Substitution model for defined procedure application

Function definition:

(**`define`** (<fun> <param1> <param2> ...)
  <body>)

Function call:

(<fun> <expr1> <expr2> ...)

The substitution model

- Evaluate expressions <expr1>, <expr2>, ...
- Substitute the value of <expr1> for <param1>, the value of <expr2> for <param2>, ... in a copy of the <body> expression in the definition of <fun> to make a new expression
- Evaluate that expression

Substitution model example

```
(define (square x) (* x x))
```

Evaluation of **`(square 2)`** by substitution model:

```
(square 2)
(* 2 2)
4
```

Second substitution example

```
(define (sum-of-squares x y)
  (+ (square x) (square y)))

(sum-of-squares 2 3)
(+ (square 2) (square 3))
   (* 2 2)
   4
              (* 3 3)
              9
(+ 4 9)
13
```

Third substitution example

```
(define (double-square x)
  (sum-of-squares x x))

(double-square 10)
(sum-of-squares 10 10)
(+ (square 10) (square 10))
   (* 10 10)
   100
               (* 10 10)
               100
(+ 100 100)
200
```

Applicative order and normal order

- Applicative order: evaluate arguments, then apply procedure to values
- Normal order: substitute argument expressions for corresponding parameters in body of procedure definition, then evaluate body

Our substitution model of evaluation uses applicative order

CAUTION!

- **Normal** procedures use **applicative order**
- **Special forms** effectively use **normal order**

(an accident of history)

Conditional statements

Two forms:
1) (**if** <test>
<then-expr>
<else-expr>) NOT optional in Scheme

Example:

```
(define (absolute x)
   (if (< x 0)
       (- x)
       x))
```

2) (**cond** (<test1> <expr1>)
(<test2> <expr2>)
. . .
(**else** <last-expr>)) NOT optional in Scheme

Example:

```
(define (absolute x)
   (cond ((> x 0) x)
         ((= x 0) 0)
         (else (- x))))
```

Comments on conditionals

- A test is considered to be true if it evaluates to anything except `#f`
- A branch of a cond can have more than one expression:

  (<test> <expr1> <expr2> . . . <exprN>)
- (<test>) returns value of <test> if it is not `#f`
- The else branch must contain at least one expression

(Confession)

Actually, I lied. The <else-expr> in **`if`** statements and the **`else`** branch in **`cond`** statements are optional, but the value that is returned is unspecified, so don't omit them.

Boolean functions

- (**and** <expr1> <expr2> . . . <exprN>)

  <expr>s evaluated in order; return `#f` if any evaluate to `#f`, else return value of <exprN>

- (**or** <expr1> <expr2> . . . <exprN>)

  <evpr>s evaluated in order; return the first value that is not `#f`; return `#f` if all are `#f`

- (**`not`** <expr>)

  returns **`#t`** or **`#f`** as appropriate

NOTE: **`define`**, **`if`**, **`cond`**, **`and`**, **`or`** are special forms