

CS 320

Computer Language Processing

Exercise Set 5

April 02, 2025

Consider a type system for a simple functional language, consisting of integers, booleans, parametric pairs, and lists. The rest of the exercises will revolve around this system.

$$\begin{array}{c}
\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{ (var)} \\
\frac{n \text{ is an integer value}}{\Gamma \vdash n : \text{int}} \text{ (int)} \\
\frac{e_1 : \text{int} \quad e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} (+) \quad \frac{e_1 : \text{int} \quad e_2 : \text{int}}{\Gamma \vdash e_1 - e_2 : \text{int}} (-) \\
\frac{b \text{ is a boolean value}}{\Gamma \vdash \text{bool}(b) : \text{bool}} \text{ (bool)} \\
\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \wedge e_2 : \text{bool}} \text{ (and)} \quad \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \vee e_2 : \text{bool}} \text{ (or)} \\
\frac{\Gamma \vdash e_1 : \text{bool}}{\Gamma \vdash \neg e_1 : \text{bool}} \text{ (not)} \\
\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : \text{bool}} \text{ (eq)} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \leq e_2 : \text{bool}} \text{ (lte)} \\
\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau} \text{ (ite)} \\
\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : (\tau_1, \tau_2)} \text{ (pair)} \\
\frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash \text{fst}(e) : \tau_1} \text{ (fst)} \quad \frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash \text{snd}(e) : \tau_2} \text{ (snd)} \\
\frac{}{\Gamma \vdash \text{Nil}() : \text{List}[\tau]} \text{ (nil)} \quad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \text{List}[\tau]}{\Gamma \vdash \text{Cons}(e_1, e_2) : \text{List}[\tau]} \text{ (cons)} \\
\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . e : \tau_1 \Rightarrow \tau_2} \text{ (fun)} \quad \frac{\Gamma \vdash e_1 : \tau_1 \Rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \text{ (app)}
\end{array}$$

Exercise 1 For each of the following term-type pairs (t, τ) , check whether the term can be ascribed with the given type, i.e., whether there exists a derivation of $\Gamma \vdash t : \tau$ for some typing context Γ in the given system. If not, briefly argue why.

1. `x, bool`
2. `x + 1, int`
3. `(x && y) == (x <= 0), bool`
4. `f => x => y => f((x, y)):`
`((List[Int], Bool) => Int) => List[Int] => Bool => Int`
5. `Cons(x, x) : List[List[Int]]`

Exercise 2 A *program* is a top-level expression t accompanied by a set of user-provided function definitions. The program is well-typed if each of the function bodies conform to the type of the function, and the top-level expression is well-typed in the context of the function definitions.

For each of the following function definitions, check whether the function body is well-typed:

1. `def f(x:Int)(y:Int):Bool = x <= y`
2. `def rec(x:Int):Int = rec(x)`
3. `def fib(n:Int):Int = if n <= 1 then 1 else (fib(n - 1) + fib(n - 2))`

Exercise 3 Consider the following term t :

$$t = 1 \Rightarrow \text{map}(1)(x \Rightarrow \text{fst}(x)(\text{snd}(x)) + \text{snd}(x))$$

where `map` is a function with type $\forall \tau, \pi. \text{List}[\tau] \Rightarrow (\tau \Rightarrow \pi) \Rightarrow \text{List}[\pi]$.

1. Label and assign type variables to each subterm of t .
2. Generate the constraints on the type variables, assuming t is well-typed, to infer the type of t .
3. Solve the constraints via unification to deduce the type of t .

Exercise 4 Consider the following definition for a recursive function g :

```
def g(n)(x) = if n <= 2 then (x, x) else (x, g(n - 1)(x))
```

1. Evaluate $g(3)(1)$ and $g(4)(2)$ using the definition of g . Suggest a type for the function g based on your observations.
2. Label and assign type variables to the definition parameters, body, and its subterms.
3. Generate the constraints on the type variables, assuming the definition of g is well-typed.
4. Attempt to solve the generated constraints via unification. Argue how the result correlates to your observations from evaluating g .