

Lecture 25 – Type Inference (1)

COSE212: Programming Languages

Jihyeok Park



2026 Fall

- **Polymorphism** is to use a single entity as **multiple types**, and there are various kinds of polymorphism:
 - **Parametric polymorphism**
 - **Subtype polymorphism**
 - **Ad-hoc polymorphism**
 - ...
- PTFAE – TFAE with **parametric polymorphism**.
- STFAE – TFAE with **subtype polymorphism**.
- In this lecture, we will learn **type inference**.

Definition (Type Inference)

Type inference is the process of automatically inferring the types of expressions.

The goal of **type inference algorithm** is to infer the type of an expression without **explicit type annotations** given by programmers.

Let's consider the following RFAE expression:

```
/* RFAE */  
def sum(x) = if (x < 1) 0 else x + sum(x + -1);  
sum
```

How can we **automatically infer** the type of sum?

- 1 Introduce **type variables** to denote unknown types
- 2 Collect the **type constraints** on the types
- 3 Find a **solution** (substitution of type variables) to the constraints

1. Example 1 – sum

2. Example 2 – app

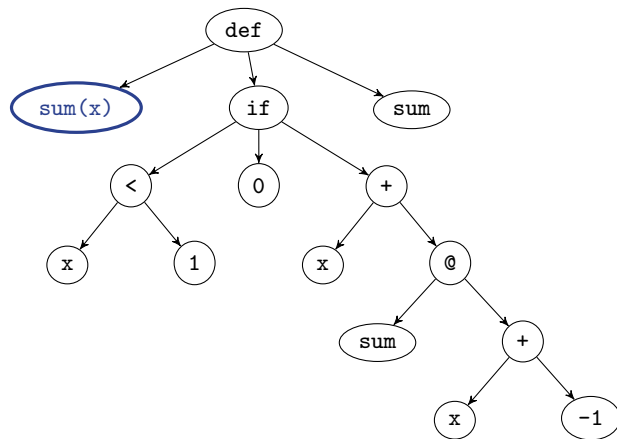
3. Example 3 – id

1. Example 1 – sum

2. Example 2 – app

3. Example 3 – id

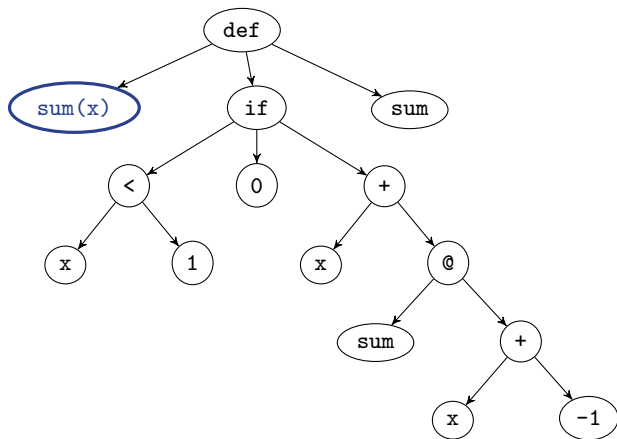
Example 1 – sum



Type Environment

X	T
x	???
sum	???

Example 1 – sum



Type Environment

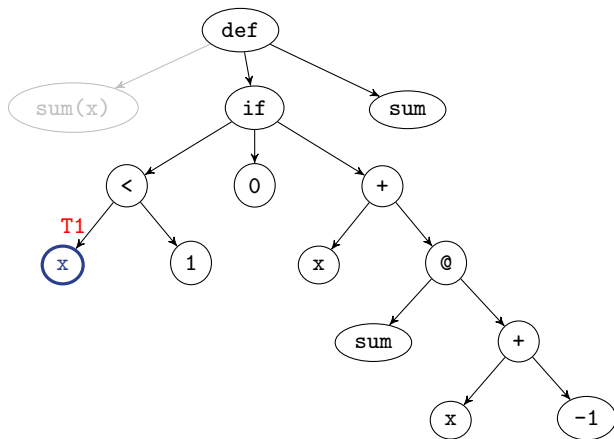
X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	-
$T2$	-

Let's define **type variables** for unknown types.

Example 1 – sum



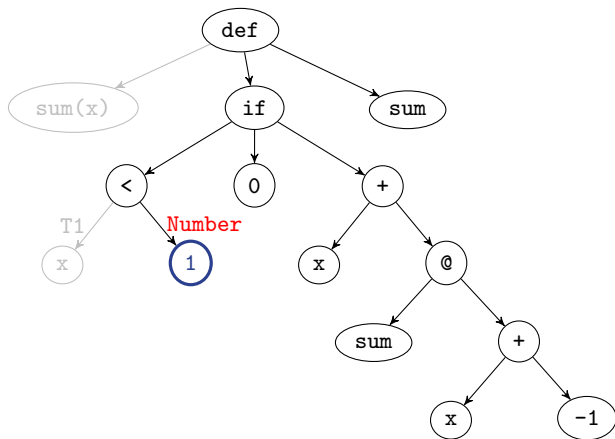
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	-
$T2$	-

Example 1 – sum



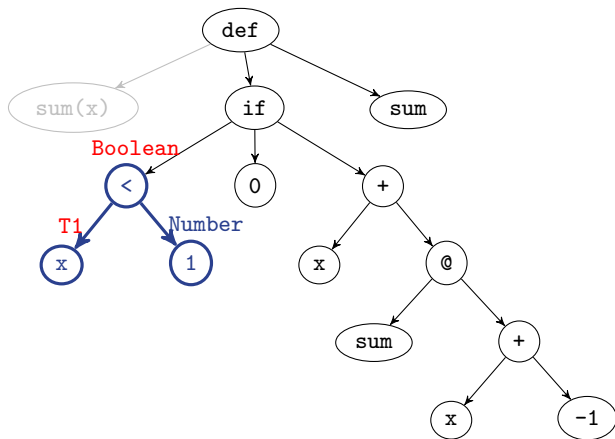
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	-
$T2$	-

Example 1 – sum



Type Environment

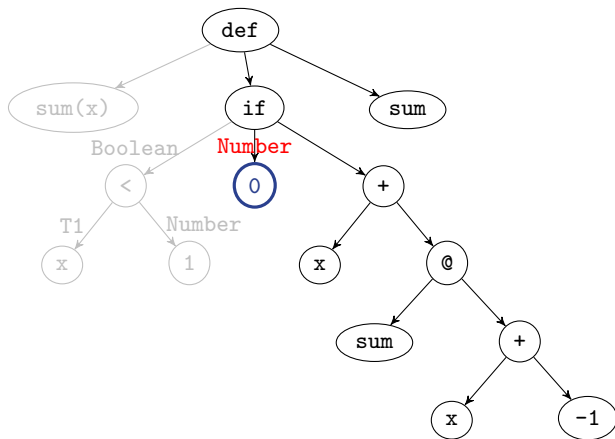
X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	Number
$T2$	$-$

The **operands** of $<$ must be of type **Number**.
So, we collected a **type constraint**: $T1 == \text{Number}$.

Example 1 – sum



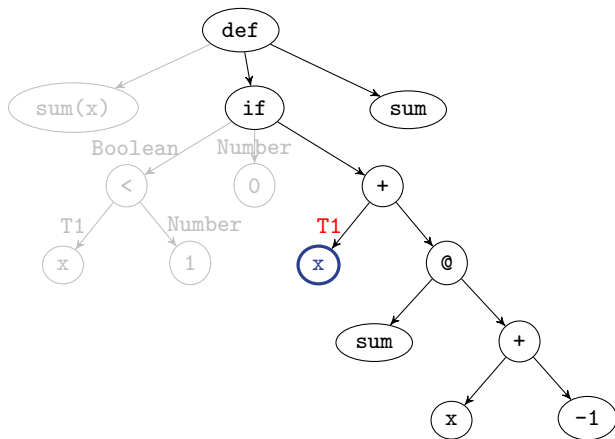
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$-$

Example 1 – sum



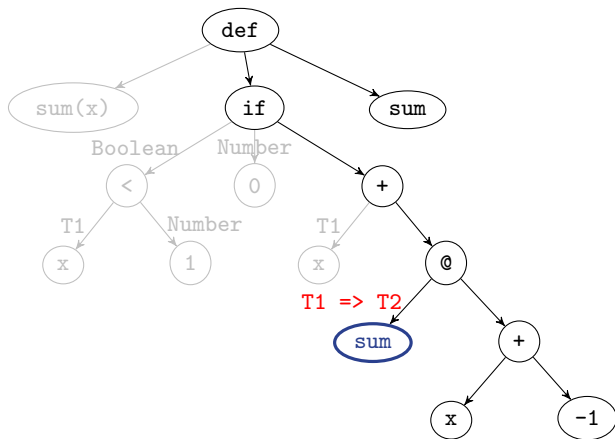
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$-$

Example 1 – sum



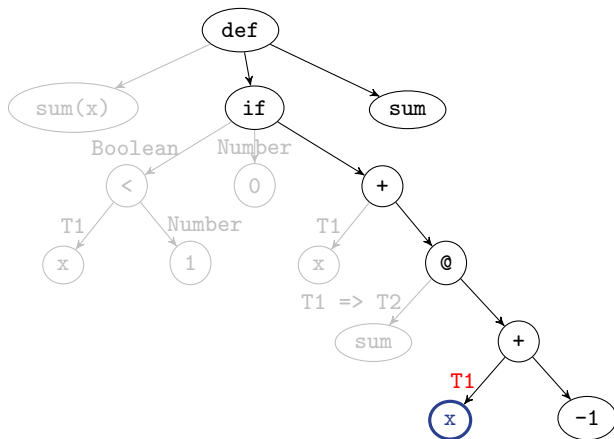
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$-$

Example 1 – sum



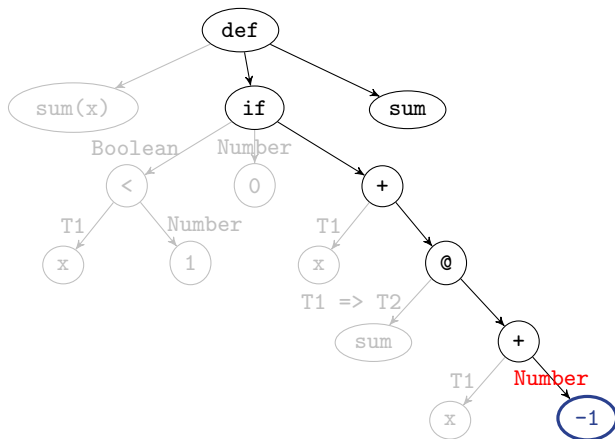
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$-$

Example 1 – sum



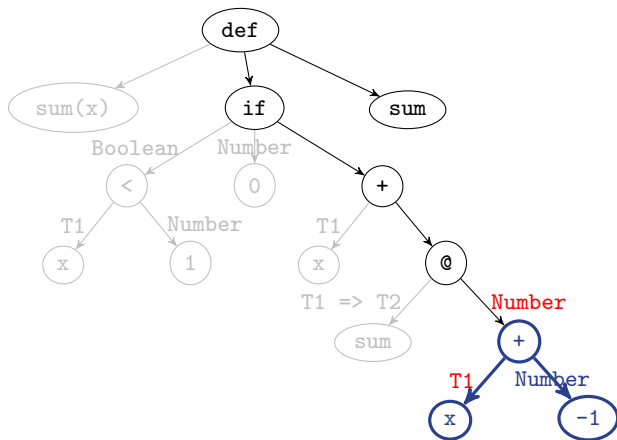
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$-$

Example 1 – sum



Type Environment

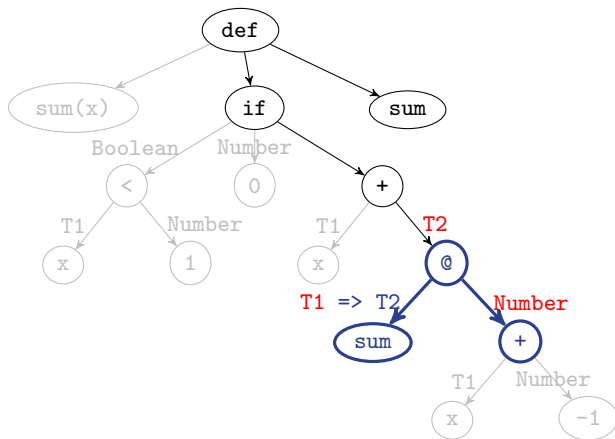
X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$-$

The **operands** of $-$ must be of type **Number**.
We collected a **type constraint**: $T1 == Number$.
But, it is not a new constraint.

Example 1 – sum



Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

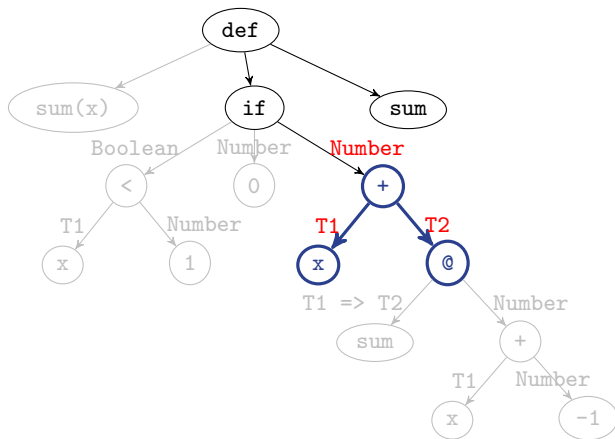
X_α	T
$T1$	$Number$
$T2$	$-$

The **argument type** should be equal to the **parameter type**.

We collected a **type constraint**: $T1 == Number$.

Again, it is not a new constraint.

Example 1 – sum



Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

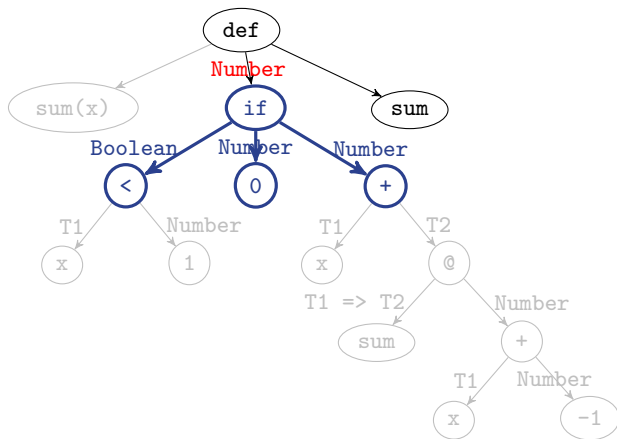
X_α	T
$T1$	$Number$
$T2$	$Number$

The **operands** of $+$ must be of type **Number**.

We collected **type constraints**: $T1 == Number$ and $T2 == Number$.

The second one is a new constraint!

Example 1 – sum



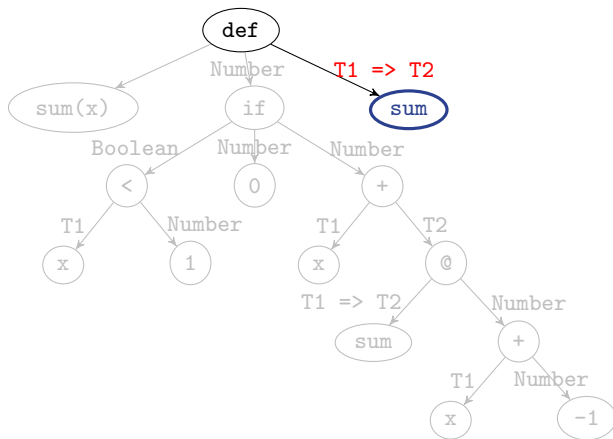
Type Environment

X	T
x	$T1$
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	$Number$
$T2$	$Number$

Example 1 – sum



Type Environment

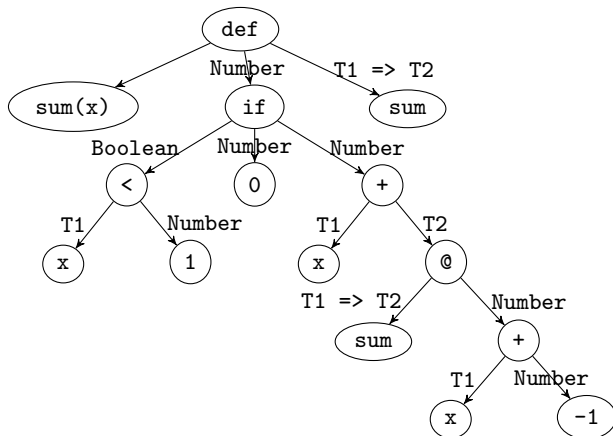
X	T
sum	$T1 \Rightarrow T2$

Solution

X_α	T
T1	Number
T2	Number

The type of sum is $T1 \Rightarrow T2$. Using the solution inferred by the collected constraints, we can instantiate it to $Number \Rightarrow Number$.

Example 1 – sum



Type Environment

X	T
sum	$T1 \Rightarrow T2$

Solution

X_α	T
$T1$	Number
$T2$	Number

```
/* TRFAE */
```

```
def sum(x: Number): Number = if (x < 1) 0 else x + sum(x + -1);
```

```
sum
```

1. Example 1 – sum

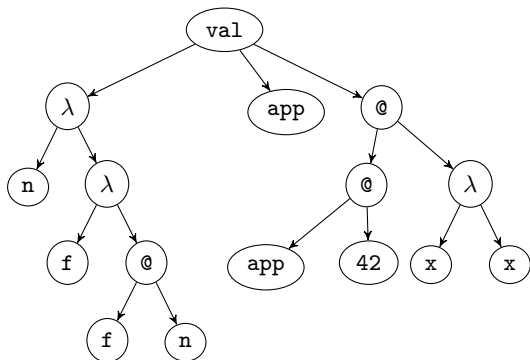
2. Example 2 – app

3. Example 3 – id

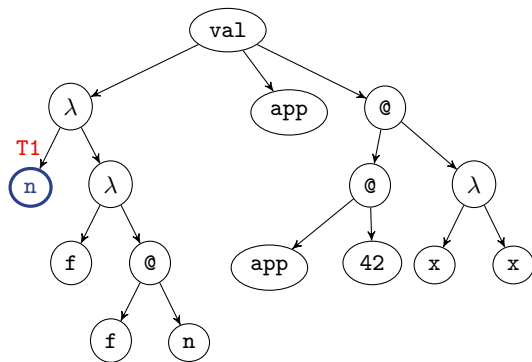
Example 2 – app

Let's infer the type of the following FAE expression:

```
/* FAE */  
val app = n => f => f(n);  
app(42)(x => x)
```



Example 2 – app



Type Environment

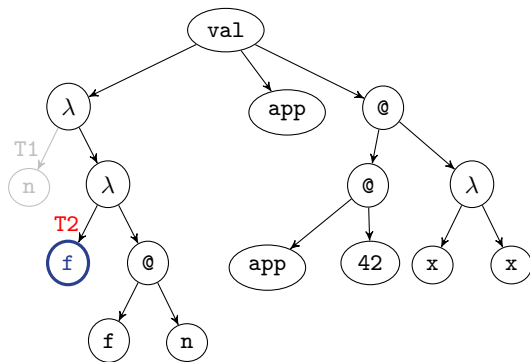
X	T
n	$T1$

Solution

X_α	T
$T1$	-

Let's define a new **type variable T1** for the parameter n .

Example 2 – app



Type Environment

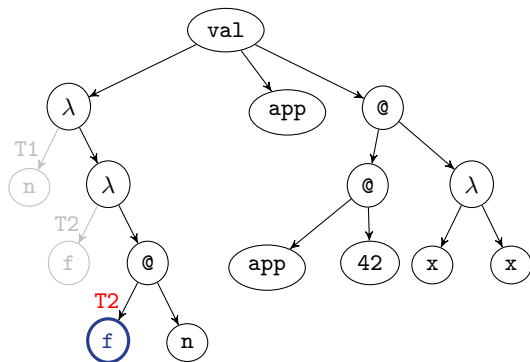
X	T
n	$T1$
f	$T2$

Solution

X_α	T
$T1$	-
$T2$	-

Let's define a new **type variable** $T2$ for the parameter f .

Example 2 – app



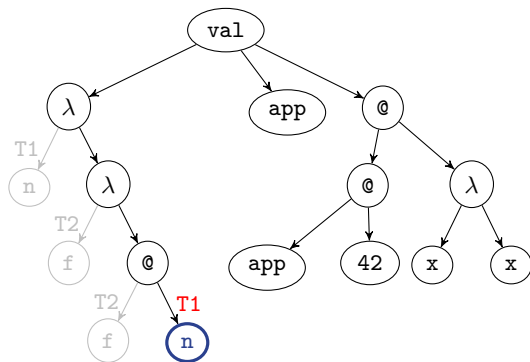
Type Environment

X	T
n	$T1$
f	$T2$

Solution

X_α	T
$T1$	–
$T2$	–

Example 2 – app



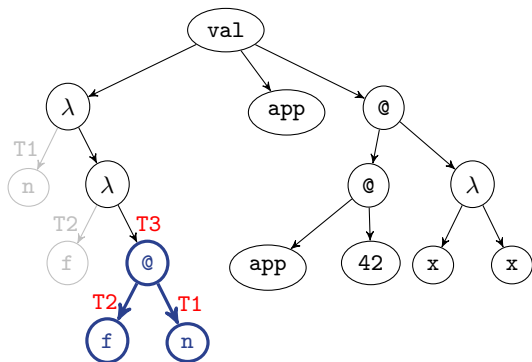
Type Environment

X	T
n	$T1$
f	$T2$

Solution

X_α	T
$T1$	-
$T2$	-

Example 2 – app



Type Environment

X	T
n	$T1$
f	$T2$

Solution

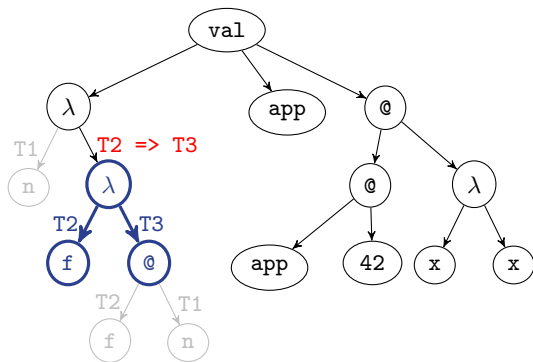
X_α	T
$T1$	-
$T2$	$T1 \Rightarrow T3$
$T3$	-

The type $T2$ of f should be in the form of $T1 \Rightarrow ???$.

Let's define a new **type variable** $T3$ for $???$ (the return type of f).

So, we collected a **type constraint**: $T2 == T1 \Rightarrow T3$.

Example 2 – app



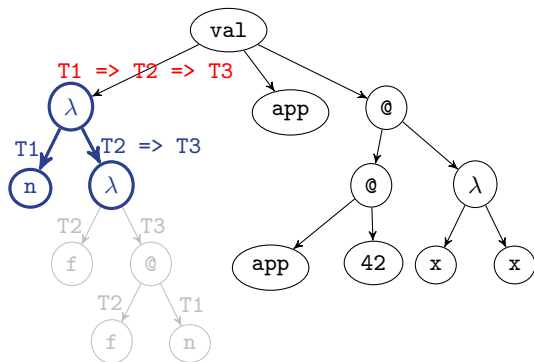
Type Environment

X	T
n	$T1$

Solution

X_α	T
$T1$	-
$T2$	$T1 \Rightarrow T3$
$T3$	-

Example 2 – app



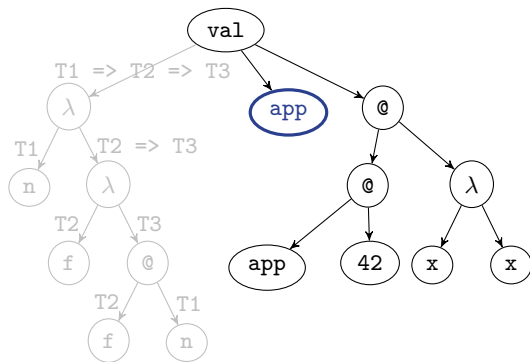
Type Environment

X	T

Solution

X_α	T
$T1$	-
$T2$	$T1 \Rightarrow T3$
$T3$	-

Example 2 – app



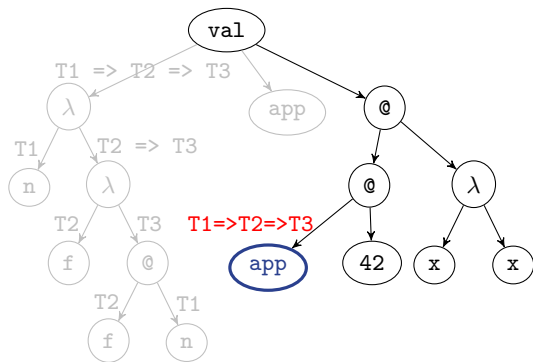
Type Environment

X	T
<code>app</code>	$T1 \Rightarrow T2 \Rightarrow T3$

Solution

X_α	T
$T1$	-
$T2$	$T1 \Rightarrow T3$
$T3$	-

Example 2 – app



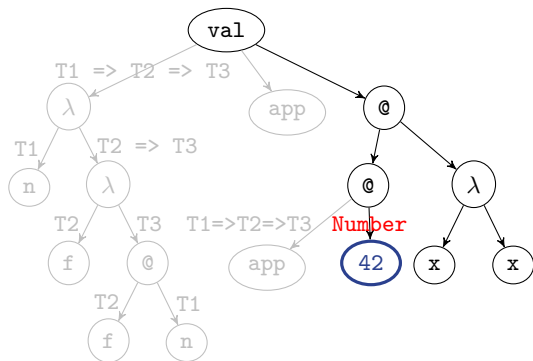
Type Environment

X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$

Solution

X_α	T
T1	-
T2	$T1 \Rightarrow T3$
T3	-

Example 2 – app



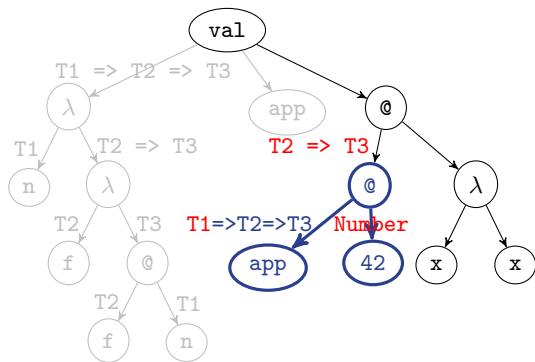
Type Environment

X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$

Solution

X_α	T
T1	-
T2	$T1 \Rightarrow T3$
T3	-

Example 2 – app



Type Environment

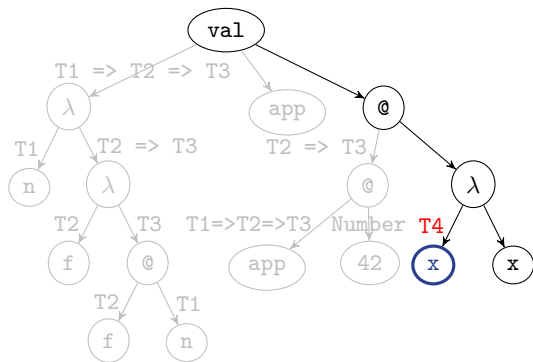
X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$

Solution

X_α	T
$T1$	Number
$T2$	$T1 \Rightarrow T3$
$T3$	-

The **parameter type** **T1** should be equal to the **argument type** **Number**.
 So, we collected a **type constraint**: **T1 == Number**.

Example 2 – app



Type Environment

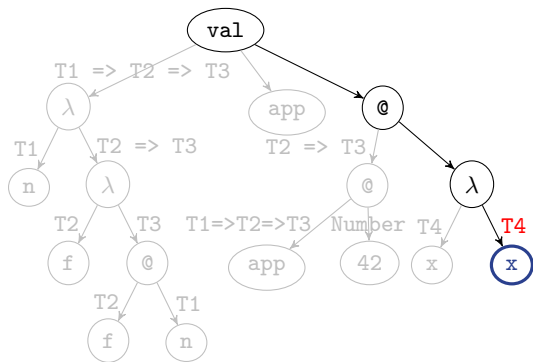
X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$
x	$T4$

Solution

X_α	T
$T1$	Number
$T2$	$T1 \Rightarrow T3$
$T3$	-
$T4$	-

Let's define a new **type variable $T4$** for the parameter x .

Example 2 – app



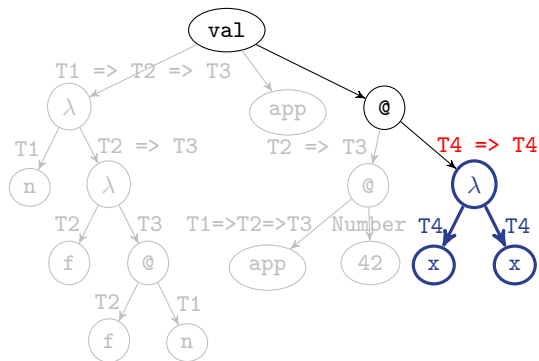
Type Environment

X	T
<code>app</code>	$T1 \Rightarrow T2 \Rightarrow T3$
<code>x</code>	$T4$

Solution

X_α	T
$T1$	<code>Number</code>
$T2$	$T1 \Rightarrow T3$
$T3$	-
$T4$	-

Example 2 – app



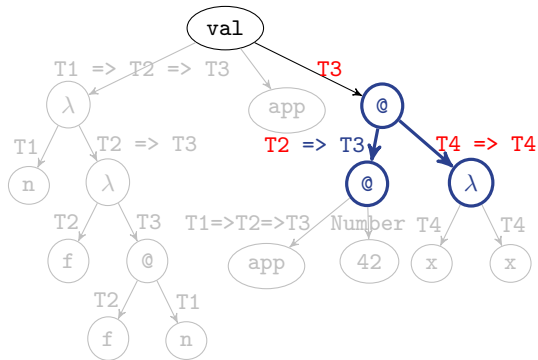
Type Environment

X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$
x	$T4$

Solution

X_α	T
T1	Number
T2	$T1 \Rightarrow T3$
T3	-
T4	-

Example 2 – app



Type Environment

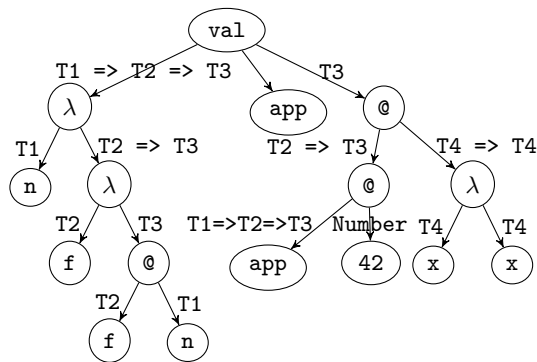
X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$
x	$T4$

Solution

X_α	T
T1	Number
T2	$T1 \Rightarrow T3$
T3	Number
T4	Number

The **parameter type** $T2$ should be equal to **argument type** $T4 \Rightarrow T4$.
 We collected **type constraints**: $T3 == \text{Number}$ and $T4 == \text{Number}$.
 Finally, the entire expression has type $T3$ ($= \text{Number}$).

Example 2 – app



Type Environment

X	T
app	$T1 \Rightarrow T2 \Rightarrow T3$
x	$T4$

Solution

X_α	T
T1	Number
T2	$T1 \Rightarrow T3$
T3	Number
T4	Number

```
/* TFAE */
```

```
val app = (n: Number) => (f: Number => Number) => f(n)
app(42)((x: Number) => x)
```

1. Example 1 – sum

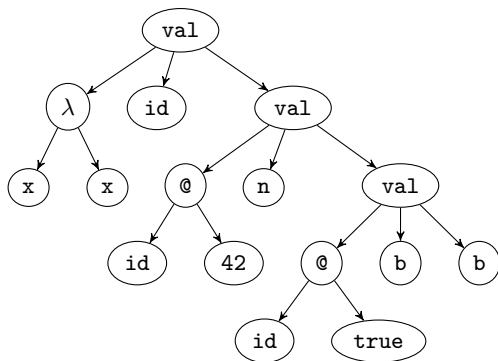
2. Example 2 – app

3. Example 3 – id

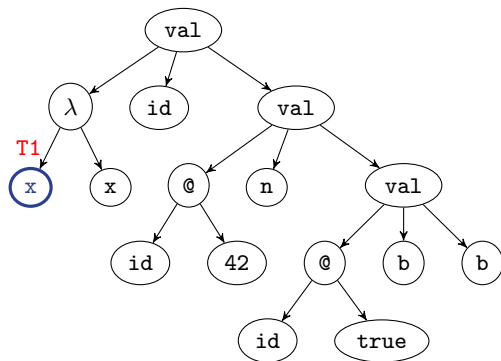
Example 3 – id

Let's infer the type of the following FAE expression:

```
/* FAE */  
val id = x => x;  
val n = id(42);  
val b = id(true);  
b
```



Example 3 – id



Type Environment

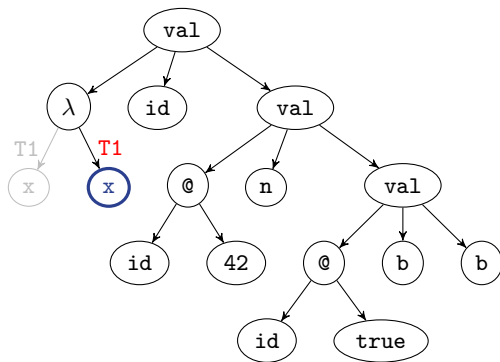
X	T
x	$T1$

Solution

X_α	T
$T1$	-

Let's define a new **type variable T1** for the parameter x .

Example 3 – id



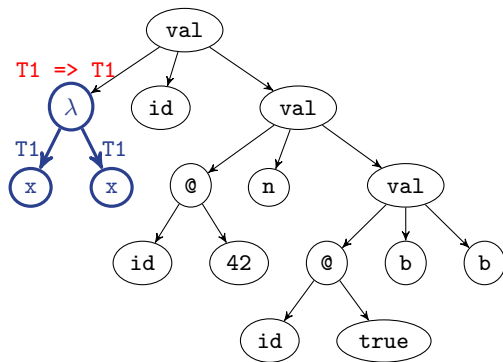
Type Environment

X	T
x	$T1$

Solution

X_{α}	T
$T1$	-

Example 3 – id



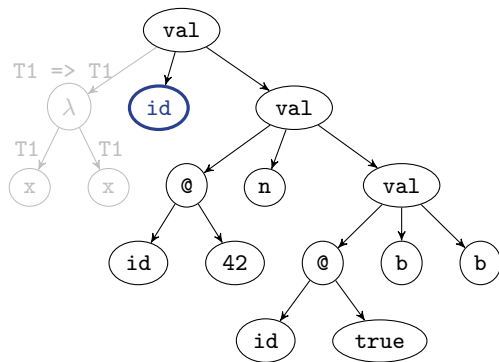
Type Environment

X	T

Solution

X_{α}	T
$T1$	-

Example 3 – id



Type Environment

X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$

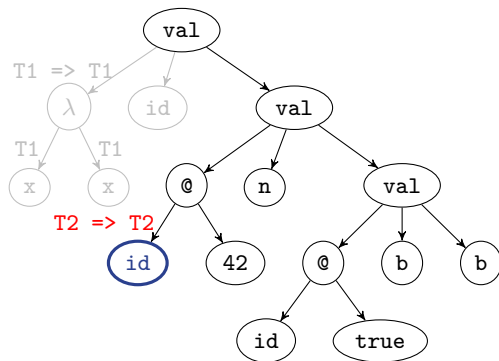
Solution

X_α	T

Let's **generalize** the type $T1 \Rightarrow T1$ into a **polymorphic type** for `id` with **type variable** $T1$ as a **type parameter**.

We call this **let-polymorphism** because it only introduces polymorphism for the let-binding (e.g., `val`).

Example 3 – id



Type Environment

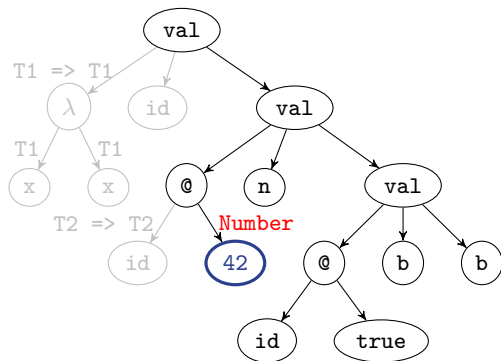
X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$

Solution

X_α	T
$T2$	-

Let's define a new **type variable** $T2$ to **instantiate** the **type variable** $T1$.
And, **substitute** $T1$ with $T2$.

Example 3 – id



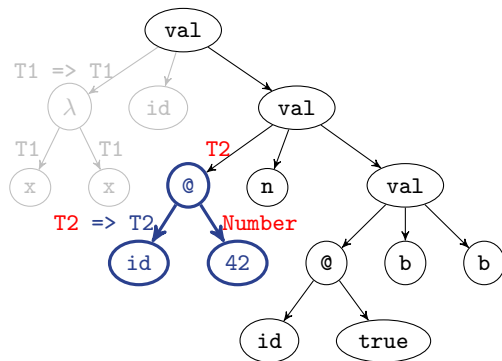
Type Environment

X	T
id	[T1] { T1 => T1 }

Solution

X_{α}	T
T2	-

Example 3 – id



Type Environment

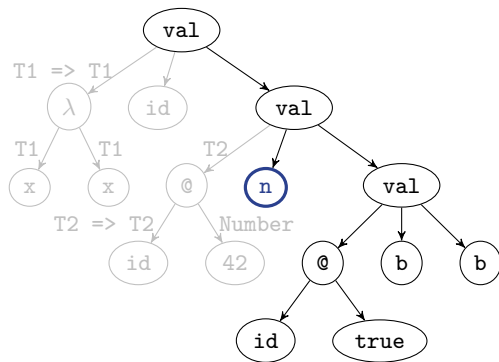
X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$

Solution

X_α	T
T2	Number

The **parameter type** **T2** should be equal to **argument type** **Number**.
 We collected a **type constraint**: **T2 == Number**.

Example 3 – id



Type Environment

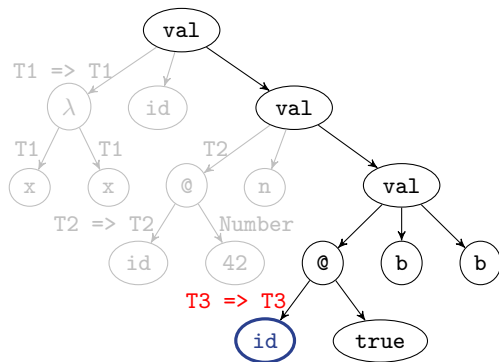
X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$
n	$T2$

Solution

X_α	T
$T2$	Number

$T2$ is not a free type variable because it actually represents **Number**.
So, we will not introduce a polymorphic type in this case.

Example 3 – id



Type Environment

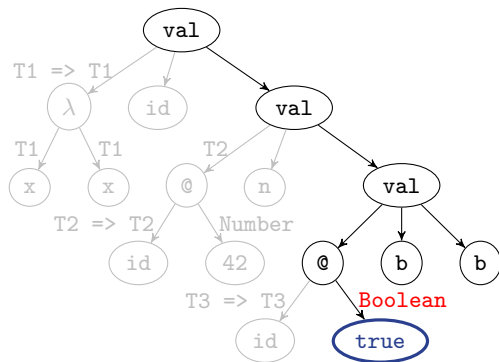
X	T
<code>id</code>	$[T1] \{ T1 \Rightarrow T1 \}$
<code>n</code>	$T2$

Solution

X_α	T
$T2$	<code>Number</code>
$T3$	<code>-</code>

Let's define a new **type variable** $T3$ to **instantiate** the **type variable** $T1$.
And, **substitute** $T1$ with $T3$.

Example 3 – id



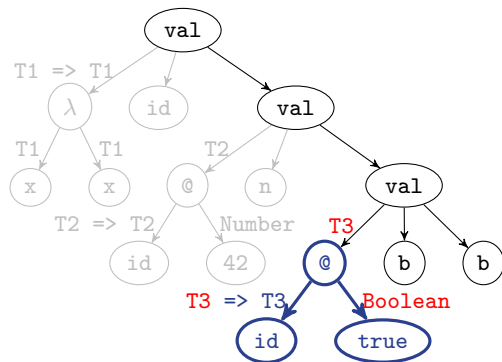
Type Environment

X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$
n	$T2$

Solution

X_α	T
$T2$	Number
$T3$	-

Example 3 – id



Type Environment

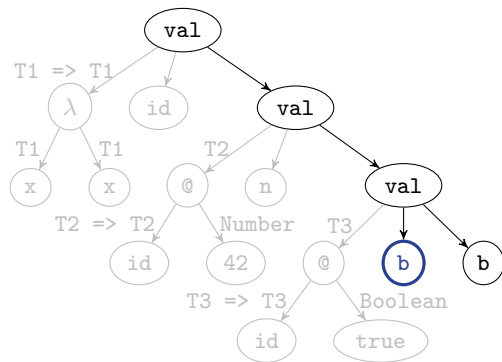
X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$
n	$T2$

Solution

X_{α}	T
$T2$	Number
$T3$	Boolean

The **parameter type** $T3$ should be equal to **argument type** **Boolean**.
 We collected a **type constraint**: $T3 == \text{Boolean}$.

Example 3 – id



Type Environment

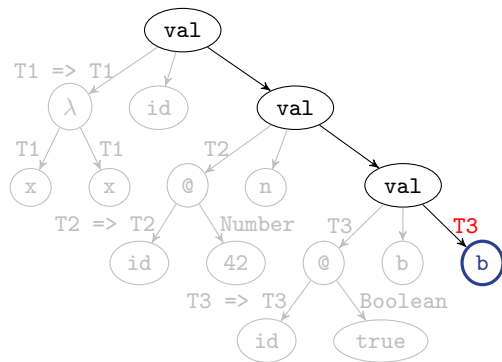
X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$
n	$T2$
b	$T3$

Solution

X_α	T
$T2$	Number
$T3$	Boolean

T3 is not a free type variable because it actually represents **Boolean**. So, we will not introduce a polymorphic type in this case.

Example 3 – id



Type Environment

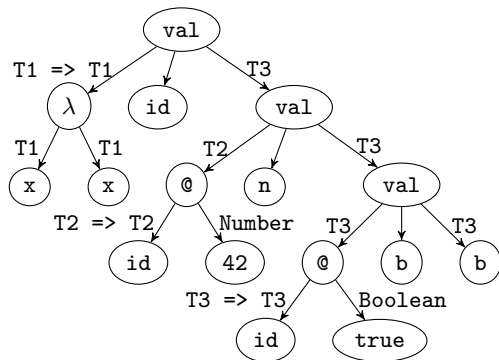
X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$
n	$T2$
b	$T3$

Solution

X_α	T
$T2$	Number
$T3$	Boolean

Finally, the entire expression has type $T3$ (= Boolean).

Example 3 – id



Type Environment

X	T
id	$[T1] \{ T1 \Rightarrow T1 \}$
n	$T2$
b	$T3$

Solution

X_α	T
$T2$	Number
$T3$	Boolean

```
/* PTFAE */
val id = forall[T] { (x: T) => x }
val n = id[Number](42)
val b = id[Boolean](true)
b
```

1. Example 1 – sum

2. Example 2 – app

3. Example 3 – id

- Type Inference (2)

Jihyeok Park

jihyeok_park@korea.ac.kr

<https://plrg.korea.ac.kr>