1. ‘let’ is a special form: when we declare a ‘let’ expression it is done in the next way:

(let ((variable init) …) expression …).

In contrary to standard forms, ‘let’ has some rules it must follow when evaluating the expression.

First, we must evaluate the ‘init’ variables, in each evaluation, we bind each ‘variable’ to it’s ‘init’. Then, we evaluate the expression(s) in the body. Those expressions are creating an extended environment to the ‘let’ expression, thus they are able to use the local variables that were created in the ‘let’ expression.

1. The role of the function is to support the apply-procedure and substitution of values back as expressions, we took care in the procedure to turn values into expressions with a special case for Literal Expressions that wrap SExp values.
2. In the case of
3. Fghh
4. An applicative evaluation for the next section of code:

(if #t 5 (/ 1 0)) will yield an error since we evaluate first the ‘then’ and ‘else’ and only then we apply the ‘if’ procedure. In addition, only one if the expressions will be returned, thus we don’t really need to evaluate both.

Normal evaluation will be a better approach here.

1. The next example taken from the github page:

(**define** square (**lambda** (**x**) (\* x x)))

(**define** sum-of-squares (**lambda** (**x** y) (+ (**square** x) (**square** y))))

(**define** f (**lambda** (**a**) (**sum-of-squares** (+ a 1) (\* a 2)))

(**f** 5)

Here with normal evaluation we will evaluate the same expression several times, while with applicative evaluation we will evaluate each expression no more than one time.

1. A) The danger in not renaming the bounded occurrences in a closure that has a free is that we might use this closure in another closure while using the substitution model, and in the latter there might be exactly the same variable name and now the free occurrence is bounded, which is obviously wrong, and might yield a wrong solution.

On the other hand, if there are no free occurrences, we don’t have this kind of danger, because all values in the body of the closure are bounded to the variable names declared in the argument array.

B) Given a closure, we check whether there are free occurrences.

If there are, we continue the substitution method as usual, renaming first.

Else, we substitute without renaming.