# CS 5510 Homework 9

Due: Wednesday, November 11th, 2015 11:59pm

## Part 1 — `true`, `false`, `=`, and `if`

Start with `typed-lambda.rkt`. The implementation already includes a `bool` type, but no expressions of `bool` type.

Add support for `true`, `false`, `{= <Expr> <Expr>}`, and `{if <Expr> <Expr> <Expr>}` expressions, where `=` produces a boolean given two numbers, and `if` requires a boolean expression for the test.

Examples:

```
(test (interp (parse '{if {= 13 {if {= 1 {+ -1 2}}
                                    12
                                    13}}
                          4
                          5})
              mt-env)
      (numV 5))

(test (typecheck (parse '{= 13 {if {= 1 {+ -1 2}}
                                   12
                                   13}})
                 mt-env)
      (boolT))

(test/exn (typecheck (parse '{+ 1 {if true true false}})
                     mt-env)
          "no type")
```

## Part 2 — Pairs

Implement `{cons <Expr> <Expr>}`, `{first <Expr>}`, and `{rest <Expr>}` expressions, as well as `{<Type> * <Type>}` types, as shown in video 10.

Examples (some of which depend on a choice of constructor and may not apply directly to your implementation):

```
(test (interp (parse '{cons 10 8})
              mt-env)
      ;; Your constructor might be different than consV:
      (consV (numV 10) (numV 8)))

(test (interp (parse '{first {cons 10 8}})
              mt-env)
      (numV 10))

(test (interp (parse '{rest {cons 10 8}})
              mt-env)
      (numV 8))

(test (typecheck (parse '{cons 10 8})
                 mt-env)
      ;; Your constructor might be different than crossT:
      (crossT (numT) (numT)))

(test (typecheck (parse '{first {cons 10 8}})
                 mt-env)
      (numT))

(test (typecheck (parse '{+ 1 {rest {cons 10 8}}})
                 mt-env)
      (numT))

(test (typecheck (parse '{lambda {[x : (num * bool)]}
                           {first x}})
                 mt-env)
      ;; Your constructor might be different than crossT:
      (arrowT (crossT (numT) (boolT)) (numT)))

(test (typecheck (parse '{{lambda {[x : (num * bool)]}
                            {first x}}
                          {cons 1 false}})
                 mt-env)
      (numT))

(test (typecheck (parse '{{lambda {[x : (num * bool)]}
                            {rest x}}
                          {cons 1 false}})
                 mt-env)
      (boolT))

(test/exn (typecheck (parse '{first 10})
                     mt-env)
          "no type")

(test/exn (typecheck (parse '{+ 1 {first {cons false 8}}})
                     mt-env)
          "no type")

(test/exn (typecheck (parse '{lambda {[x : (num * bool)]}
                               {if {first x}
                                   1
                                   2}})
                     mt-env)
          "no type")
```

## Part 3 — Functions that Accept Multiple Arguments, Yet Again

With pairs, functions can accept multiple arguments by accepting paired values, but we can also add direct support for multiple arguments.

Change the interpreter to allow multiple function arguments and multiple arguments at function calls. The grammar of the language is now as follows (not counting the `let` sugar, which you do not have to change):

```
<Expr> = <Num>
       | true
       | false
       | {+ <Expr> <Expr>}
       | {* <Expr> <Expr>}
       | {= <Expr> <Expr>}
       | <Sym>
       | {if <Expr> <Expr> <Expr>}
       | {lambda {[<Sym> : <Type>]*} <Expr>}
       | {<Expr> <Expr>*}
       | {cons <Expr> <Expr>}
       | {first <Expr>}
       | {rest <Expr>}
```

Examples:

```
(test (interp (parse '{{lambda {}
                        10}})
              mt-env)
      (numV 10))

(test (interp (parse '{{lambda {[x : num] [y : num]} {+ x y}}
                      10
                      20})
              mt-env)
      (numV 30))


(test (typecheck (parse '{{lambda {[x : num] [y : bool]} y}
                         10
                         false})
                 mt-env)
      (boolT))

(test/exn (typecheck (parse '{{lambda {[x : num] [y : bool]} y}
                             false
                             10})
                     mt-env)
          "no type")
```

---

Last update: Friday, November 6th, 2015
*mflatt@cs.utah.edu*