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.

$$\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{ (var)}$$

$$\frac{n \text{ is an integer value}}{\Gamma \vdash \mathbf{num}(n) : \texttt{int}} \text{ (int)}$$

$$\frac{e_1 : \texttt{int} \qquad e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}} \text{ (+)} \qquad \frac{e_1 : \texttt{int} \qquad e_2 : \texttt{int}}{\Gamma \vdash e_1 - e_2 : \texttt{int}} \text{ (-)}$$

$$\frac{b \text{ is a boolean value}}{\Gamma \vdash \texttt{bool}(b) : \texttt{bool}} \text{ (bool)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1 \land e_2 : \texttt{bool}} \text{ (and)} \qquad \frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1 \lor e_2 : \texttt{bool}} \text{ (or)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool}}{\Gamma \vdash \neg e_1 : \texttt{bool}} \text{ (not)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\, e_1\, \texttt{then}\, e_2\, \texttt{else}\, e_3 : \tau} \text{ (ite)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : (\tau_1, \tau_2)} \text{ (pair)}$$

$$\frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash \texttt{fst}(e) : \tau_1} \text{ (fst)} \qquad \frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash \texttt{snd}(e) : \tau_2} \text{ (snd)}$$

$$\frac{}{\Gamma \vdash \texttt{Nil()} : \texttt{List[}\tau\texttt{]}} \text{ (nil)} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \texttt{List[}\tau\texttt{]}}{\Gamma \vdash \texttt{Cons(}e_1, e_2\texttt{)} : \texttt{List[}\tau\texttt{]}} \text{ (cons)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1 \texttt{ => } e : \tau_1 \texttt{ => } \tau_2} \text{ (fun)} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \texttt{ => } \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau_2} \text{ (app)}$$

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. $\texttt{x}, \texttt{bool}$
2. $\texttt{x} + 1, \texttt{int}$
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:

$$t = \texttt{l => map(l, x =>fst(x)(snd(x))+ snd(x))}$$

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

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.