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

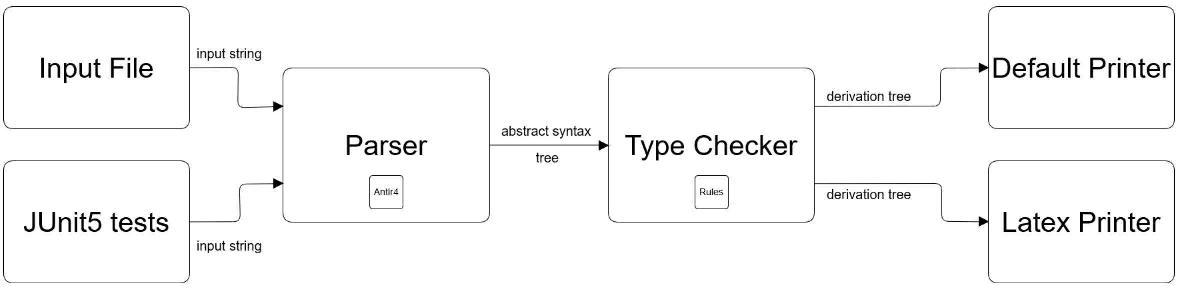

Figure 1: Project architecture.

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
                                          SubBase(bool, int)
-------------------------(var)            -------------------------(subBase)
x: bool ? x : bool                                  bool <: int
-------------------------------------------------(subsumption)
x: bool ? x : int
```

Here is the output of the latex printer which uses bussproofs package centered proofs:

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

```
Yes
\begin{prooftree}
\AxiomC{} \RightLabel{\scriptsize var}
\UnaryInfC{$x: bool \vdash x : bool$}
\AxiomC{\scriptsize $SubBase(bool,int)$}
\RightLabel{\scriptsize subBase}
\UnaryInfC{$bool <: int$}
\RightLabel{\scriptsize subsumption}
\BinaryInfC{$x: bool \vdash x : int$}
\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. $$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash (t_1 t_2)[T_1] : T_2}\text{app}$$

3. $$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x.t : T_1 \to T_2}\lambda$$

3.2 Subtyping rules for simple types

1. $$\frac{\Gamma \vdash t : T_1 \qquad T_1 <: T_2}{\Gamma \vdash t : T_2}\text{subsumption}$$

2. $$\frac{}{T <: T}\text{reflexive}$$

3. $$\frac{SubBase(b_1, b_2)}{b_1 <: b_2}\ \text{subBase}$$

4. $$\frac{T_1' <: T_1 \qquad T_2 <: T_2'}{T_1 \to T_2 <: T2_1 \to T_2'}\ \text{arrow}$$

5. $$\frac{T_1 <: T_2 \qquad T_2 <: T_3}{T_1 <: T_3}\ \text{transitive}$$

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**

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : (T1 \to T2), y : T1 \vdash x : (T1 \to T2)}\ \text{var} \qquad \dfrac{}{x : (T1 \to T2), y : T1 \vdash y : T1}\ \text{var}}{x : (T1 \to T2), y : T1 \vdash (x\ y)[T1] : T2}\ \text{app}}{x : (T1 \to T2) \vdash \lambda y.(x\ y)[T1] : (T1 \to T2)}\ \lambda}{\cdot \vdash \lambda x.\lambda y.(x\ y)[T1] : ((T1 \to T2) \to (T1 \to T2))}\ \lambda$$

4. **Direct Subtyping**

   (a) **Valid subtyping**

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

   (b) **Invalid subtyping**

$$\dfrac{\dfrac{}{x : int \vdash x : int}\ \text{var} \qquad \dfrac{\perp}{int <: bool}\ \text{invalid}}{x : int \vdash x : bool}\ \text{subsumption}$$

5. **Transitive subtyping**

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

6. **Transitive subtyping**

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

7. **Arrow subtyping (subBase)**

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

8. **Arrow subtyping (reflexive, subBase)**

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

9. **Arrow subtyping (reflexive, subBase)**

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

10. **Arrow subtyping (invalid)**

$$\dfrac{\dfrac{}{x : (bool \to bool) \vdash x : (bool \to bool)}\,\text{var} \quad \dfrac{\dfrac{\bot}{int <: bool}\,\text{invalid} \quad \dfrac{\text{SubBase}(bool, int)}{bool <: int}\,\text{subBase}}{(bool \to bool) <: (int \to int)}\,\text{arrow}}{x : (bool \to bool) \vdash x : (int \to int)}\,\text{subsumption}$$

11. **Arrow subtyping (Invalid)**

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

3.4 Delaying applying the subsumption rule

The sumbumption 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:

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

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

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

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

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

$$\frac{\dfrac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash (t_1 t_2)[T_1] : T_2}\ \text{app} \qquad T_2 <: T_2'}{\Gamma \vdash (t_1 t_2)[T_1] : T_2'}\ \text{subsumption}$$

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

$$\frac{\dfrac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \dfrac{\dfrac{}{T_1 <: T_1}\ \text{reflexive} \qquad T_2 <: T_2'}{T_1 \to T_2 <: T_1 \to T_2'}\ \text{arrow}}{\Gamma \vdash t_1 : T_1 \to T_2'}\ \text{subsumption} \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash (t_1 t_2)[T_1] : T_2'}\ \text{app}$$

Using the induction hypothesis, the subsumption rule can be delayed until variable terms are generated in the derivations of $\Gamma \vdash t_1 : T_1 \to T_2'$ and $\Gamma \vdash t_2 : T_1$.

3. Case $t = \lambda x.t'$: assume $T_1 \to T_2 <: T_1' \to T_2'$ and the following derivation:

$$\frac{\dfrac{\Gamma, x : T_1 \vdash t' : T_2}{\Gamma \vdash \lambda x.t' : T_1 \to T_2}\ \lambda \qquad \dfrac{T_1' <: T_1 \qquad T_2 <: T_2'}{T_1 \to T_2 <: T_1' \to T_2'}\ \text{arrow}}{\Gamma \vdash \lambda x.t' : T_1' \to T_2'}\ \text{subsumption}$$

Alternatively we can derive:

$$\frac{\dfrac{\Gamma, x : T_1' \vdash t' : T_2 \qquad T_2 <: T_2'}{\Gamma, x : T_1' \vdash t' : T_2'}\ \text{subsumption}}{\Gamma \vdash \lambda x.t' : T_1' \to T_2'}\ \lambda$$

Assuming $T_1' <: T_1$ is derivable, then by the subsumption rule:

$$\frac{\dfrac{}{\Gamma, x : T_1' \vdash x : T_1'}\ \text{var} \qquad T_1' <: T_1}{\Gamma, x : T_1' \vdash x : T_1}\ \text{subsumption}$$

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.

$\square$

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

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

However

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

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.

$$\dfrac{\dfrac{\color{red}{x : \mathit{double} \vdash x : \mathit{int}}}{\Gamma \vdash \lambda x.x : \mathit{double} \to \mathit{int}}\,\lambda \qquad \dfrac{\dfrac{\mathit{SubBase}(\mathit{bool}, \mathit{double})}{\mathit{bool} <: \mathit{double}}\,\text{subBase} \qquad \dfrac{}{\mathit{int} <: \mathit{int}}\,\text{reflexive}}{\mathit{double} \to \mathit{int} <: \mathit{bool} \to \mathit{int}}\,\text{arrow}}{\Gamma \vdash \lambda x.x : \mathit{bool} \to \mathit{int}}\,\text{subsumption}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : \mathit{bool} \vdash x : \mathit{bool}}\,\text{var} \qquad \dfrac{\mathit{SubBase}(\mathit{bool}, \mathit{int})}{\mathit{bool} <: \mathit{int}}\,\text{suBase}}{\Gamma, x : \mathit{bool} \vdash x : \mathit{int}}\,\text{subsumption}}{\Gamma \vdash \lambda x.x : \mathit{bool} \to \mathit{int}}\,\lambda$$

4 System F

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

$$t ::= x \mid (t_1 t_2)[T] \mid \lambda x.t \mid t\ [[T]]$$
$$T ::= X \mid T_1 \to T_2 \mid \forall X.T$$

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

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

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

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

3. Elimination rule with annotation

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

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

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

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

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

5. Y is free in the typing context

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

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

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

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

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

9. Nested elimination annotation

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

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.

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

11. Introduction branch

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

12. Application or elimination non-determinism

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

13. Elimination branch

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

14. Zero

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

15. Zero with free variable X

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

Successor missing annotation

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash s : (X \to X)$}
\AxiomC{}
\RightLabel{\textcolor{blue}{unknown}}
\UnaryInfC{\textcolor{blue}{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash n : ((X \to X) \to (X \to X))$}}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash s : (X \to X)$}
\RightLabel{app}
\BinaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash (n\ s)[(X \to X)] : (X \to X)$}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash z : X$}
\RightLabel{app}
\BinaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash ((n\ s)[(X \to X)]\ z)[X] : X$}
\RightLabel{app}
\BinaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash (s\ ((n\ s)[(X \to X)]\ z)[X])[X] : X$}
\RightLabel{λ}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), s : (X \to X) \vdash (\lambda z.(s\ ((n\ s)[(X \to X)]\ z)[X])[X]) : (X \to X)$}
\RightLabel{λ}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)) \vdash (\lambda s.(\lambda z.(s\ ((n\ s)[(X \to X)]\ z)[X])[X])) : ((X \to X) \to (X \to X))$}
\RightLabel{introduction}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)) \vdash (\lambda s.(\lambda z.(s\ ((n\ s)[(X \to X)]\ z)[X])[X])) : \forall X.((X \to X) \to (X \to X))$}
\RightLabel{λ}
\UnaryInfC{$\cdot \vdash (\lambda n.(\lambda s.(\lambda z.(s\ ((n\ s)[(X \to X)]\ z)[X])[X]))) : (\forall X.((X \to X) \to (X \to X)) \to \forall X.((X \to X) \to (X \to X)))$}
\end{prooftree}
$$

Successor well annotated

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash s : (X \to X)$}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash n : \forall X1.((X1 \to X1) \to (X1 \to X1))$}
\RightLabel{elimination}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash n[[X]][X] : ((X \to X) \to (X \to X))$}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash s : (X \to X)$}
\RightLabel{app}
\BinaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash (n[[X]]\ s)[(X \to X)] : (X \to X)$}
\AxiomC{}
\RightLabel{var}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash z : X$}
\RightLabel{app}
\BinaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash ((n[[X]]\ s)[(X \to X)]\ z)[X] : X$}
\RightLabel{app}
\BinaryInfC{$n : \forall X.((X \to X) \to (X \to X)), z : X, s : (X \to X) \vdash (s\ ((n[[X]]\ s)[(X \to X)]\ z)[X])[X] : X$}
\RightLabel{λ}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)), s : (X \to X) \vdash (\lambda z.(s\ ((n[[X]]\ s)[(X \to X)]\ z)[X])[X]) : (X \to X)$}
\RightLabel{λ}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)) \vdash (\lambda s.(\lambda z.(s\ ((n[[X]]\ s)[(X \to X)]\ z)[X])[X])) : ((X \to X) \to (X \to X))$}
\RightLabel{introduction}
\UnaryInfC{$n : \forall X.((X \to X) \to (X \to X)) \vdash (\lambda s.(\lambda z.(s\ ((n[[X]]\ s)[(X \to X)]\ z)[X])[X])) : \forall X.((X \to X) \to (X \to X))$}
\RightLabel{λ}
\UnaryInfC{$\cdot \vdash (\lambda n.(\lambda s.(\lambda z.(s\ ((n[[X]]\ s)[(X \to X)]\ z)[X])[X]))) : (\forall X.((X \to X) \to (X \to X)) \to \forall X.((X \to X) \to (X \to X)))$}
\end{prooftree}
$$