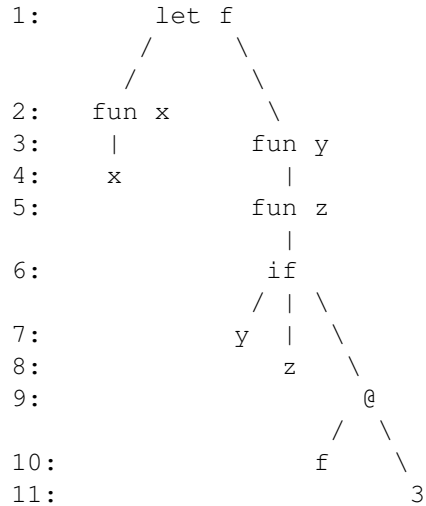


**CS558 Programming Languages – Fall 2023 – Suggested Study Question Solutions for Lecture 7b**

1. (a)

Here is the AST, with one node on each numbered line (arbitrarily numbered breadth-first).



From this tree, we generate the following constraints:

Node #	Rule	Constraint
1	let	$t_2 = t_f$ and $t_1 = t_3$
2	fun	$t_2 = t_x \rightarrow t_4$
3	fun	$t_3 = t_y \rightarrow t_5$
4	var	$t_4 = t_x$
5	fun	$t_5 = t_z \rightarrow t_6$
6	if	$t_7 = \text{Bool}$ and $t_6 = t_8 = t_9$
7	var	$t_7 = t_y$
8	var	$t_8 = t_z$
9	app	$t_{10} = t_{11} \rightarrow t_9$
10	var	$t_{10} = t_f$
11	int	$t_{11} = \text{Int}$

We can solve this by inspection:

First, using the identities for  $t_2, t_4, t_7, t_8, t_{10}, t_{11}$  we can substitute for these variables, leading to the following modified constraints:

- 2'  $t_f = t_x \rightarrow t_x$
- 6'  $t_y = \text{Bool}$  and  $t_6 = t_z = t_9$
- 9'  $t_f = \text{Int} \rightarrow t_9$

Using  $t_1 = t_3$ , we can substitute for  $t_3$  to get the modified constraint

- 3'  $t_1 = t_y \rightarrow t_5$

(Choosing whether to get rid of  $t_1$  or  $t_3$  is fairly arbitrary, but we ultimately want to know the root expression type  $t_1$ , so we keep that.)

Similarly, from  $t_6 = t_z = t_9$ , we can substitute for  $t_6$  and  $t_9$  (again fairly arbitrary, but we ultimately want to know  $t_z$ ), getting

```
5'      t5 = tz -> tz
9''     tf = Int -> tz
```

Equating the two expressions (2' and 9'') for  $t_f$ , we get that

```
tx -> tx = Int -> tz
```

which implies that  $t_x = t_z = \text{Int}$ .

Summarizing, we have

```
t1 = Bool -> (Int -> Int)
tf = Int -> Int
tx = Int
ty = Bool
tz = Int
```

(b)

The AST:

```
1:  fun f
    |
2:  fun g
    |
3:  fun x
    |
4:  @
   / \
5: f   \
6:     @
       / \
7:  g   \
8:     x
```

The generated constraints:

Node #	Rule	Constraint
1	fun	$t_1 = t_f \rightarrow t_2$
2	fun	$t_2 = t_g \rightarrow t_3$
3	fun	$t_3 = t_x \rightarrow t_4$
4	app	$t_5 = t_6 \rightarrow t_4$

```

5      var      t5 = tf
6      app      t7 = t8 -> t6
7      var      t7 = tg
8      var      t8 = tx

```

Solution by inspection:

Using the identities from nodes 5, 6, and 7, we can rewrite the constraints for nodes 4 and 6 as

```

4'      tf = t6 -> t4
6'      tg = tx -> t6

```

There are no other constraints on  $tx$ ,  $t4$  and  $t6$ , so the overall type must be parametric in (i.e. polymorphic over) these types. The type of the overall expression is

```

t1 = tf -> t2          (by 1)
    = tf -> (tg -> t3)  (by 2)
    = tf -> (tg -> (tx -> t4)) (by 3)
    = (t6 -> t4) -> ((tx -> t6) -> (tx -> t4)) (by 4' and 6')

```

Or, using more suggesting names for the polymorphic types, and the convention that  $\rightarrow$  associates to the right:

```

t1 = (tb -> tc) -> (ta -> tb) -> (ta -> tc)
tx = ta
tf = tb -> tc
tg = ta -> tb

```

which makes sense for a general-purpose “compose” function.