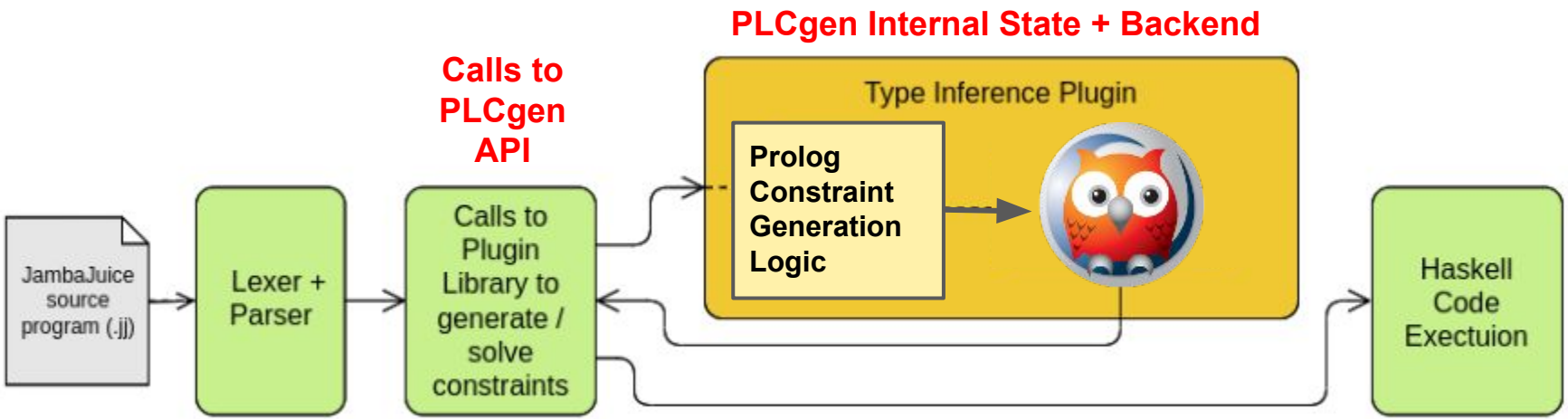
JambaJuice

A small, functional language with modular type inference

*PLCGen = Prolog Constraint Generation



An example JambaJuice Program

```
jambatime fib n = {
    if n == 0 { 0 }
    else {
        if n == 1 { 1 }
        else { fib (n - 1) + fib (n - 2) }
    }
}
```

```
fact = {
fix \fact -> \b -> (if b == 0 {1} else {b
* fact (b - 1)})
}
```

```
jambajuice = {
 let hi = fib 15 in { // 610
    let sup = fact 5 in { // 120
        if (hi > sup)
            { (\x -> x + 2) hi }
        else { sup }
    }
 }
}
```

Core Language

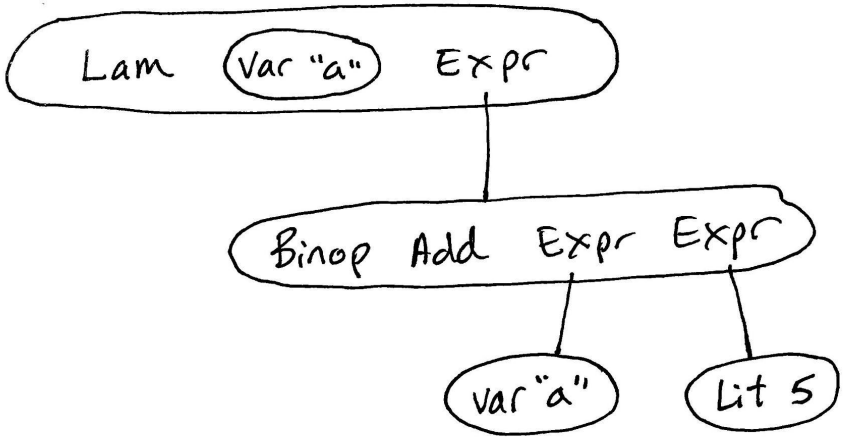
```
type Var = String


data Expr
  = Var Var
  | App Expr Expr
  | Lam Var Expr
  | Let Var Expr Expr
  | Lit Lit
  | If Expr Expr Expr
  | Fix Expr
  | Op Binop Expr Expr
```

```
data Lit
  = LInt Integer
  | LBool Bool


data Binop = Add | Sub | Mul |
Eql | Neq | Lt | Le | Gt | Ge


type Decl = (Var, Expr)


data Program = Program [Decl]
```

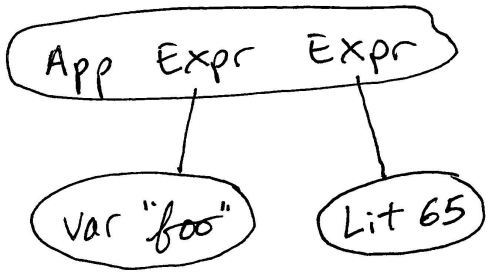
Library walkthrough

```
foo a = {
    a + 5
}

jambajuice = {
    foo 65
}
```

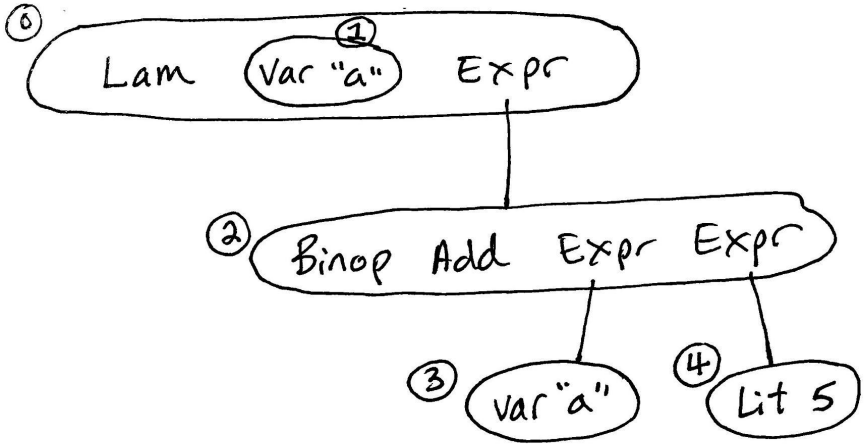
```
foo a = {
    a + 5
}
```



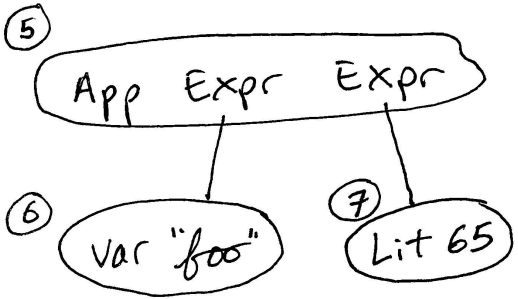

```
jambajuice = {
    foo 65
}
```

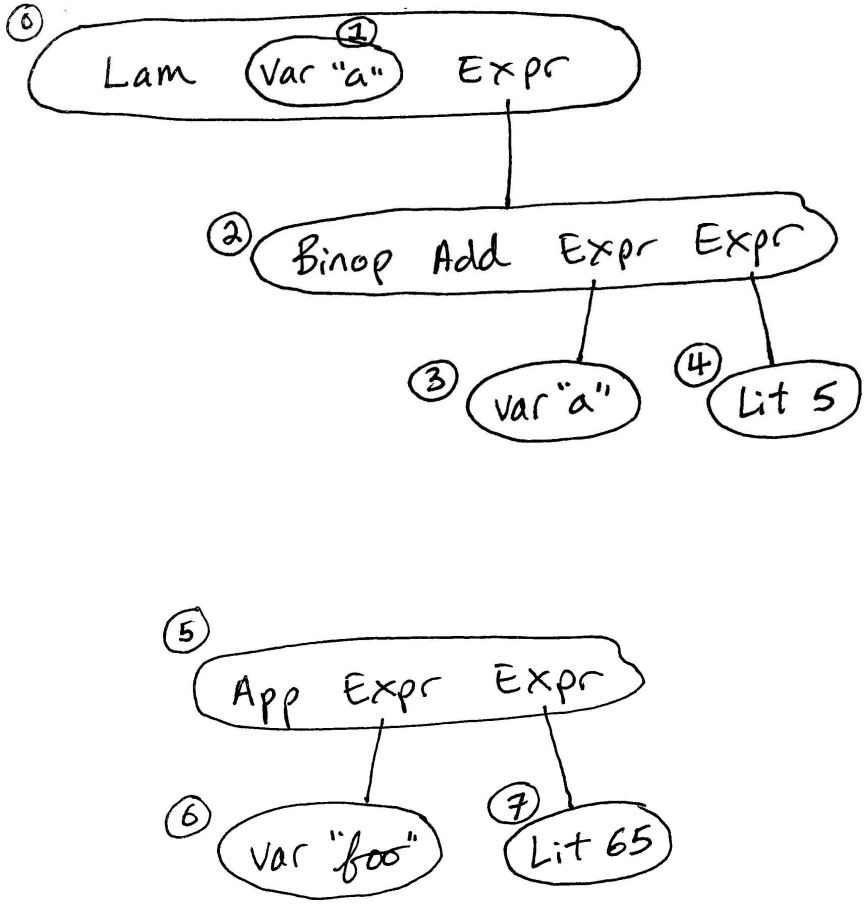


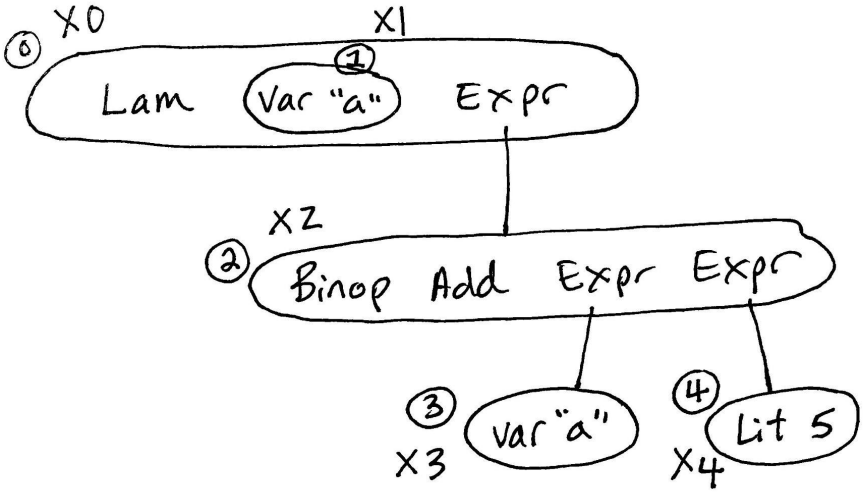
```
foo a = {
    a + 5
}
```

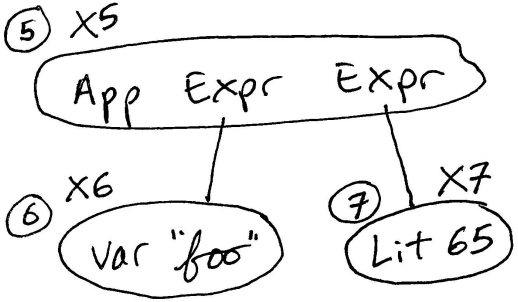



```
jambajuice = {
    foo 65
}
```

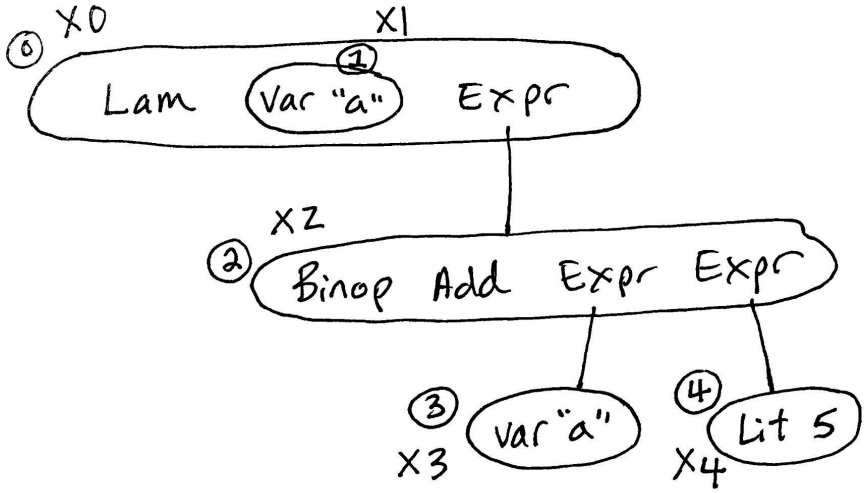


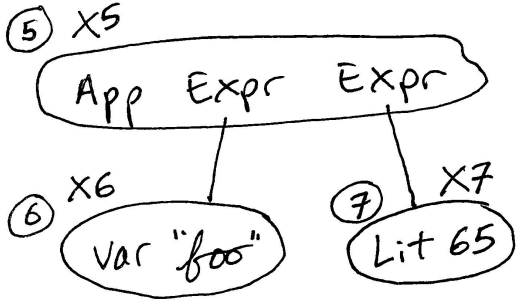

0
Lam
1
Var "a"
Expr
2
Binop Add Expr Expr
3
Var "a"
4
Lit 5
5
App Expr Expr
6
Var "foo"
7
Lit 65

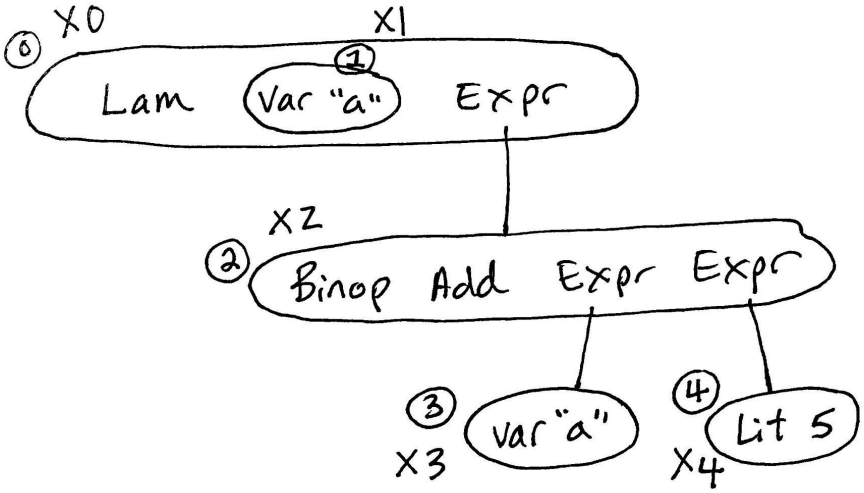




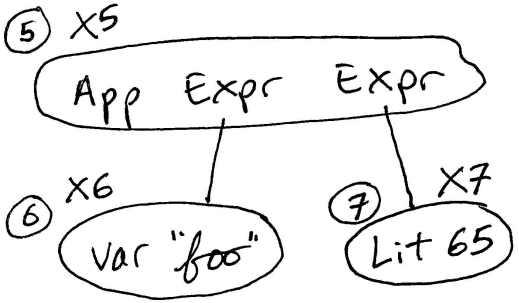
NodeID	Type Variable
0	X0
1	X1
2	X2
3	X3
4	X4
5	X5
6	X6
7	X7


0
X0
X1
1
Lam
Var "a"
Expr
X2
2
Binop Add Expr Expr
3
Var "a"
X3
4
Lit 5
X4

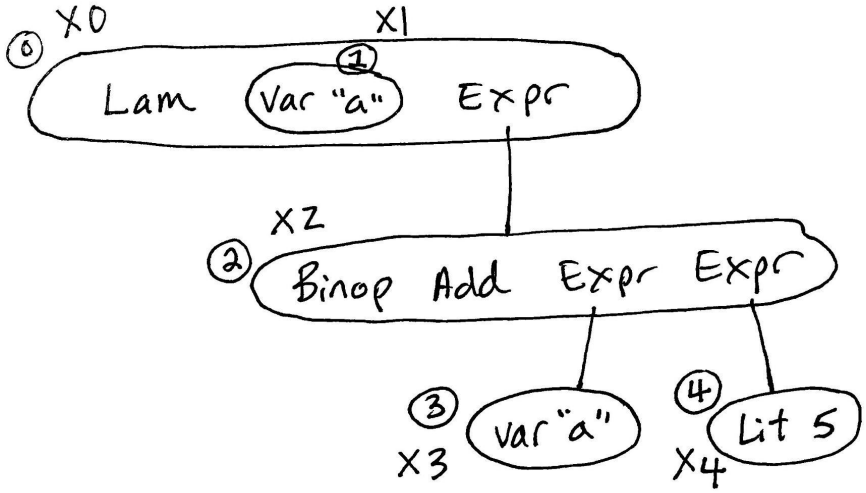

5
X5
App Expr Expr
6
X6
Var "foo"
7
X7
Lit 65



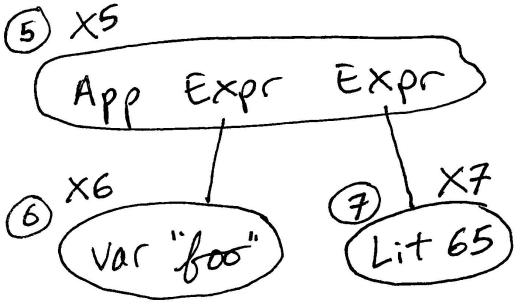

```
hasType(foo , X0):-
foo_typechecks(X0,X1,X2,X3,X4).
```



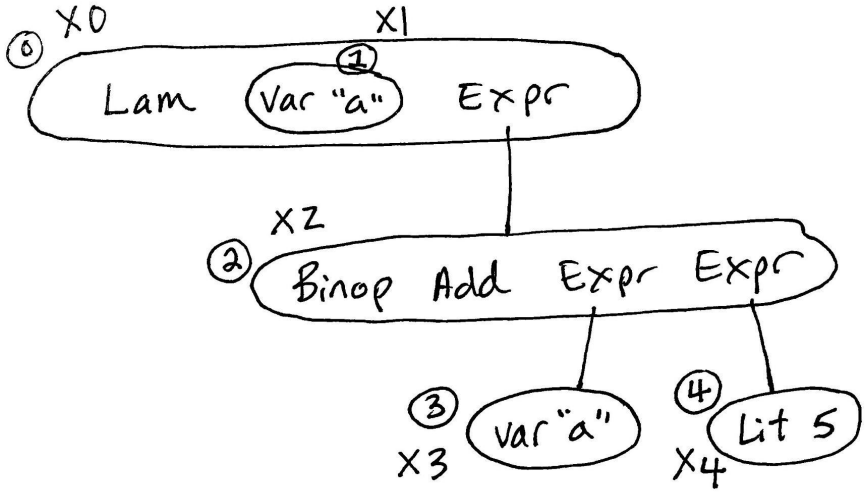

```
hasType(jambajuice , X5):-
jambajuice_typechecks(X5,X6,X7).
```



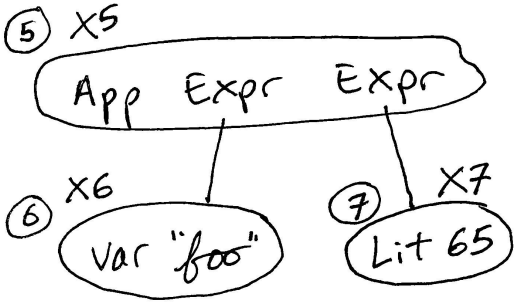

```
hasType(foo , X0):-
foo_typechecks(X0,X1,X2,X3,X4).

foo_typechecks(X0,X1,X2,X3,X4):-
arrow(X0),fst(X0,X1),snd(X0,X2),
X1=X1,
X3=X4,X4=X2,X2=int,
X3=X1,
X4=int.
```



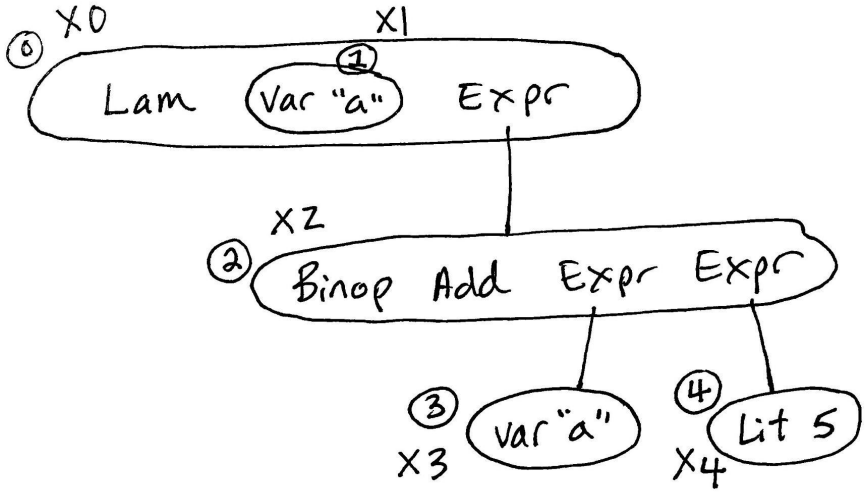

```
hasType(jambajuice , X5):-
jambajuice_typechecks(X5,X6,X7).
```

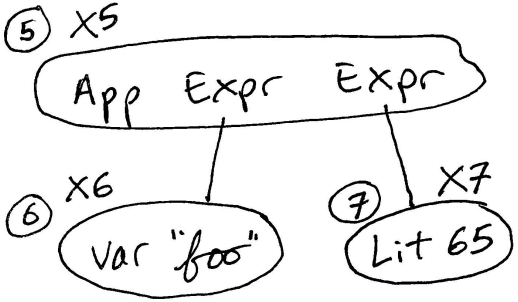



```
hasType(foo , X0):-
foo_typechecks(X0,X1,X2,X3,X4).

foo_typechecks(X0,X1,X2,X3,X4):-
arrow(X0),fst(X0,X1),snd(X0,X2),
X1=X1,
X3=X4,X4=X2,X2=int,
X3=X1,
X4=int.
```

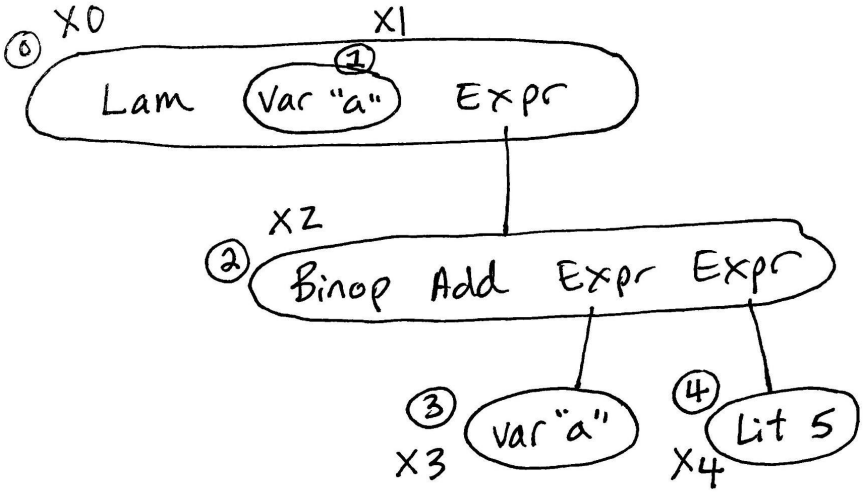


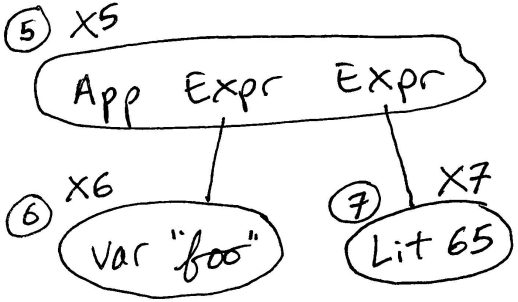

```
hasType(jambajuice , X5):-
jambajuice_typechecks(X5,X6,X7).

jambajuice_typechecks(X5,X6,X7):-
arrow(X6),fst(X6,X7),snd(X6,X5),
X7=int,
instantiates(X6,foo).
```






```
node_0 [int,int]
node_1 int
node_2 int
node_3 int
node_4 int
node_5 int
node_5 int
node_6 [int,int]
node_6 [int,int]
node_7 int
node_7 int
```





NodeID	Type
0	Int -> Int
1	Int
2	Int
3	Int
4	Int
5	Int
6	Int -> Int
7	Int

Works Cited

[1] Stephen Diehl. 2015. Write You A Haskell. https://smunix.github.io/dev.stephendiehl.com/fun/index.html

[2] Stephen A. Edwards. 2023. The Hindley-Milner Type System. http://www.cs.columbia.edu/~sedwards/classes/2023/6998-spring-tlc/hindleymilner.pdf

[3] Benjamin Lerner. 2019. Lecture 11: Type Inference. https://course.ccs.neu.edu/cs4410sp19/lec_type-inference_notes.html

[4] Benjamin C. Pierce. 2007. Types and Programming Languages. The MIT Press.

Demo!