# Excercises

## 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 $\tau \to \sigma$. 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 $x : \tau \to \tau \to \sigma$ and $y : \tau$.

| | | | |
|---|---|---|---|
| $a$ | 1. | $x : \tau \to \tau \to \sigma$ | **ctx** |
| $b$ | 2. | $y : \tau$ | **ctx** |
| $1$ | 3. | $xy : \tau \to \sigma$ | **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 $\tau = \sigma \to \tau \to \alpha$ and $y : \sigma$. One can construct an infinite chain of function type by substituting $\tau$: $\tau = \sigma \to (\sigma \to (\sigma \to \dots \to \alpha) \to \alpha) \to \alpha$. 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 $x : (\tau \to \tau)$ and $y : \tau$.

|   |    |              |                   |
|---|----|--------------|-------------------|
| a | 1. | $x : \tau \to \tau$ | **ctx**    |
| b | 2. | $y : \tau$   | **ctx**           |
| 1 | 3. | $\quad xy : \tau$ | **T-App (b) on (a)** |
| 2 | 4. | $x(xy) : \tau$ | **T-App (1) on (a)** |

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

|   |    |              |                   |
|---|----|--------------|-------------------|
| a | 1. | $x : \tau \to \sigma$ | **ctx**    |
| b | 2. | $y : (\tau \to \sigma) \to \tau$ | **ctx** |
| 1 | 3. | $\quad yx : \tau$ | **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

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

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. | $x : \beta$            | **bind**     |
| 1 | 3. | $\quad x : \beta$      | **T-Var (b)**  |
| 2 | 4. | $\lambda x.x : \beta \to \beta$ | **T-Abst (1)** |
| 3 | 5. | $\lambda f : \alpha.x : \beta.x : \alpha \to \beta \to \beta$ | **T-Abst (2)** |

Term for one is

$$\text{one} := \lambda fx.fx$$

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

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

Type derivation shown as below

| | | | |
|---|---|---|---|
| a | 1. | $f : \alpha \to \beta$ | **bind** |
| b | 2. | $x : \alpha$ | **bind** |
| 1 | 3. | $fx : \beta$ | **T-App (b) on a** |
| 2 | 4. | $\lambda x.fx : \alpha \to \beta$ | **T-Abst (1)** |
| 3 | 5. | $\lambda f : \alpha \to \beta.x : \alpha.fx : (\alpha \to \beta) \to \alpha \to \beta$ | **T-Abst (2)** |

Same type signatures can be given to two

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

Type derivation shown as below

| | | | |
|---|---|---|---|
| a | 1. | $f : \alpha \to \beta$ | **bind** |
| b | 2. | $x : \alpha$ | **bind** |
| 1 | 3. | $fx : \beta$ | **T-App (b) on (a)** |
| 2 | 4. | $ffx : \beta$ | **T-App (2) on (b)** |
| 3 | 5. | $\lambda x.ffx : \alpha \to \beta$ | **T-Abst (2)** |
| 4 | 6. | $\lambda f : \alpha \to \beta.x : \alpha.ffx : (\alpha \to \beta) \to \alpha \to \beta$ | **T-Abst (3)** |

## Problem

Find types for

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

*Solution.* There are no occurences 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

| | | | |
|---|---|---|---|
| a | 1. | $x : \alpha$ | **bind** |
| b | 2. | $y : \beta$ | **bind** |
| 1 | 3. | $x : \alpha$ | **T-Var** |
| 2 | 4. | $\lambda y : \beta.x : \beta \to \alpha$ | **T-Abst on (1)** |
| 3 | 5. | $\lambda x : \alpha.\lambda y : \beta.x : \alpha \to \beta \to \alpha$ | **T-Abst on (2)** |

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 $y : \alpha \to \beta$ for some output type $\beta$. As $x$ takes $z$ and $(yz)$, it must be of type $\alpha \to \beta \to \delta$.

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

Complete type derivation shown as below:

$a$    1.    $x : \alpha \to \beta \to \delta$

**bind**

$b$    2.    $\bigg| \ y : \alpha \to \beta$

**bind**

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

**bind**

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

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

2    5.    $\bigg| \ \bigg| \ \bigg| \ xz : \beta \to \delta$

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

3    6.    $\bigg| \ \bigg| \ \underline{xz(yz) : \delta}$

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

4    7.    $\bigg| \ \bigg| \ \underline{\lambda z : \alpha.xz(yz) : \alpha \to \delta}$

**T-Abstr on (3)**

5    8.    $\bigg| \ \underline{\lambda y : \alpha \to \beta.\lambda z.\alpha.xz(yz) : (\alpha \to \beta) \to \alpha \to \delta}$

**T-Abstr on (4)**

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

**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 $y : \alpha \to \beta$ to satisfy the application rule. Because the application yielded a type of $\beta$, by the application rule $x : \beta \to \delta$ for some type $\delta$.

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

Complete type derivation shown below

*a*   1.   $x : \beta \to \delta$

**bind**

*b*   2.   $\quad | \; y : \alpha \to \beta$

**bind**

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

**bind**

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

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

*2*   5.   $\quad | \; | \; | \; x(yz) : \delta$

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

*3*   6.   $\quad | \; | \; \lambda z : \alpha.x(yz) : \alpha \to \delta$

**T-Abst on (2)**

*4*   7.   $\quad | \; \lambda y : \alpha \to \beta.\lambda z : \alpha.x(yz) : (\alpha \to \beta) \to \alpha \to \delta$

**T-Abst on (3)**

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

**T-Abst on (4)**

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

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

. Complete type derivation shown below

*a*   1.   $x : \alpha \to \beta$

**bind**

*b*   2.   $\quad | \; y : \beta \to \alpha \to \delta$

**bind**

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

**bind**

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

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

*2*　5.　$\Big|\ \Big|\ \Big| y(xz) : \alpha \to \delta$

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

*3*　6.　$\Big|\ \Big|\ \Big| y(xz)z : \delta$

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

*4*　7.　$\Big|\ \Big| \lambda z : \alpha.y(xz)z : \alpha \to \delta$

**T-Abst on (3)**

*5*　8.　$\Big| \lambda y : \beta \to \alpha \to \delta.\lambda z.y(xz)z : (\beta \to \alpha \to \delta) \to \alpha \to \delta$

**T-Abst on (4)**

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

**T-Abst on (5)**

---

<div style="background:teal">

**Problem**

</div>

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

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

$\quad : ((\delta \to \alpha) \to \alpha \to \beta) \to \alpha \to \beta$　　**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, $\Gamma' \vdash \lambda z : \alpha.x : \alpha \to \tau_x$. Therefore, $\tau_x = (\alpha \to \tau_x) \to \tau$. 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.

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

Using the tree format and the flag format.

---

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

*Proof.*

$$\cfrac{\cfrac{}{x : (\alpha \to \beta) \to \alpha}\text{(Bound)} \qquad \cfrac{\cfrac{\Gamma, z : \alpha \vdash y : \beta}{\Gamma \vdash (\lambda z : \alpha.y) : \alpha \to \beta}\text{(T-Abst)}}{\Gamma, x : (\alpha \to \beta) \to \alpha \vdash (x(\lambda z : \alpha.y)) : \alpha}\text{(T-App)}}{\Gamma \vdash \lambda x : ((\alpha \to \beta) \to \alpha).x(\lambda z : \alpha.y) : ((\alpha \to \beta) \to \beta) \to \alpha}\text{(T-Abst)}$$

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

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

$\qquad : (\alpha \to \beta) \to \beta \to \alpha$      **T-Abst on (3)**

## Problem

Derive

$$f : A \to B \land g : B \to C \vdash g \circ f : A \to C$$

Using the rules

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