

EXERCISES

CHAPTER 2

SEAN LI ¹

1. Reducted

Problem

Type the following terms

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

Solution. The first term cannot be typed.

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

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

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

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 τ : $\tau = \sigma \rightarrow (\sigma \rightarrow (\sigma \rightarrow \dots \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$. 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 $y : \tau$.

<i>a</i>	1.	$x : \tau \rightarrow \tau$	ctx
<i>b</i>	2.	$y : \tau$	ctx
1	3.	$\boxed{xy : \tau}$	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 $y : (\tau \rightarrow \sigma) \rightarrow \tau$:

<i>a</i>	1.	$x : \tau \rightarrow \sigma$	ctx
<i>b</i>	2.	$y : (\tau \rightarrow \sigma) \rightarrow \tau$	ctx
1	3.	$\boxed{yx : \tau}$	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:

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

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

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

For the S combinator, no term was applied to z . Therefore it can be given a simple base type α . As z was applied to y , it implies that $y : \alpha \rightarrow \beta$ for some output type β . 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:

$$\begin{array}{ll}
a & 1. \quad x : \alpha \rightarrow \beta \rightarrow \delta \\
\textbf{bind} & \\
b & 2. \quad | \quad y : \alpha \rightarrow \beta \\
\textbf{bind} & \\
c & 3. \quad | \quad | \quad z : \alpha \\
\textbf{bind} & \\
1 & 4. \quad | \quad | \quad | \quad yz : \beta \\
\textbf{T-App (c) on (b)} & \\
2 & 5. \quad | \quad | \quad | \quad xz : \beta \rightarrow \delta \\
\textbf{T-App (c) on (a)} & \\
3 & 6. \quad | \quad | \quad | \quad \underline{xz(yz) : \delta} \\
\textbf{T-App (1) on (2)} & \\
4 & 7. \quad | \quad | \quad \underline{\lambda z : \alpha. xz(yz) : \alpha \rightarrow \delta} \\
\textbf{T-Abstr on (3)} & \\
5 & 8. \quad | \quad | \quad \underline{\lambda y : \alpha \rightarrow \beta. \lambda z. \alpha. xz(yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta} \\
\textbf{T-Abstr on (4)} & \\
6 & 9. \quad \lambda x : \alpha \rightarrow \beta \rightarrow \delta. \lambda y : \alpha \rightarrow \beta. \lambda z. \alpha. xz(yz) : (\alpha \rightarrow \beta \rightarrow \delta) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta \\
\textbf{T-Abstr on (5)} &
\end{array}$$

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 α . z was applied to y , therefore $y : \alpha \rightarrow \beta$ to satisfy the application rule. Because the application yielded a type of β , by the application rule $x : \beta \rightarrow \delta$ for some type δ .

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

Complete type derivation shown below

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

bind

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

bind

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

bind

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

T-App (c) on (b)

$$2 \quad 5. \quad | \quad | \quad | \quad \boxed{x(yz) : \delta}$$

T-App (1) on (a)

$$3 \quad 6. \quad | \quad | \quad | \quad \boxed{\lambda z : \alpha. x(yz) : \alpha \rightarrow \delta}$$

T-Abst on (2)

$$4 \quad 7. \quad | \quad | \quad | \quad \boxed{\lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. x(yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \delta}$$

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 β . y takes $xz : \beta$ and $z : \alpha$, therefore it is of type $y : \beta \rightarrow \alpha \rightarrow \delta$ for some δ .

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

. Complete type derivation shown below

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

bind

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

bind

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

bind

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

T-App (c) on (b)

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

T-App (1) on (b)

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

T-App (c) on (2)

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

T-Abst on (3)

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

T-Abst on (4)

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

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

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

$$: ((\delta \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta \quad \text{**T-Abst on (5)**}$$

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 Γ' . 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 τ_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.

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

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.	$x : (\alpha \rightarrow \beta) \rightarrow \alpha$	Bound
(b)	2.	$z : \alpha$	Bound
(1)	3.	$y : \beta$	$\dashv \Gamma$
(2)	4.	$(\lambda z : \alpha.y) : \alpha \rightarrow \beta$	T-Abst on (1)
(3)	5.	$x(\lambda z : \alpha.y) : \beta$	T-App (2) on (a)
(4)	6.		

$$\begin{aligned} \lambda x : ((\alpha \rightarrow \beta) \rightarrow \beta).x(\lambda z : \alpha.y) \\ : (\alpha \rightarrow \beta) \rightarrow \beta \rightarrow \alpha \end{aligned} \quad \text{■}$$

Problem

Derive

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

Using the rules

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