I05: More Examples: Twice Factorial

```
(def twice =  (f => x => (f (f x)))
 def fact n = (if n then (* n (fact (- n 1))) else 1)
 (twice fact 3))
~~> 720

(def twice1 =  (f => fact => (f (f fact)))
 def fact n = (if n then (* n (fact (- n 1))) else 1)
 (twice1 fact 3))

  FUN: (f => (fact => (f (f fact))))
  ARG: (n => (if n then (* n (fact (- n 1))) else 1))
  FUN: (fact => ((n => (if n then (* n (fact (- n 1))) else 1))
              ((n => (if n then (* n (fact (- n 1))) else 1)) fact)))
  ARG: 3
  (if 3 then (* 3 (3 (- 3 1))) else 1)
  (3 (- 3 1))
 java.lang.Exception: Cannot apply non-function 3 in a call
```

I05: Variable Capture

```
def subst(e: Expr, n: String, r: Expr): Expr = e match
  ...
  case Fun(formal,body) =>
       if formal==n then e  // do not substitute under (n => ...)
       else Fun(formal, subst(body,n,r))
```

The last line exhibits ***variable capture problem***.
If 'formal' occurs free in 'r', then it will be captured by Fun(formal,...) even though that outside occurrence of 'formal' in 'r' has nothing to do with the bound variable in the anonymous function.

```
  FUN: (f => (fact => (f (f fact))))
  ARG: (n => (if n then (* n (fact (- n 1))) else 1))
fact => ((n => (if n then (* n (fact (- n 1))) else 1))
           ((n => (if n then (* n (fact (- n 1))) else 1))  fact))
```

When we supply the integer argument we will also substitute it instead of the name of the factorial function, resulting in run-time error (or, in other cases, wrong result).

I05: We Want to Rename Bound Variable to Avoid Capture

In situation like this:

```
FUN: (f => (fact => (f (f fact))))
ARG: (n => (if n then (* n (fact (- n 1))) else 1))
```

when substituting ARG inside the body of FUN, we first rename bound variable in FUN body into one that does not occur in ARG:

```
FUN: (f => (fact' => (f (f fact'))))
ARG: (n => (if n then (* n (fact (- n 1))) else 1))
```

Instead of fact' we can choose any fresh name for bound variable!
Result then evaluates correctly:

```
fact' => ((n => (if n then (* n (fact (- n 1))) else 1))
          ((n => (if n then (* n (fact (- n 1))) else 1))  fact'))
```

I05: Renaming Bound Variables in Subst

We call our previous substitution ***naive substitution***:

```
def subst0(e: Expr, n: String, r: Expr): Expr = e match ...
  case Fun(formal,body) =>
       if formal==n then e else Fun(formal, subst0(body,n,r))
```

To avoid problems, we use ***capture-avoiding substitution***:

```
def subst(e: Expr, n: String, r: Expr): Expr = e match ...
  case Fun(formal,body) =>
       if formal==n then e else
         val fvs = freeVars(r)
         val (formal1, body1) =
           if fvs.contains(formal) then // rename bound formal
             val formal1 = differentName(formal, fvs)
             (formal1, subst0(body, formal, N(formal1)))
           else (formal, body)
         Fun(formal1, subst(body1,n,r)) // substitute either way
```

I05: Free Variables and Finding a Different Name

```
def differentName(n: String, s: Set[String]): String =
  if s.contains(n) then differentName(n + "'", s)
  else n

def freeVars(e: Expr): Set[String] = e match
  case C(c) => Set()
  case N(s) => Set(s)
  case BinOp(op, e1, e2) => freeVars(e1) ++ freeVars(e2)
  case IfNonzero(cond, trueE, falseE) =>
       freeVars(cond) ++ freeVars(trueE) ++ freeVars(falseE)
  case Call(f, arg) => freeVars(f) ++ freeVars(arg)
  case Fun(formal,body) => freeVars(body) - formal
```