

Type checker for System F

Mudathir Mahgoub

May 3, 2018

1 Project Problem

This project is a type checker for annotated simply typed lambda calculus and system F. It supports subtyping for simple types. In general, a type checker would decide whether $\Gamma \vdash t : T$ is derivable: can the term t be assigned the type T under the typing context Γ ?

Section 2 describes the software and its architecture. Section 3 describes the rules used for simple types and provides some examples. It also explains how subsumption rule can be delayed until variable terms are generated, and provides a proof for correctness. Section 3 describes the rules used for system F and provides some examples.

2 Software description

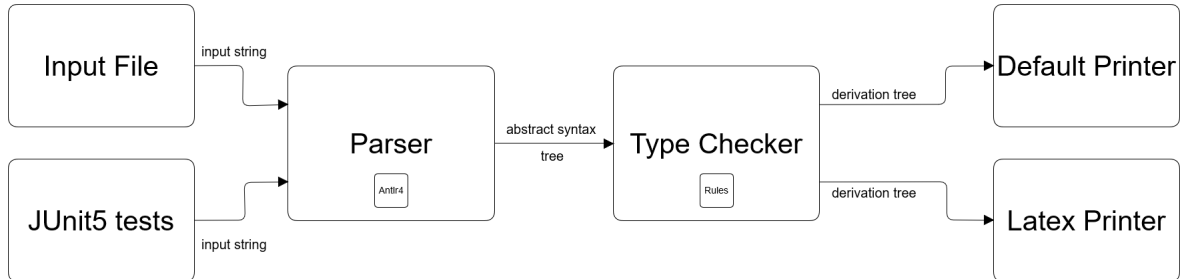


Figure 1: Project architecture.

The project is implemented using Java and the executable is a jar file (target/TypeChecker.jar) which is generated using the command:

```
mvn install
```

The program receives as an input a text file containing subtypes definitions and a judgment to be checked. For testing, **JUnit5** was used to test the program directly without files. The input is passed to the **Parser** which uses the ANTLR library to parse the input into an abstract syntax tree. This abstract syntax tree is consumed by the **Type Checker** which uses the rules in sections 3 and 4 to build a derivation tree and determine the answer of the type checking. The answer can be **Yes**, **No**, or **Unknown**. Finally the derivation tree can be printed using the **Default Printer** or the **Latex Printer**.

Here is an example of an input:

Listing 1: test.txt

```
SubBase( bool , int );
. |- \lambda x. \lambda y. (x y)[bool]: (int ->T) -> (bool -> T);
```

Here is the output of the default printer:

Listing 2: `java -jar TypeChecker.jar -i test.txt`

Yes

$$\frac{}{x: \text{bool} \text{ ? } x : \text{bool}} \text{---(var)} \quad \frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{---(subBase)}$$

$$\frac{}{x: \text{bool} \text{ ? } x : \text{int}} \text{---(subsumption)}$$

Here is the output of the latex printer:

Listing 3: `java -jar TypeChecker.jar -i test.txt -latex`

Yes

```
\begin{prooftree}
\AxiomC{} \RightLabel{\scriptsize var}
\UnaryInfC{$x: \text{bool} \vdash x : \text{bool}$}
\AxiomC{\scriptsize SubBase}
\RightLabel{\scriptsize subBase}
\UnaryInfC{$\text{bool} <: \text{int}$}
\RightLabel{\scriptsize subsumption}
\BinaryInfC{$x: \text{bool} \vdash x : \text{int}$}
\end{prooftree}
```

The following sections focus on terms and rules used in the **Type Checker**.

3 Simple types

Annotated simply typed terms recognized by the parser have the form:

$$t ::= x \mid (t_1 t_2)[T] \mid \lambda x. t$$

The annotation $[T]$ in the application term $(t_1 t_2)[T]$ is used to remove the non-determinism in the application rule. This annotation is required in the parser.

3.1 Typing rules for simple types

1. $\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \text{var}$
2. $\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash (t_1 t_2)[T_1] : T_2} \text{app}$
3. $\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x. t : T_1 \rightarrow T_2} \lambda$

3.2 Subtyping rules for simple types

1. $\frac{\Gamma \vdash t : T_1 \quad T_1 <: T_2}{\Gamma \vdash t : T_2} \text{subsumption}$
2. $\frac{}{T <: T} \text{reflexive}$

3.
$$\frac{SubBase(b_1, b_2)}{b_1 <: b_2} \text{subBase}$$
4.
$$\frac{T'_1 <: T_1 \quad T_2 <: T'_2}{T_1 \rightarrow T_2 <: T'_1 \rightarrow T'_2} \text{arrow}$$
5.
$$\frac{T_1 <: T_2 \quad T_2 <: T_3}{T_1 <: T_3} \text{transitive}$$

3.3 Examples

Type checking is complete for annotated simply typed lambda calculus with subtypes. This means the answer is **Yes** if the judgment can be derived, and **No** if the judgment can not be derived. The Unknown answer is never returned because annotated simple types are decidable.

Here are some examples tested by the **TypeChecker**. A red rule in the derivation tree mean its judgment is not derivable.

1. Valid variable rule

$$\frac{}{x : T \vdash x : T} \text{var}$$

2. Invalid variable rule

$$\frac{}{\cdot \vdash x : T} \text{invalid var}$$

3. Lambda & application rules

$$\frac{\frac{\frac{}{x : (T1 \rightarrow T2), y : T1 \vdash x : (T1 \rightarrow T2)} \text{var} \quad \frac{}{x : (T1 \rightarrow T2), y : T1 \vdash y : T1} \text{var}}{x : (T1 \rightarrow T2), y : T1 \vdash (x y)[T1] : T2} \text{app} \quad \frac{x : (T1 \rightarrow T2), y : T1 \vdash (x y)[T1] : T2}{x : (T1 \rightarrow T2) \vdash \lambda y.(x y)[T1] : (T1 \rightarrow T2)} \lambda}{\cdot \vdash \lambda x.\lambda y.(x y)[T1] : ((T1 \rightarrow T2) \rightarrow (T1 \rightarrow T2))} \lambda$$

4. Direct Subtyping

(a) Valid subtyping

$$\frac{\frac{}{x : bool \vdash x : bool} \text{var} \quad \frac{SubBase(bool, int)}{bool <: int} \text{subBase}}{x : bool \vdash x : int} \text{subsumption}$$

(b) Invalid subtyping

$$\frac{\frac{}{x : int \vdash x : int} \text{var} \quad \frac{\perp}{int <: bool} \text{invalid}}{x : int \vdash x : bool} \text{subsumption}$$

5. Transitive subtyping

$$\frac{\frac{}{x : bool \vdash x : bool} \text{var} \quad \frac{SubBase(bool, int)}{bool <: int} \text{subBase} \quad \frac{SubBase(int, double)}{int <: double} \text{subBase}}{x : bool \vdash x : double} \text{subsumption}$$

6. Transitive subtyping

$$\frac{\frac{x : \text{bool} \vdash x : \text{bool}}{\text{var}} \quad \frac{\frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase} \quad \frac{\frac{\text{SubBase}(\text{int}, \text{quotient})}{\text{int} <: \text{quotient}} \text{subBase} \quad \frac{\text{SubBase}(\text{quotient}, \text{double})}{\text{quotient} <: \text{double}} \text{subBase}}{\text{bool} <: \text{quotient}} \text{transitive} \quad \frac{\text{SubBase}(\text{quotient}, \text{double})}{\text{quotient} <: \text{double}} \text{subBase}}{\text{bool} <: \text{double}} \text{transitive} \quad \frac{x : \text{bool} \vdash x : \text{bool}}{x : \text{bool} \vdash x : \text{double}} \text{subsumption}$$

7. Arrow subtyping (subBase)

$$\frac{\frac{x : (\text{int} \rightarrow \text{bool}) \vdash x : (\text{int} \rightarrow \text{bool})}{\text{var}} \quad \frac{\frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase} \quad \frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase}}{(\text{int} \rightarrow \text{bool}) <: (\text{bool} \rightarrow \text{int})} \text{arrow} \quad \frac{x : (\text{int} \rightarrow \text{bool}) \vdash x : (\text{int} \rightarrow \text{bool})}{x : (\text{int} \rightarrow \text{bool}) \vdash x : (\text{bool} \rightarrow \text{int})} \text{subsumption}$$

8. Arrow subtyping (reflexive, subBase)

$$\frac{\frac{x : (\text{int} \rightarrow \text{bool}) \vdash x : (\text{int} \rightarrow \text{bool})}{\text{var}} \quad \frac{\frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase} \quad \frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase}}{(\text{int} \rightarrow \text{bool}) <: (\text{int} \rightarrow \text{int})} \text{arrow} \quad \frac{x : (\text{int} \rightarrow \text{bool}) \vdash x : (\text{int} \rightarrow \text{bool})}{x : (\text{int} \rightarrow \text{bool}) \vdash x : (\text{int} \rightarrow \text{int})} \text{subsumption}$$

9. Arrow subtyping (reflexive, subBase)

$$\frac{\frac{x : (\text{int} \rightarrow T), y : \text{bool} \vdash x : (\text{int} \rightarrow T)}{\text{var}} \quad \frac{\frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase} \quad \frac{T <: T}{\text{reflexive}} \text{arrow}}{(\text{int} \rightarrow T) <: (\text{bool} \rightarrow T)} \text{subsumption} \quad \frac{x : (\text{int} \rightarrow T), y : \text{bool} \vdash x : (\text{bool} \rightarrow T)}{x : (\text{int} \rightarrow T), y : \text{bool} \vdash y : \text{bool}} \text{app} \quad \frac{x : (\text{int} \rightarrow T), y : \text{bool} \vdash (x y)[\text{bool}] : T}{x : (\text{int} \rightarrow T) \vdash \lambda y. (x y)[\text{bool}] : (\text{bool} \rightarrow T)} \lambda \quad \frac{\cdot \vdash \lambda x. \lambda y. (x y)[\text{bool}] : ((\text{int} \rightarrow T) \rightarrow (\text{bool} \rightarrow T))}{\cdot \vdash \lambda x. \lambda y. (x y)[\text{bool}] : ((\text{int} \rightarrow T) \rightarrow (\text{bool} \rightarrow T))} \lambda$$

10. Arrow subtyping (invalid)

$$\frac{\frac{x : (\text{bool} \rightarrow \text{bool}) \vdash x : (\text{bool} \rightarrow \text{bool})}{\text{var}} \quad \frac{\frac{\perp}{\text{int} <: \text{bool}} \text{invalid} \quad \frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase}}{(\text{bool} \rightarrow \text{bool}) <: (\text{int} \rightarrow \text{int})} \text{arrow} \quad \frac{x : (\text{bool} \rightarrow \text{bool}) \vdash x : (\text{bool} \rightarrow \text{bool})}{x : (\text{bool} \rightarrow \text{bool}) \vdash x : (\text{int} \rightarrow \text{int})} \text{subsumption}$$

11. Arrow subtyping (Invalid)

$$\frac{\frac{x : (\text{bool} \rightarrow T), y : \text{bool} \vdash x : (\text{bool} \rightarrow T)}{\text{var}} \quad \frac{\frac{\perp}{\text{int} <: \text{bool}} \text{invalid} \quad \frac{T <: T}{\text{reflexive}} \text{arrow}}{(\text{bool} \rightarrow T) <: (\text{int} \rightarrow T)} \text{subsumption} \quad \frac{x : (\text{bool} \rightarrow T), y : \text{bool} \vdash y : \text{bool}}{x : (\text{bool} \rightarrow T), y : \text{bool} \vdash y : \text{int}} \text{app} \quad \frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase} \quad \frac{x : (\text{bool} \rightarrow T), y : \text{bool} \vdash (x y)[\text{int}] : T}{x : (\text{bool} \rightarrow T) \vdash \lambda y. (x y)[\text{int}] : (\text{bool} \rightarrow T)} \lambda \quad \frac{\cdot \vdash \lambda x. \lambda y. (x y)[\text{int}] : ((\text{bool} \rightarrow T) \rightarrow (\text{bool} \rightarrow T))}{\cdot \vdash \lambda x. \lambda y. (x y)[\text{int}] : ((\text{bool} \rightarrow T) \rightarrow (\text{bool} \rightarrow T))} \lambda$$

3.4 Delaying applying the subsumption rule

The subsumption rule can be delayed until the term in the judgment is a variable. This simplifies the code since there is only one rule to be applied for application and λ abstraction terms. Here is the correctness proof using the subsumption rule:

$$\frac{\Gamma \vdash t : T \quad T <: T'}{\Gamma \vdash t : T'} \text{ subsumption}$$

Proof. By induction hypothesis on the structure of the term t :

1. Case $t = x$: trivial since t is a variable.

$$\frac{\Gamma \vdash x : T \quad T <: T'}{\Gamma \vdash x : T'} \text{ subsumption}$$

2. Case $t = t_1 t_2$: assume $T_2 <: T'_2$ and the following derivation:

$$\frac{\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash (t_1 t_2)[T_1] : T_2} \text{ app} \quad T_2 <: T'_2}{\Gamma \vdash (t_1 t_2)[T_1] : T'_2} \text{ subsumption}$$

We can get a different derivation tree where the subsumption rule is delayed:

$$\frac{\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \frac{\frac{T_1 <: T_1}{\text{reflexive}} \quad T_2 <: T'_2}{T_1 \rightarrow T_2 <: T_1 \rightarrow T'_2} \text{ arrow}}{\Gamma \vdash t_1 : T_1 \rightarrow T'_2} \text{ subsumption} \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash (t_1 t_2)[T_1] : T'_2} \text{ app}$$

Using the induction hypothesis, the subsumption rule can be delayed until variable terms are generated in the derivations of $\Gamma \vdash t_1 : T_1 \rightarrow T'_2$ and $\Gamma \vdash t_2 : T_1$.

3. Case $t = \lambda x. t'$: assume $T_1 \rightarrow T_2 <: T'_1 \rightarrow T'_2$ and the following derivation:

$$\frac{\frac{\Gamma, x : T_1 \vdash t' : T_2}{\Gamma \vdash \lambda x. t' : T_1 \rightarrow T_2} \lambda \quad \frac{\frac{T'_1 <: T_1 \quad T_2 <: T'_2}{T_1 \rightarrow T_2 <: T'_1 \rightarrow T'_2} \text{ arrow}}{\Gamma \vdash \lambda x. t' : T'_1 \rightarrow T'_2} \text{ subsumption}$$

Alternatively we can derive:

$$\frac{\frac{\Gamma, x : T'_1 \vdash t' : T_2 \quad T_2 <: T'_2}{\Gamma, x : T'_1 \vdash t' : T'_2} \text{ subsumption}}{\Gamma \vdash \lambda x. t' : T'_1 \rightarrow T'_2} \lambda$$

Assuming $T'_1 <: T_1$ is derivable, then by the subsumption rule:

$$\frac{\frac{\Gamma, x : T'_1 \vdash x : T'_1}{\text{var}} \quad T'_1 <: T_1}{\Gamma, x : T'_1 \vdash x : T_1} \text{ subsumption}$$

Therefore if $\Gamma, x : T_1 \vdash t' : T_2$ is derivable, then $\Gamma, x : T'_1 \vdash t' : T_2$ is also derivable which concludes the proof. □

Remark. If $\Gamma, x : T'_1 \vdash t' : T_2$, it is not always true that $\Gamma, x : T_1 \vdash t' : T_2$ where $T'_1 <: T_1$.

A counter example would be

$$x : \text{bool} \vdash x : \text{int} \not\Rightarrow x : \text{double} \vdash x : \text{int}, \quad \text{bool} <: \text{int} <: \text{double}$$

However

$$x : \text{int} \vdash x : \text{double} \Rightarrow x : \text{bool} \vdash x : \text{double}, \quad \text{bool} <: \text{int} <: \text{double}$$

Therefore in the following example, using the subsumption rule first would fail and slow the type checker because it needs to backtrack and check the λ rule. However using the λ rule first would prove the derivation and it is faster.

$$\frac{\frac{x : \text{double} \vdash x : \text{int}}{\Gamma \vdash \lambda x. x : \text{double} \rightarrow \text{int}} \lambda \quad \frac{\frac{\text{SubBase}(\text{bool}, \text{double})}{\text{bool} <: \text{double}} \text{subBase} \quad \frac{\text{int} <: \text{int}}{\text{int} <: \text{int}} \text{reflexive}}{\text{double} \rightarrow \text{int} <: \text{bool} \rightarrow \text{int}} \text{arrow}}{\Gamma \vdash \lambda x. x : \text{bool} \rightarrow \text{int}} \text{subsumption}$$

$$\frac{\frac{\Gamma, x : \text{bool} \vdash x : \text{bool}}{\Gamma, x : \text{bool} \vdash x : \text{int}} \text{var} \quad \frac{\text{SubBase}(\text{bool}, \text{int})}{\text{bool} <: \text{int}} \text{subBase}}{\Gamma, x : \text{bool} \vdash x : \text{int}} \text{subsumption}$$

$$\frac{\Gamma, x : \text{bool} \vdash x : \text{int}}{\Gamma \vdash \lambda x. x : \text{bool} \rightarrow \text{int}} \lambda$$

4 System F

Annotated system F terms, and types recognized by the parser have the form:

$$t ::= x \mid (t_1 t_2)[T] \mid \lambda x. t \mid t [[T]]$$

$$T ::= X \mid T_1 \rightarrow T_2 \mid \forall X. T$$

The annotation $[T]$ in the application term $(t_1 t_2)[T]$ is used to remove the non-determinism in the application rule. The annotation $[[T]]$ in the term $t[[T]]$ is used to remove the non-determinism in the elimination rule. Unlike the application annotation, the elimination annotation is not required in the parser which makes the **TypeChecker** incomplete. Whenever the **TypeChecker** needs the elimination annotation and it is not provided, it returns the current derivation tree with answer **Unknown**.

4.1 Rules

1. If X is not free in Γ

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t : \forall X. T} \text{introduction}$$

2. If X is free in Γ , choose X_i such that X_i is not free in Γ

$$\frac{\frac{\Gamma \vdash t : [X_i/X]T}{\Gamma \vdash t : \forall X_i. [X_i/X]T} \text{introduction}}{\Gamma \vdash t : \forall X. T} \text{renaming}$$

3. Elimination rule with annotation

$$\frac{\Gamma \vdash t : \forall X. [X/T']T}{\Gamma \vdash t [[T']] : T} \text{elimination}$$

4.2 Examples

Here are some examples tested by the **TypeChecker**. A red rule in the derivation tree mean its judgment is not derivable. A blue rule means the **TypeChecker** needs more annotation to continue the type checking.

1. Same type variable

$$\frac{}{x : \forall X.X \vdash x : \forall X.X} \text{ var}$$

2. Different type variables

$$\frac{}{x : \forall X.X \vdash x : \forall Y.Y} \text{ var}$$

3. Arrows

$$\frac{}{x : \forall X.(X \rightarrow X) \vdash x : \forall Y.(Y \rightarrow Y)} \text{ var}$$

4. Y is free in the typing context and the term type

$$\frac{}{x : \forall X.(X \rightarrow Y) \vdash x : \forall Z.(Z \rightarrow Y)} \text{ var}$$

5. Y is free in the typing context

$$\frac{}{x : \forall X.(X \rightarrow Y) \vdash x : \forall Y.(Y \rightarrow Y)} \text{ invalid var}$$

6. Y is free in the typing context, and Z is free in the term type

$$\frac{}{x : \forall X.(X \rightarrow Y) \vdash x : \forall Y.(Y \rightarrow Z)} \text{ invalid var}$$

7. Elimination annotation

$$\frac{\frac{}{x : \forall X.X \vdash x : \forall X1.X1} \text{ var}}{x : \forall X.X \vdash x[Y] : Y} \text{ elimination}$$

8. Elimination annotation with arrow

$$\frac{\frac{}{x : \forall X.X \vdash x : \forall X1.X1} \text{ var}}{x : \forall X.X \vdash x[(Y \rightarrow Y)] : (Y \rightarrow Y)} \text{ elimination}$$

9. Nested elimination annotation

$$\frac{\frac{\frac{}{x : \forall X.X \vdash x : \forall X1.X1} \text{ var}}{x : \forall X.X \vdash x[[\forall X.X \rightarrow \forall X.X]] : (\forall X.X \rightarrow \forall X.X)} \text{ elimination}}{x : \forall X.X \vdash (x[[\forall X.X \rightarrow \forall X.X]]) x[\forall X.X] : \forall X.X} \text{ app}$$

10. Application or introduction non-determinism: The **TypeChecker** attempts applications first. When it fails, it backtracks and attempts the introduction rule as shown in the next example.

$$\begin{array}{c}
\frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash x : (T_1 \rightarrow T_2)} \text{var} \quad \frac{}{T_1 <: T_1} \text{reflexive} \quad \frac{\perp}{T_2 <: \forall X.T_2} \text{invalid} \\
\frac{}{(T_1 \rightarrow T_2) <: (T_1 \rightarrow \forall X.T_2)} \text{arrow} \\
\frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash x : (T_1 \rightarrow \forall X.T_2)} \text{subsumption} \quad \frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash y : T_1} \text{var} \\
\frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash (x y)[T_1] : \forall X.T_2} \text{app}
\end{array}$$

11. Introduction branch

$$\frac{\frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash x : (T_1 \rightarrow T_2)} \text{var} \quad \frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash y : T_1} \text{var}}{x : (T_1 \rightarrow T_2), y : T_1 \vdash (x y)[T_1] : T_2} \text{app} \\
\frac{}{x : (T_1 \rightarrow T_2), y : T_1 \vdash (x y)[T_1] : \forall X.T_2} \text{introduction}$$

12. Application or elimination non-determinism

$$\begin{array}{c}
\frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash x : (T_1 \rightarrow \forall X.T_2)} \text{var} \quad \frac{}{T_1 <: T_1} \text{reflexive} \quad \frac{\perp}{\forall X.T_2 <: T_2} \text{invalid} \\
\frac{}{(T_1 \rightarrow \forall X.T_2) <: (T_1 \rightarrow T_2)} \text{arrow} \\
\frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash x : (T_1 \rightarrow T_2)} \text{subsumption} \quad \frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash y : T_1} \text{var} \\
\frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash (x y)[T_1][[Y]] : T_2} \text{app}
\end{array}$$

13. Elimination branch

$$\frac{\frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash x : (T_1 \rightarrow \forall X_1.T_2)} \text{var} \quad \frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash y : T_1} \text{var}}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash (x y)[T_1] : \forall X_1.T_2} \text{app} \\
\frac{}{x : (T_1 \rightarrow \forall X.T_2), y : T_1 \vdash (x y)[T_1][[Y]] : T_2} \text{elimination}$$

14. Zero

$$\frac{\frac{}{z : X, s : (X \rightarrow X) \vdash z : X} \text{var} \quad \frac{}{s : (X \rightarrow X) \vdash (\lambda z.z) : (X \rightarrow X)} \lambda}{\cdot \vdash (\lambda s.(\lambda z.z)) : ((X \rightarrow X) \rightarrow (X \rightarrow X))} \lambda \\
\frac{}{\cdot \vdash (\lambda s.(\lambda z.z)) : \forall X.((X \rightarrow X) \rightarrow (X \rightarrow X))} \text{introduction}$$

15. Zero with free variable X

$$\frac{\frac{}{y : X, z : X_2, s : (X_2 \rightarrow X_2) \vdash z : X_2} \text{var} \quad \frac{}{y : X, s : (X_2 \rightarrow X_2) \vdash (\lambda z.z) : (X_2 \rightarrow X_2)} \lambda}{y : X \vdash (\lambda s.(\lambda z.z)) : ((X_2 \rightarrow X_2) \rightarrow (X_2 \rightarrow X_2))} \lambda \\
\frac{}{y : X \vdash (\lambda s.(\lambda z.z)) : \forall X_2.((X_2 \rightarrow X_2) \rightarrow (X_2 \rightarrow X_2))} \text{introduction} \\
\frac{}{y : X \vdash (\lambda s.(\lambda z.z)) : \forall X.((X \rightarrow X) \rightarrow (X \rightarrow X))} \text{renaming}$$

