a canonical form c such that $t \rightarrow^\ell c$, where \rightarrow^ℓ denotes the lazy operational semantics relation.
- (iii) $t \Downarrow^e$ (pronounced, t converges in the *eager denotational* semantics) iff there exists $v \in V_\tau$ such that for all environments ρ , $\llbracket t \rrbracket^e \rho = \lfloor v \rfloor$.
- (iv) $t \Downarrow^{\ell_1}$ (pronounced, t converges in the *lazy1 denotational* semantics) iff there exists $v \in V_\tau$ such that for all environments ρ , $\llbracket t \rrbracket^{\ell_1} \rho = \lfloor v \rfloor$.
- (v) $t \Downarrow^{\ell_2}$ (pronounced, t converges in the *lazy2 denotational* semantics) iff for all environments ρ , $halt_\tau(\llbracket t \rrbracket^{\ell_2} \rho) = true$.

Note that the definition of convergence $t \Downarrow^{\ell_2}$ in the *lazy2* denotational semantics (see Point (v) above) cannot be given in a way which is analogous to that of convergence in the *lazy1* denotational semantics (see Point (iv) above), because for any term t of type τ , $\llbracket t \rrbracket^{\ell_2} \rho$ is of type V_τ , and not of type $(V_\tau)_\perp$. (Note, however, that for any term of type *int*, $\llbracket t \rrbracket^{\ell_2} \rho$ belongs to V_{int} which is N_\perp , not N .)

Now we introduce the notion of adequacy of the denotational semantics w.r.t. the operational semantics for the following three languages:

- (i) the Eager language,
- (ii) the Lazy language with the lazy1 denotational semantics, called Lazy1, and
- (iii) the Lazy language, with the lazy2 denotational semantics, called Lazy2.

These three definitions of adequacy follow the same pattern and they can be derived one from the other by making the changes indicated in the following Table 3.

DEFINITION 8.2. [**Adequacy of the Eager Denotational Semantics**] The eager denotational semantics $\llbracket _ \rrbracket^e$ is said to be *adequate* w.r.t. the eager operational semantics \rightarrow^e iff

- (A^e) for all closed, typed terms t , $t \Downarrow^e$ iff $t \Downarrow^e$, and
- (B^e) for all closed, typed terms t , there exists a canonical form c such that $t \rightarrow^e c$ implies $\llbracket t \rrbracket^e \rho = \llbracket c \rrbracket^e \rho$.

	Eager	Lazy1	Lazy2
operational semantics and operational convergence	$\rightarrow^e \quad \downarrow^e$	$\rightarrow^\ell \quad \downarrow^\ell$	$\rightarrow^\ell \quad \downarrow^\ell$
denotational semantics and operational convergence	$\llbracket _ \rrbracket^e \quad \Downarrow^e$	$\llbracket _ \rrbracket^{\ell 1} \quad \Downarrow^{\ell 1}$	$\llbracket _ \rrbracket^{\ell 2} \quad \Downarrow_{halt}^{\ell 2}$

TABLE 3. Notations for operational semantics, denotational semantics, and convergence. Note that Lazy1 and Lazy2 languages have the same operational semantics.

DEFINITION 8.3. [**Adequacy of the Lazy1 Denotational Semantics**] The lazy1 denotational semantics $\llbracket _ \rrbracket^{\ell 1}$ is said to be *adequate* w.r.t. the lazy operational semantics \rightarrow^ℓ iff

- (A^{ℓ1}) for all closed, typed terms t , $t \downarrow^\ell$ iff $t \Downarrow^{\ell 1}$, and
(B^{ℓ1}) for all closed, typed terms t , there exists a canonical form c such that
 $t \rightarrow^v c$ implies $\llbracket t \rrbracket^{\ell 1} \rho = \llbracket c \rrbracket^{\ell 1} \rho$.

DEFINITION 8.4. [**Adequacy of the Lazy2 Denotational Semantics**] The lazy2 denotational semantics $\llbracket _ \rrbracket^{\ell 2}$ is said to be *adequate* w.r.t. the lazy operational semantics \rightarrow^ℓ iff

- (A^{ℓ2}) for all closed term t of type τ , $t \downarrow^\ell$ iff $t \Downarrow^{\ell 2}$, and
(B^{ℓ2}) for all closed, typed terms t , there exists a canonical form c such that
 $t \rightarrow^\ell c$ implies $\llbracket t \rrbracket^{\ell 2} \rho = \llbracket c \rrbracket^{\ell 2} \rho$.

With reference to the above Definitions 8.2, 8.3, and 8.4, we have the following theorem.

THEOREM 8.5. For every closed term t of type int ,

- (i) (A^e) is equivalent to (B^e) with ‘implies’ replaced by ‘iff’ and
- (ii) (A^{ℓ1}) is equivalent to (B^{ℓ1}) with ‘implies’ replaced by ‘iff’ and
- (iii) (A^{ℓ2}) is equivalent to (B^{ℓ2}) with ‘implies’ replaced by ‘iff’.

PROOF. (i) We have to show that:

- (i.1) (A^e) implies (B^e) with ‘implies’ replaced by ‘iff’, and
- (i.2) (B^e) with ‘implies’ replaced by ‘iff’, implies (A^e).

For (i.1) we have that (B^e) with ‘implies’ replaced by ‘iff’, is proved in Corollary 11.15 [6, page 200]. For (i.2) we have to show that:

$$\exists n \in N. t \rightarrow^e n \text{ iff } \exists n \in N. \llbracket t \rrbracket^e \rho = \lfloor n \rfloor \quad (C1)$$

because the canonical forms of type int are the elements of N . Now, Equivalence (C1) is a consequence of Corollary 11.15 [6, page 200] and thus, we get the thesis.

(ii) The proof of this point is analogous that of Point (i), but we have to refer to Corollary 11.24 [6, page 209], instead of Corollary 11.15 [6, page 200].

(iii) We have to show that: (iii.1) (A^{ℓ2}) implies (B^{ℓ2}) with ‘implies’ replaced by ‘iff’, and (iii.2) (B^{ℓ2}) with ‘implies’ replaced by ‘iff’, implies (A^{ℓ2}).

For (iii.1) we have that (B^{ℓ_2}) with ‘implies’ replaced by ‘iff’, is proved in Exercise 11.28 (3) [6, page 217]. For (iii.2) we have to show that:

$$\exists n \in N. t \rightarrow^\ell n \text{ iff } \text{halt}_{int}(\llbracket t \rrbracket^{\ell_2} \rho) = \text{true} \quad (C2)$$

that is, $\exists n \in N. t \rightarrow^\ell n$ iff $\exists n \in N. \llbracket t \rrbracket^{\ell_2} \rho = [n]$, because the canonical forms of type int are the elements of N . Now, Equivalence (C2) is a consequence of Exercise 11.28 (3) [6, page 217] and thus, we get the thesis. \square

We also have the following theorem.

THEOREM 8.6. For some closed, typed terms t_1 and t_2 both of type $int \rightarrow int$,

- (i) $\llbracket t_1 \rrbracket^e \rho = \llbracket t_2 \rrbracket^e \rho$ does not imply $t_1 \rightarrow^e t_2$ and
- (ii) $\llbracket t_1 \rrbracket^{\ell_1} \rho = \llbracket t_2 \rrbracket^{\ell_1} \rho$ does not imply $t_1 \rightarrow^\ell t_2$ and
- (iii) $\llbracket t_1 \rrbracket^{\ell_2} \rho = \llbracket t_2 \rrbracket^{\ell_2} \rho$ does not imply $t_1 \rightarrow^\ell t_2$.

PROOF. (i) Let us consider the terms $\lambda x.x+0$ and $\lambda x.x$. We have that: $\llbracket \lambda x.x+0 \rrbracket^e \rho = \llbracket \lambda x.x \rrbracket^e \rho$. However, since in the eager operational semantics $\lambda x.x+0$ and $\lambda x.x$ are both canonical forms, it is not the case that $\lambda x.x+0 \rightarrow^e \lambda x.x$.

The proofs of (ii) and (iii) are analogous to the proof of (i). \square

As a consequence of this Theorem 8.6, the above Theorem 8.5 cannot be extended to the case where the type of the term t is not int . In particular, (A^e) holds, while (B^e) with ‘implies’ replaced by ‘iff’, does *not* hold (because of Theorem 8.6). Analogously, for ℓ_1 and ℓ_2 , instead of e .

THEOREM 8.7. [Adequacy of the Eager Denotational Semantics] The eager denotational semantics $\llbracket _ \rrbracket^e$ is adequate w.r.t. the eager operational semantics \rightarrow^e .

PROOF. Omitted. \square

THEOREM 8.8. [Adequacy of the Lazy1 Denotational Semantics] The lazy1 denotational semantics $\llbracket _ \rrbracket^{\ell_1}$ is adequate w.r.t. the lazy operational semantics \rightarrow^ℓ .

PROOF. Omitted. \square

As a consequence of Theorems 8.5, 8.7, and 8.8 we have the following corollary.

COROLLARY 8.9. For every closed term t of type int and $n \in N$,

- (i) $t \rightarrow^e n$ iff $\llbracket t \rrbracket^e \rho = \llbracket n \rrbracket^e \rho$ and
- (ii) $t \rightarrow^\ell n$ iff $\llbracket t \rrbracket^{\ell_1} \rho = \llbracket n \rrbracket^{\ell_1} \rho$ and
- (iii) $t \rightarrow^\ell n$ iff $\llbracket t \rrbracket^{\ell_2} \rho = \llbracket n \rrbracket^{\ell_2} \rho$.

Recall that for any n and ρ , we have that: $\llbracket n \rrbracket^e \rho = \llbracket n \rrbracket^{\ell_1} \rho = \llbracket n \rrbracket^{\ell_2} \rho = [n]$.

Let us consider the following two terms of the Lazy language:

- (i) $\mathbf{rec} w.w$, also denoted Ω , where the variable w is of type $int \rightarrow int$ and the term $\mathbf{rec} w.w$ is of type $int \rightarrow int$, and
- (ii) $\lambda x.((\mathbf{rec} w.w) x)$, also denoted $\lambda x.(\Omega x)$, where the variable x is of type int , the variable w is of type $int \rightarrow int$, and the term $\lambda x.((\mathbf{rec} w.w) x)$ is of type $int \rightarrow int$.

We have the following lemma and theorem.

LEMMA 8.10. For any ρ , $\llbracket \Omega \rrbracket^{\ell 2} \rho = \llbracket \lambda x.(\Omega x) \rrbracket^{\ell 2} \rho$ (both sides belong to $[N_{\perp} \rightarrow N_{\perp}]$).

PROOF. This lemma is a consequence of the fact that the η -rule holds in the Lazy2 language (see page 94). We have that for any ρ ,

$$\begin{aligned} \llbracket \Omega \rrbracket^{\ell 2} \rho &= \llbracket \mathbf{rec} w.w \rrbracket^{\ell 2} \rho = \mathit{fix}(\lambda d. \llbracket w \rrbracket^{\ell 2} \rho[d/w]) = \\ &= \mathit{fix}(\lambda d.d), \text{ with } \mathit{fix} \in [[[N_{\perp} \rightarrow N_{\perp}] \rightarrow [N_{\perp} \rightarrow N_{\perp}]] \rightarrow [N_{\perp} \rightarrow N_{\perp}]] = \\ &= \lambda d.\perp, \text{ with } d \in N_{\perp} \text{ and } \perp \in N_{\perp}. \end{aligned}$$

Recall that $\lambda d.\perp$ is the bottom element in $[N_{\perp} \rightarrow N_{\perp}]$. \square

REMARK 8.11. For any ρ , $\llbracket \Omega \rrbracket^{\ell 1} \rho \neq \llbracket \lambda x.(\Omega x) \rrbracket^{\ell 1} \rho$ (both sides belong to $[N_{\perp} \rightarrow N_{\perp}]_{\perp}$). Indeed, for the left hand side we have that:

$$\begin{aligned} \llbracket \Omega \rrbracket^{\ell 1} \rho &= \llbracket \mathbf{rec} w.w \rrbracket^{\ell 1} \rho = \mathit{fix}(\lambda d. \llbracket w \rrbracket^{\ell 1} \rho[d/w]) = \\ &= \mathit{fix}(\lambda d.d), \text{ with } \mathit{fix} \in [[[N_{\perp} \rightarrow N_{\perp}]_{\perp} \rightarrow [N_{\perp} \rightarrow N_{\perp}]_{\perp}] \rightarrow [N_{\perp} \rightarrow N_{\perp}]_{\perp}] = \\ &= \perp, \text{ with } \perp \in [N_{\perp} \rightarrow N_{\perp}]_{\perp}. \end{aligned}$$

For the right hand side we have that:

$$\begin{aligned} \llbracket \lambda x.(\Omega x) \rrbracket^{\ell 1} \rho &= \lfloor \lambda d.(\llbracket \Omega x \rrbracket^{\ell 1} \rho[d/x]) \rfloor \in [N_{\perp} \rightarrow N_{\perp}]_{\perp} \text{ with } d \in N_{\perp} = \\ &= \lfloor \lambda d.(\mathit{let} \varphi \leftarrow \llbracket \Omega \rrbracket^{\ell 1} \rho[d/x] \bullet \varphi(\llbracket x \rrbracket^{\ell 1} \rho[d/x])) \rfloor = \\ &= \{\text{since } \llbracket \Omega \rrbracket^{\ell 1} \rho = \perp \in [N_{\perp} \rightarrow N_{\perp}]_{\perp} \text{ (as we have now shown) we have that} \\ &\quad \varphi = \lambda n.\perp \in [N_{\perp} \rightarrow N_{\perp}]\} = \\ &= \lfloor \lambda d.(\perp(\llbracket x \rrbracket^{\ell 1} \rho[d/x])) \rfloor = \\ &= \lfloor \lambda d.\perp \rfloor \in [N_{\perp} \rightarrow N_{\perp}]_{\perp}. \end{aligned} \quad \square$$

THEOREM 8.12. [**The Lazy2 Denotational Semantics is not adequate**] The lazy2 denotational semantics $\llbracket _ \rrbracket^{\ell 2}$ is *not* adequate w.r.t. the lazy operational semantics \rightarrow^{ℓ} .

PROOF. Let us assume, by absurdum, that the lazy2 denotational semantics $\llbracket _ \rrbracket^{\ell 2}$ is adequate w.r.t. the lazy operational semantics \rightarrow^{ℓ} . In particular, we assume that:

$$\Omega \downarrow^{\ell} \text{ iff } \Omega \Downarrow^{\ell 2} \quad \text{and} \quad (\dagger 1)$$

$$\lambda x.(\Omega x) \downarrow^{\ell} \text{ iff } \lambda x.(\Omega x) \Downarrow^{\ell 2}. \quad (\dagger 2)$$

We have that:

$$\begin{aligned} &\Omega \downarrow^{\ell} && \{\text{by } (\dagger 1)\} \\ \text{iff } &\Omega \Downarrow^{\ell 2} && \{\text{by definition of } \Downarrow^{\ell 2}\} \\ \text{iff } &\mathit{halt}_{int \rightarrow int}(\llbracket \Omega \rrbracket^{\ell 2} \rho) && \{\text{by Lemma 8.10}\} \\ \text{iff } &\mathit{halt}_{int \rightarrow int}(\llbracket \lambda x.(\Omega x) \rrbracket^{\ell 2} \rho) && \{\text{by definition of } \Downarrow^{\ell 2}\} \\ \text{iff } &\lambda x.(\Omega x) \Downarrow^{\ell 2} && \{\text{by } (\dagger 2)\} \\ \text{iff } &\lambda x.(\Omega x) \downarrow^{\ell} \end{aligned}$$

Now, it cannot be the case that: $\Omega \downarrow^{\ell}$ iff $\lambda x.(\Omega x) \downarrow^{\ell}$, because:

(i) $\lambda x.(\Omega x)$ is a lazy canonical form, being an abstraction, and thus, $\lambda x.(\Omega x) \rightarrow \lambda x.(\Omega x)$, and

(ii) no lazy canonical form c exists such that $\Omega \rightarrow^{\ell} c$ because the only way of deducing $\mathbf{rec} w.w \rightarrow^{\ell} c$ is to prove $\mathbf{rec} w.w \rightarrow^{\ell} c$ (see the lazy operational rule for \mathbf{rec}). \square

Now we sum up the main notions and results we have presented in this section.

Let us consider a closed, typed term t . Let t be of type τ . $t : any$ means that the term t is of any type and $t : int$ means that the term t is of type int .

In the Eager, Lazy1, and Lazy2 languages

$t \Downarrow$ stands for there exists a canonical form c such that $t \rightarrow c$.

In the Eager and Lazy1 languages

$t \Downarrow$ stands for $\exists v \in V_\tau. \llbracket t \rrbracket \rho = \lfloor v \rfloor$.

In the Lazy2 language for any $t : int$

$t \Downarrow$ stands for $\exists n \in N_\perp. \llbracket t \rrbracket \rho = \lfloor n \rfloor$.

The following Table 4 sums up the results of this section.

		Eager	Lazy1	Lazy2
• Property A1:	$t : any \quad t \Downarrow \text{ iff } t \Downarrow$	yes	yes	no
Property A1 int :	$t : int \quad t \Downarrow \text{ iff } t \Downarrow$	yes	yes	yes
• Property A2:	$t : any \quad t \rightarrow c \text{ implies } \llbracket t \rrbracket \rho = \llbracket c \rrbracket \rho$	yes	yes	yes
Property A2* int :	$t : int \quad t \rightarrow n \text{ iff } \llbracket t \rrbracket \rho = \lfloor n \rfloor$	yes	yes	yes

TABLE 4. Adequacy of the Eager and Lazy1 languages (that is, Properties A1 and A2 hold). The Lazy2 language is not adequate. In Property A2 ‘implies’ cannot be replaced by ‘iff’ because in the Eager, Lazy1, and Lazy2 languages, $\llbracket \lambda x.x \rrbracket \rho = \llbracket \lambda x.x+0 \rrbracket \rho$ and it is not the case that $\lambda x.x \rightarrow \lambda x.x+0$.

Adequacy of the denotational semantics with respect to the operational semantics is the conjunction of Property A1 and Property A2. Note that in Property A2 the term t is of any type, not necessarily of type int .

Table 4 indicates that: (i) the eager denotational semantics is adequate with respect to the eager operational semantics, (ii) the lazy1 denotational semantics is adequate with respect to the lazy operational semantics, while (iii) the lazy2 denotational semantics is *not* adequate with respect to the lazy operational semantics. Indeed, in the case of the lazy2 denotational semantics Property A1 holds only for terms t of type int , and not for terms of any type.

9. Half and Full Abstraction

Let us consider a generic, higher order operational semantics relation, denoted $_ \rightarrow _$, and a generic, higher order denotational semantics function, denoted $\llbracket _ \rrbracket \rho$. For every closed, typed term t , we write $t \Downarrow$ iff there exists a canonical form c such that $t \rightarrow c$.

Let us also consider the following two formulas, which are assumed to have as parameters two terms t_1 and t_2 with the same type:

$Op(t_1, t_2)$ which holds iff for every context $C[-]$ such that $C[t_1]$ and $C[t_2]$ are typable, closed terms, $C[t_1] \downarrow$ iff $C[t_2] \downarrow$

$Den(t_1, t_2)$ which holds iff for every environment ρ , $\llbracket t_1 \rrbracket \rho = \llbracket t_2 \rrbracket \rho$.

The name of the function Op derives from the fact that its definition refers to the *operational semantics*. Analogously, the name of the function Den derives from the fact that its definition refers to the *denotational semantics*.

The following definitions relate the convergence of the operational semantics to the equality of the denotational semantics. These definitions are meaningful because through contexts we can establish equality of denotational values. For instance, in the Eager language for any term t of type int , for any $n \in N$, we have that:

$$\begin{aligned} t \rightarrow^e n \text{ iff } & \mathbf{if} (t - n) \mathbf{then} 0 \mathbf{else} \Omega \rightarrow n \\ & \text{iff } \llbracket t \rrbracket^e \rho = \lfloor n \rfloor. \end{aligned}$$

Analogous properties hold for the Lazy1 and Lazy2 languages.

DEFINITION 9.1. [Half Abstraction of a Denotational Semantics with respect to an Operational Semantics] A denotational semantics $\llbracket - \rrbracket$ is said to be *half abstract* w.r.t. the observation of convergence of the operational semantics \rightarrow (or simply, w.r.t. the operational semantics \rightarrow) if

$$\text{for all terms } t_1 \text{ and } t_2, \quad Op(t_1, t_2) \text{ if } Den(t_1, t_2).$$

DEFINITION 9.2. [Full Abstraction of a Denotational Semantics with respect to an Operational Semantics] A denotational semantics $\llbracket - \rrbracket$ is said to be *fully abstract* w.r.t. the observation of convergence of the operational semantics \rightarrow (or simply, w.r.t. the operational semantics \rightarrow) if

$$\text{for all terms } t_1 \text{ and } t_2, \quad Op(t_1, t_2) \text{ if and only if } Den(t_1, t_2).$$

THEOREM 9.3. [The Eager Denotational Semantics is Half Abstract] For the eager semantics $\llbracket - \rrbracket^e$ which is adequate w.r.t. \rightarrow^e , we have that for all terms t_1 and t_2 of the same type, $Op(t_1, t_2)$ if $Den(t_1, t_2)$.

PROOF. Let us consider the terms t_1 and t_2 of type τ . If $\llbracket t_1 \rrbracket^e \rho = \llbracket t_2 \rrbracket^e \rho$ we get that: for every context $C[-]$, $\llbracket C[t_1] \rrbracket^e \rho = \llbracket C[t_2] \rrbracket^e \rho$. Now there are two cases.

Case (i): $\llbracket C[t_1] \rrbracket^e \rho \neq \perp \in (V_\tau)_\perp$ and Case (ii): $\llbracket C[t_1] \rrbracket^e \rho = \perp \in (V_\tau)_\perp$.

In Case (i), from $\llbracket C[t_1] \rrbracket^e \rho = \llbracket C[t_2] \rrbracket^e \rho \neq \perp$, by adequacy, we get that: $C[t_1] \downarrow$ and $C[t_2] \downarrow$.

In Case (ii), from $\llbracket C[t_1] \rrbracket^e \rho = \llbracket C[t_2] \rrbracket^e \rho = \perp$, by adequacy, we get that: $\neg C[t_1] \downarrow$ and $\neg C[t_2] \downarrow$.

Thus, in both cases we have $C[t_1] \downarrow$ iff $C[t_2] \downarrow$. □

THEOREM 9.4. [The Lazy1 Denotational Semantics is Half Abstract] For the lazy1 semantics $\llbracket - \rrbracket^{\ell_1}$ which is adequate w.r.t. \rightarrow^ℓ , we have that for all terms t_1 and t_2 of the same type, $Op(t_1, t_2)$ if $Den(t_1, t_2)$.

PROOF. Similar to the proof of Theorem 9.3. □

THEOREM 9.5. [The Lazy2 Denotational Semantics is not Half Abstract] For the lazy2 semantics $\llbracket - \rrbracket^{\ell_2}$ which is *not* adequate w.r.t. \rightarrow^ℓ , it is *not* the case that for all terms t_1 and t_2 of the same type, $Op(t_1, t_2)$ if $Den(t_1, t_2)$.

PROOF. Let us consider the terms Ω and $\lambda x.(\Omega x)$ both of type $int \rightarrow int$. We have that:

(i) $\llbracket \Omega \rrbracket^{\ell 2} \rho = \llbracket \lambda x.(\Omega x) \rrbracket^{\ell 2} \rho$,

(ii) $\lambda x.(\Omega x) \rightarrow^{\ell} \lambda x.(\Omega x)$, and

(iii) no lazy operational canonical form c exists such that $\Omega \rightarrow^{\ell} c$.

Thus, by Point (i), $Den(\Omega, \lambda x.(\Omega x))$ holds, and by Point (ii), it is not the case that $\Omega \downarrow^{\ell}$ iff $\lambda x.(\Omega x) \downarrow^{\ell}$, and thus, $Op(\Omega, \lambda x.(\Omega x))$ does *not* hold because if we take the context $C[_]$ to be empty context, it is not the case that $C[\Omega] \downarrow^{\ell}$ iff $C[\lambda x.(\Omega x)] \downarrow^{\ell}$. \square

THEOREM 9.6. [The Eager Denotational Semantics is Not Fully Abstract]
The eager denotational semantics $\llbracket _ \rrbracket^e$ is *not* fully abstract w.r.t. the eager operational semantics \rightarrow^e .

PROOF. Omitted. \square

THEOREM 9.7. [The Lazy1 Denotational Semantics is Not Fully Abstract]
The lazy1 denotational semantics $\llbracket _ \rrbracket^{\ell 1}$ is *not* fully abstract w.r.t. the lazy operational semantics \rightarrow^{ℓ} .

PROOF. Omitted. \square

THEOREM 9.8. [The Lazy2 Denotational Semantics is Not Fully Abstract]
The lazy2 denotational semantics $\llbracket _ \rrbracket^{\ell 2}$ is *not* fully abstract w.r.t. the lazy operational semantics \rightarrow^{ℓ} .

PROOF. It is a consequence of Theorem 9.5. \square

Tables 5 and 6 on the facing page summarize the results of Sections 5, 6, 8, and 9. ‘yes’ means that the rule (or the property) holds, while ‘no’ means that the rule (or the property) does not hold.

operational semantics	α -rule	β -rule	η -rule
Eager	no	no	no
Lazy	no	yes	no

TABLE 5. Validity of the α -rule, β -rule, and η -rule in the eager and lazy operational semantics.

denotational semantics	α -rule	β -rule	η -rule	$A1int$ & $A2*int$	adequacy $A1$ & $A2$	abstraction half	full
Eager	yes	no	no	yes	yes	yes	no
Lazy1	yes	yes	no	yes	yes	yes	no
Lazy2	yes	yes	yes	yes	no	no	no

TABLE 6. Validity of the α -rule, β -rule, η -rule, adequacy, half abstraction, and full abstraction in the eager, lazy1, and lazy2 denotational semantics. Properties $A1$, $A2$, $A1int$, and $A2*int$ are defined in Table 4 on page 106. Half abstraction and full abstraction are defined in Definition 9.1 on page 107 and Definition 9.2 on page 107, respectively.