

# EXERCISES

## CHAPTER 2

SEAN LI <sup>1</sup>

### 1. Reducted

#### Problem

Type the following terms

$xyx \quad xyy \quad xyx \quad x(xy) \quad x(yx)$

*Solution.* The first term cannot be typed.

*Proof.*  $xyx = (xx)y$ . Therefore,  $x$  is a function type, denote it as  $\tau \rightarrow \sigma$ . By the application rule, a subterm applied to  $x$  must be of  $\tau$ , which means that the application  $xx$  is not legally typed. ■

The second one is typable where  $x : \tau \rightarrow \tau \rightarrow \sigma$  and  $y : \tau$ .

<i>a</i>	1.	$x : \tau \rightarrow \tau \rightarrow \sigma$	<b>ctx</b>
<i>b</i>	2.	$y : \tau$	<b>ctx</b>
1	3.	$xy : \tau \rightarrow \sigma$	<b>T-App (a)</b>
<i>c</i>	4.	$xyy : \sigma$	<b>T-App (1)</b>

The third term is not typable.

*Proof.* Assume  $xyx = (xy)x$  is typable. Therefore,  $x : \tau$  where  $\tau = \sigma \rightarrow \tau \rightarrow \alpha$  and  $y : \sigma$ . One can construct an infinite chain of function type by substituting  $\tau$ :  $\tau = \sigma \rightarrow (\sigma \rightarrow (\sigma \rightarrow \dots \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$ . By induction, it can be proven that only lambda abstractions can construct function types, meaning that the term is of form

$$(\lambda n : \tau. \lambda m : \tau. \dots (\lambda a : \sigma. \lambda b : \sigma. \dots))y (\lambda n : \tau. \lambda m : \tau. \dots (\lambda a : \sigma. \lambda b : \sigma. \dots))$$

meaning that an infinite reduction path is needed. This is impossible in STLC. ■

The fourth type is typable where  $x : (\tau \rightarrow \tau)$  and  $y : \tau$ .

$a$  1.  $x : \tau \rightarrow \tau$  **ctx**  
 $b$  2.  $y : \tau$  **ctx**  
1 3.  $\frac{}{xy : \tau}$  **T-App (b) on (a)**  
2 4.  $x(xy) : \tau$  **T-App (1) on (a)**

The fifth term is typable where  $x : (\tau \rightarrow \sigma)$  and  $y : (\tau \rightarrow \sigma) \rightarrow \tau$ :

$a$  1.  $x : \tau \rightarrow \sigma$  **ctx**  
 $b$  2.  $y : (\tau \rightarrow \sigma) \rightarrow \tau$  **ctx**  
1 3.  $\frac{}{yx : \tau}$  **T-App (a) on (b)**  
2 4.  $x(yx) : \sigma$  **T-App (1) on (a)**

### Problem

Find types for zero, one, and two

*Solution.* Term for zero is

$$\text{zero} := \lambda f x. x$$

Here  $x$  is only used as a

$$\text{zero} := \lambda f : \alpha. \lambda x : \beta. x$$

Type derivation shown as below:

$a$  1.  $f : \alpha$  **bind**  
 $b$  2.  $\frac{}{x : \beta}$  **bind**  
1 3.  $\frac{}{x : \beta}$  **T-Var (b)**  
2 4.  $\frac{}{\lambda x. x : \beta \rightarrow \beta}$  **T-Abst (1)**  
3 5.  $\lambda f : \alpha. x : \beta. x : \alpha \rightarrow \beta \rightarrow \beta$  **T-Abst (2)**

Term for one is

$$\text{one} := \lambda f x. f x$$

Let  $f$  be an arbitrary function type that consumes  $x$

$$\text{one} := \lambda f : \alpha \rightarrow \beta. x : \alpha. f x$$

Type derivation shown as below

<i>a</i>	1.	$f : \alpha \rightarrow \beta$	<b>bind</b>
<i>b</i>	2.	$x : \alpha$	<b>bind</b>
1	3.	$fx : \beta$	<b>T-App (b) on a</b>
2	4.	$\lambda x. fx : \alpha \rightarrow \beta$	<b>T-Abst (1)</b>
3	5.	$\lambda f : \alpha \rightarrow \beta. x : \alpha. fx : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$	<b>T-Abst (2)</b>

Same type signatures can be given to two

$$\text{two} := \lambda f : \alpha \rightarrow \beta. \lambda x : \alpha. ffx$$

Type derivation shown as below

<i>a</i>	1.	$f : \alpha \rightarrow \beta$	<b>bind</b>
<i>b</i>	2.	$x : \alpha$	<b>bind</b>
1	3.	$fx : \beta$	<b>T-App (b) on (a)</b>
2	4.	$ffx : \beta$	<b>T-App (2) on (b)</b>
3	5.	$\lambda x. ffx : \alpha \rightarrow \beta$	<b>T-Abst (2)</b>
4	6.	$\lambda f : \alpha \rightarrow \beta. x : \alpha. ffx : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$	<b>T-Abst (3)</b>

### Problem

Find types for

$$K := \lambda xy. x$$

$$S := \lambda xyz. xz(yz)$$

*Solution.* There are no occurrences of application in  $K$ 's subterms. Therefore all its binding variables could be given a simple base type.

$$K := \lambda x : \alpha. \lambda y : \beta. x$$

Type derivation shown as below

<i>a</i>	1.	$x : \alpha$	<b>bind</b>
<i>b</i>	2.	$y : \beta$	<b>bind</b>
1	3.	$x : \alpha$	<b>T-Var</b>
2	4.	$\lambda y : \beta. x : \beta \rightarrow \alpha$	<b>T-Abst on (1)</b>
3	5.	$\lambda x : \alpha. \lambda y : \beta. x : \alpha \rightarrow \beta \rightarrow \alpha$	<b>T-Abst on (2)</b>

For the  $S$  combinator, no term was applied to  $z$ . Therefore it can be given a simple base type  $\alpha$ . As  $z$  was applied to  $y$ , it implies that  $y : \alpha \rightarrow \beta$  for some output type  $\beta$ . As  $x$  takes  $z$  and  $(yz)$ , it must be of type  $\alpha \rightarrow \beta \rightarrow \delta$ .

$$S := \lambda x : \alpha \rightarrow \beta \rightarrow \delta. \lambda y : \alpha \rightarrow \beta. \lambda z. \alpha. xz(yz)$$

Complete type derivation shown as below:

$$a \quad 1. \quad x : \alpha \rightarrow \beta \rightarrow \delta$$

**bind**

$$b \quad 2. \quad \left| y : \alpha \rightarrow \beta \right.$$

**bind**

$$c \quad 3. \quad \left| \left| z : \alpha \right. \right.$$

**bind**

$$1 \quad 4. \quad \left| \left| \left| yz : \beta \right. \right. \right.$$

**T-App (c) on (b)**

$$2 \quad 5. \quad \left| \left| \left| xz : \beta \rightarrow \delta \right. \right. \right.$$

**T-App (c) on (a)**

$$3 \quad 6. \quad \left| \left| \left| \underline{xz(yz) : \delta} \right. \right. \right.$$

**T-App (1) on (2)**

$$4 \quad 7. \quad \left| \left| \left| \underline{\lambda z : \alpha. xz(yz) : \alpha \rightarrow \delta} \right. \right. \right.$$

**T-Abstr on (3)**

$$5 \quad 8. \quad \left| \left| \left| \underline{\lambda y : \alpha \rightarrow \beta. \lambda z. \alpha. xz(yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta} \right. \right. \right.$$

**T-Abstr on (4)**

$$6 \quad 9. \quad \lambda x : \alpha \rightarrow \beta \rightarrow \delta. \lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. xz(yz) : (\alpha \rightarrow \beta \rightarrow \delta) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta$$

**T-Abstr on (5)**

### Problem

Type the bound variables

$$\lambda xyz. x(yz)$$

$$\lambda xyz. y(xz)z$$

*Solution.* For the first term,  $z$  had nothing applied to it. Therefore it could be given a simple base type  $\alpha$ .  $z$  was applied to  $y$ , therefore  $y : \alpha \rightarrow \beta$  to satisfy the application rule. Because the application yielded a type of  $\beta$ , by the application rule  $x : \beta \rightarrow \delta$  for some type  $\delta$ .

$$\lambda x : \beta \rightarrow \delta. \lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. x(yz)$$

Complete type derivation shown below

$$a \quad 1. \quad x : \beta \rightarrow \delta$$

**bind**

$$b \quad 2. \quad \left| y : \alpha \rightarrow \beta \right.$$

**bind**

$$c \quad 3. \quad \left| \left| z : \alpha \right. \right.$$

**bind**

$$1 \quad 4. \quad \left| \left| \left| yz : \beta \right. \right. \right.$$

**T-App (c) on (b)**

$$2 \quad 5. \quad \left| \left| \left| x(yz) : \delta \right. \right. \right.$$

**T-App (1) on (a)**

$$3 \quad 6. \quad \left| \left| \lambda z : \alpha. x(yz) : \alpha \rightarrow \delta \right. \right.$$

**T-Abst on (2)**

$$4 \quad 7. \quad \left| \lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. x(yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta \right.$$

**T-Abst on (3)**

$$5 \quad 8. \quad \lambda x : \beta \rightarrow \delta. \lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. x(yz) : (\beta \rightarrow \delta) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta$$

**T-Abst on (4)**

In the second term  $z$  could still be given a simple base type  $z : \alpha$ . Therefore  $x : \alpha \rightarrow \beta$  for some type  $\beta$ .  $y$  takes  $xz : \beta$  and  $z : \alpha$ , therefore it is of type  $y : \beta \rightarrow \alpha \rightarrow \delta$  for some  $\delta$ .

$$\lambda x : \alpha \rightarrow \beta. \lambda y : \beta \rightarrow \alpha \rightarrow \delta. \lambda z : \alpha. y(xz)z$$

. Complete type derivation shown below

$$a \quad 1. \quad x : \alpha \rightarrow \beta$$

**bind**

$$b \quad 2. \quad \left| y : \beta \rightarrow \alpha \rightarrow \delta \right.$$

**bind**

$$c \quad 3. \quad \left| \left| z : \alpha \right. \right.$$

**bind**

$$1 \quad 4. \quad \left| \left| \left| xz : \beta \right. \right. \right.$$

**T-App (c) on (b)**

2 5.  $\left| \left| \left| y(xz) : \alpha \rightarrow \delta \right. \right. \right|$

**T-App (1) on (b)**

3 6.  $\left| \left| \left| \underline{y(xz)z : \delta} \right. \right. \right|$

**T-App (c) on (2)**

4 7.  $\left| \left| \left| \underline{\lambda z : \alpha. y(xz)z : \alpha \rightarrow \delta} \right. \right. \right|$

**T-Abst on (3)**

5 8.  $\left| \left| \left| \underline{\lambda y : \beta \rightarrow \alpha \rightarrow \delta. \lambda z. y(xz)z : (\beta \rightarrow \alpha \rightarrow \delta) \rightarrow \alpha \rightarrow \delta} \right. \right. \right|$

**T-Abst on (4)**

6 9.  $\lambda x : \alpha \rightarrow \beta. \lambda y : \beta \rightarrow \alpha \rightarrow \delta. \lambda z. y(xz)z : (\alpha \rightarrow \beta) \rightarrow (\beta \rightarrow \alpha \rightarrow \delta) \rightarrow \alpha \rightarrow \delta$

**T-Abst on (5)**

### Problem

Try to type the following terms, and prove if not typable.

$\lambda xy. x(\lambda z. y)y$

$\lambda xy. x(\lambda z. x)y.$

*Solution.* The first term is trivially typable.

a	1.	$x : (\delta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta$	<b>bind</b>
b	2.	$y : \alpha$	<b>bind</b>
1	3.	$x : (\delta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta$	<b>T-Var</b>
c	4.	$z : \delta$	<b>bind</b>
2	5.	$\left  \underline{y : \alpha} \right $	<b>T-Var</b>
3	6.	$\lambda z : \delta. y : \delta \rightarrow \alpha$	<b>T-Abst on (1)</b>
4	7.	$x(\lambda z : \delta. y) : \alpha \rightarrow \beta$	<b>T-App (3) on (1)</b>
5	8.	$\left  \underline{x(\lambda z : \delta. y)y : \beta} \right $	<b>T-App (b) on (4)</b>
6	9.	$\left  \underline{\lambda y : \alpha. x(\lambda z : \delta. y)y : \alpha \rightarrow \beta} \right $	<b>T-Abst on (5)</b>
7	10.		

$\lambda x : ((\delta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta) \lambda y : \alpha. x(\lambda z : \delta. y)y$

$: ((\delta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$

**T-Abst on (5)**

The second term is not typable in STLC.

*Proof.* By induction on the type inference rule that constructed the type judgement for subterm  $x(\lambda z.x)$ . Because the term is an application, the only rule that applies is the application rule.

We denote the context inside the abstraction as  $\Gamma'$ . Suppose  $\mathcal{J} \equiv \Gamma' \vdash x(\lambda z.x) : \tau$ . By the inference rule of application,  $x$  must be a function type that accepts the type of  $(\lambda z.x)$ . Let  $\Gamma' \vdash z : \alpha$ , and type of  $x$  as  $\tau_x$ . Therefore,  $\Gamma' \vdash \lambda z : \alpha. x : \alpha \rightarrow \tau_x$ . Therefore,  $\tau_x = (\alpha \rightarrow \tau_x) \rightarrow \tau$ . This is a recursive type, which is not constructable as it requires infinitely nested lambda abstractions that requires infinite reduction paths to reach a normal form. ■

### Problem

Prove the pretyped term below is legal.

$$\lambda x : ((\alpha \rightarrow \beta) \rightarrow \alpha). x(\lambda z : \alpha. y)$$

Using the tree format and the flag format.

*Solution.* We suppose a context  $\Gamma \vdash y : \beta$  that obviously exists.

*Proof.*

$$\frac{\frac{}{x : (\alpha \rightarrow \beta) \rightarrow \alpha} \text{(Bound)} \quad \frac{\Gamma, z : \alpha \vdash y : \beta}{\Gamma \vdash (\lambda z : \alpha. y) : \alpha \rightarrow \beta} \text{(T-Abst)}}{\Gamma, x : (\alpha \rightarrow \beta) \rightarrow \alpha \vdash (x(\lambda z : \alpha. y)) : \alpha} \text{(T-App)} \quad \frac{}{\Gamma \vdash \lambda x : ((\alpha \rightarrow \beta) \rightarrow \alpha). x(\lambda z : \alpha. y) : ((\alpha \rightarrow \beta) \rightarrow \beta) \rightarrow \alpha} \text{(T-Abst)}$$

A valid type could be given to the term. Therefore, the term is typable under an existing context. ■

The flag derivation is given below:

(a)	1.	$x : (\alpha \rightarrow \beta) \rightarrow \alpha$	<b>Bound</b>
	(b)	2.	<b>Bound</b>
		3.	$\vdash \Gamma$
	(1)	3.	$\vdash \Gamma$
	(2)	4.	<b>T-Abst on (1)</b>
	(3)	5.	<b>T-App (2) on (a)</b>
	(4)	6.	

$$\lambda x : ((\alpha \rightarrow \beta) \rightarrow \beta). x(\lambda z : \alpha. y)$$

$$: (\alpha \rightarrow \beta) \rightarrow \beta \rightarrow \alpha \quad \textbf{T-Abst on (3)}$$

### Problem

Derive

$$f : A \rightarrow B \wedge g : B \rightarrow C \vdash g \circ f : A \rightarrow C$$

Using the rules

$$\frac{f : A \rightarrow B, c \in A}{f(c) \in B} \text{(F-App)}$$

$$\frac{\forall x \in A, f(x) \in B}{f : A \rightarrow B} \text{(F-Abst)}$$