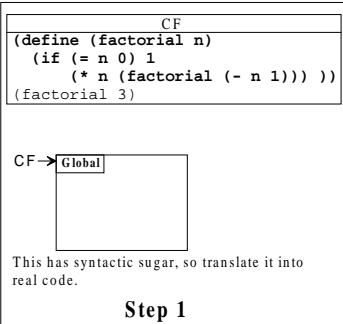


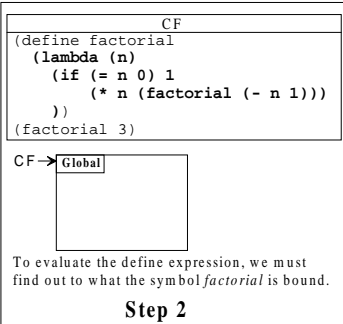
Evaluate the following:

```
> (define (factorial n)
  (if (= n 0) 1
      (* n (factorial (- n 1)))))
> (factorial 3)
```



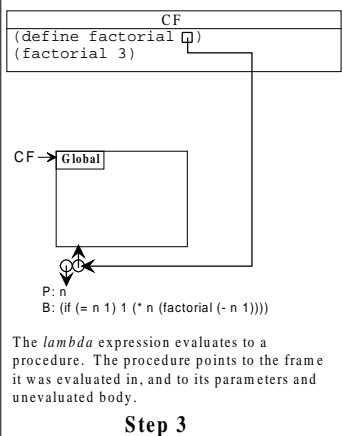
Step 1

This has syntactic sugar, so translate it into real code.



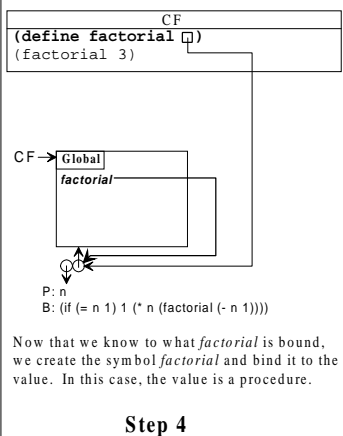
Step 2

To evaluate the define expression, we must find out to what the symbol *factorial* is bound.



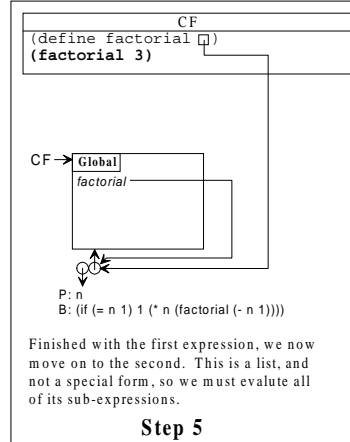
Step 3

The *lambda* expression evaluates to a procedure. The procedure points to the frame it was evaluated in, and to its parameters and unevaluated body.



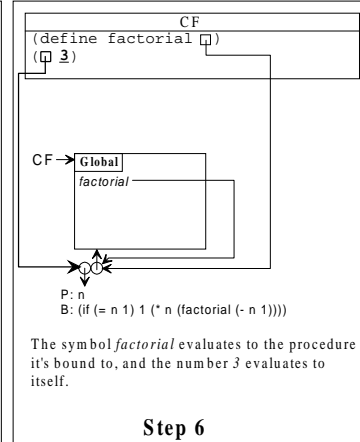
Step 4

Now that we know to what *factorial* is bound, we create the symbol *factorial* and bind it to the value. In this case, the value is a procedure.



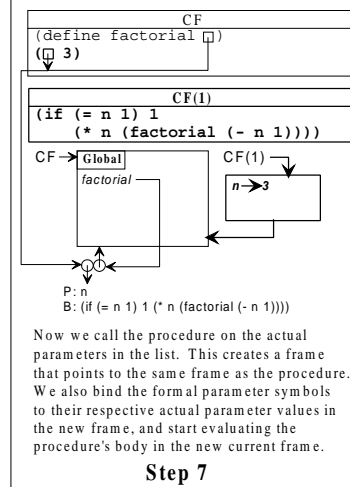
Step 5

Finished with the first expression, we now move on to the second. This is a list, and not a special form, so we must evaluate all of its sub-expressions.



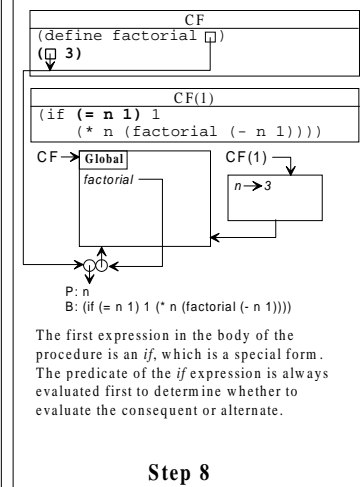
Step 6

The symbol *factorial* evaluates to the procedure it's bound to, and the number 3 evaluates to itself.



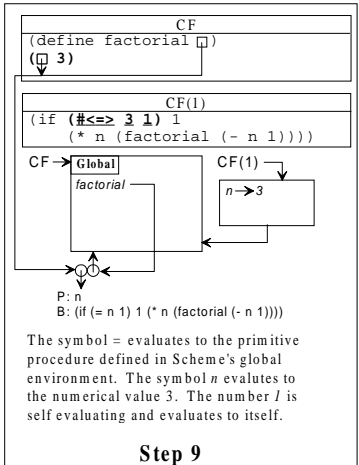
Step 7

Now we call the procedure on the actual parameters in the list. This creates a frame that points to the same frame as the procedure. We also bind the formal parameter symbols to their respective actual parameter values in the new frame, and start evaluating the procedure's body in the new current frame.



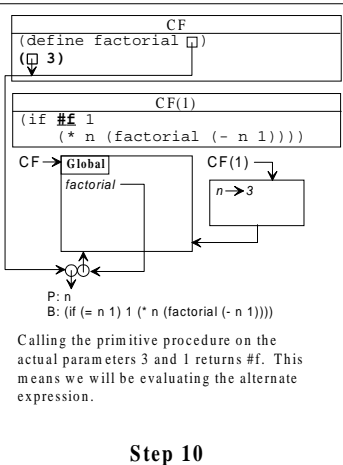
Step 8

The first expression in the body of the procedure is an *if*, which is a special form. The predicate of the *if* expression is always evaluated first to determine whether to evaluate the consequent or alternate.



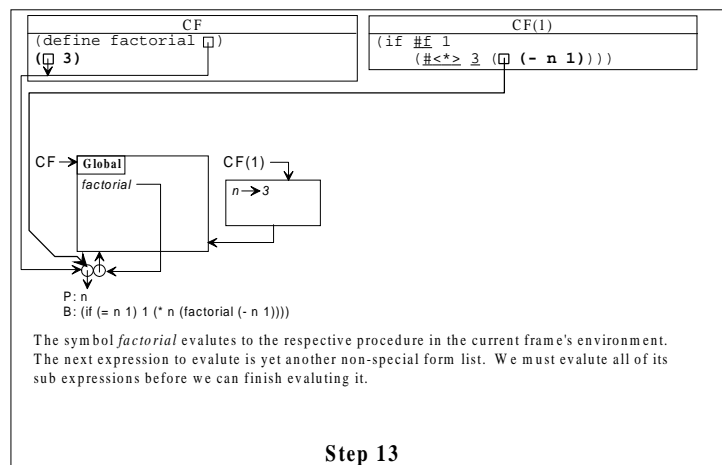
Step 9

The symbol = evaluates to the primitive procedure defined in Scheme's global environment. The symbol *n* evaluates to the numerical value 3. The number 1 is self evaluating and evaluates to itself.



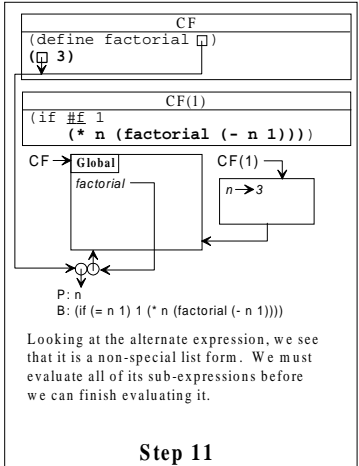
Step 10

Calling the primitive procedure on the actual parameters 3 and 1 returns #f. This means we will be evaluating the alternate expression.



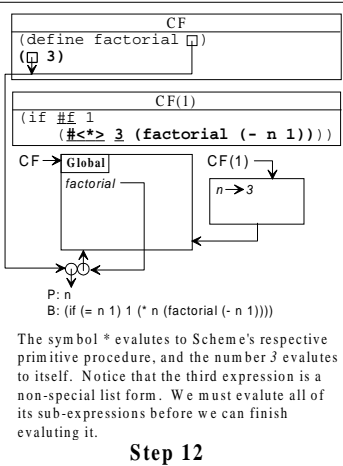
Step 13

The symbol *factorial* evaluates to the respective procedure in the current frame's environment. The next expression to evaluate is yet another non-special form list. We must evaluate all of its sub-expressions before we can finish evaluating it.



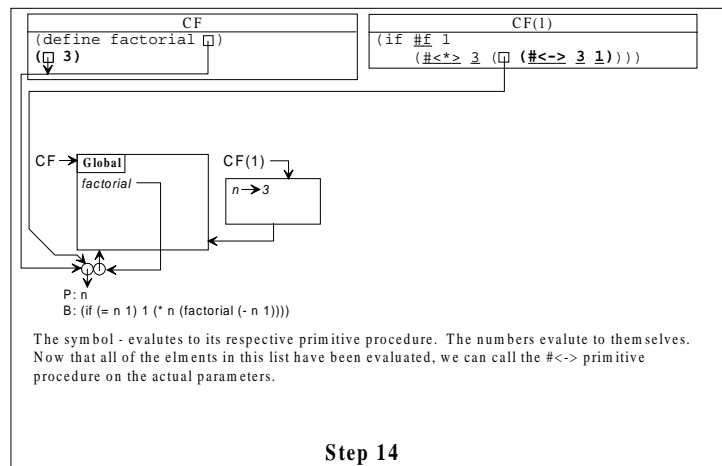
Step 11

Looking at the alternate expression, we see that it is a non-special list form. We must evaluate all of its sub-expressions before we can finish evaluating it.



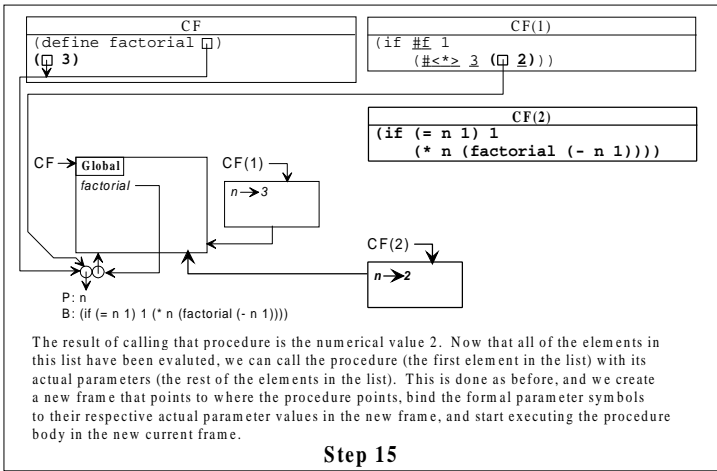
Step 12

The symbol * evaluates to Scheme's respective primitive procedure, and the number 3 evaluates to itself. Notice that the third expression is a non-special list form. We must evaluate all of its sub-expressions before we can finish evaluating it.

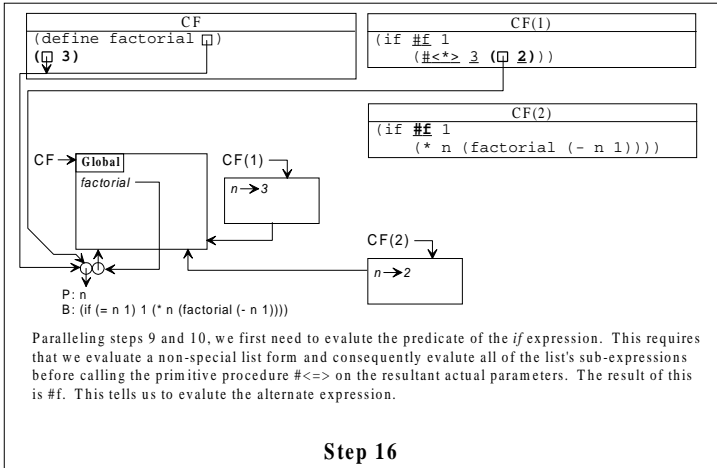


Step 14

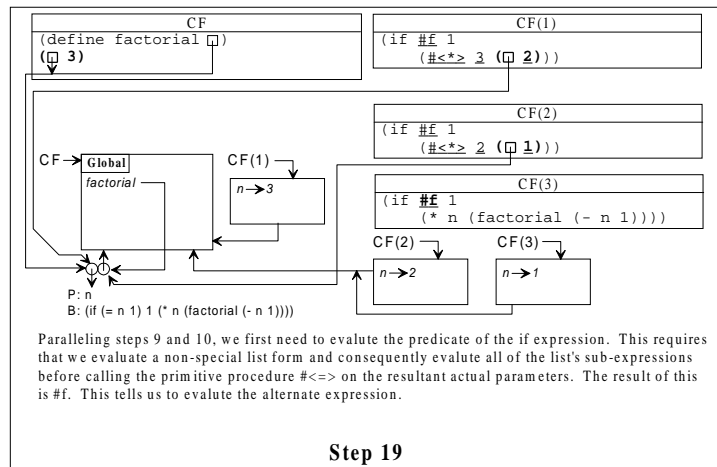
The symbol - evaluates to its respective primitive procedure. The numbers evaluate to themselves. Now that all of the elements in this list have been evaluated, we can call the #<=> primitive procedure on the actual parameters.



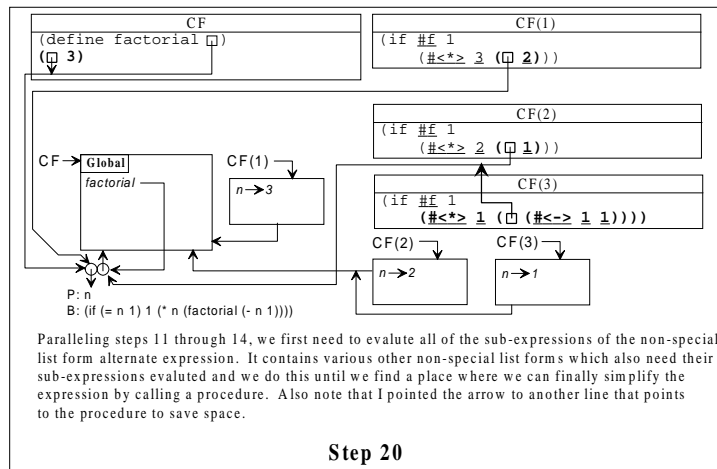
Step 15



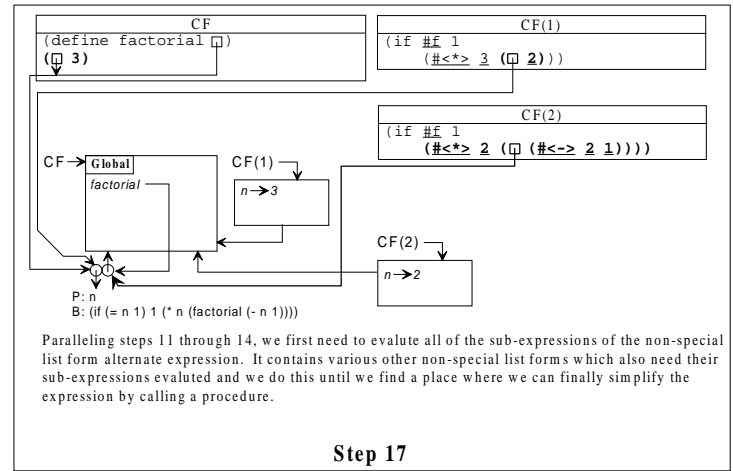
Step 16



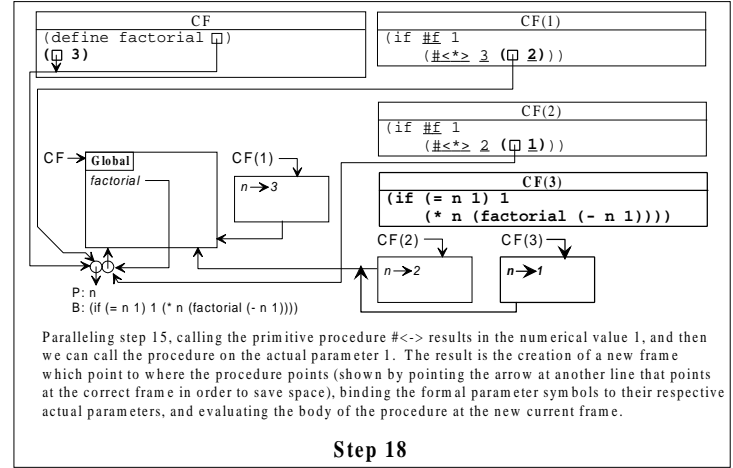
Step 19



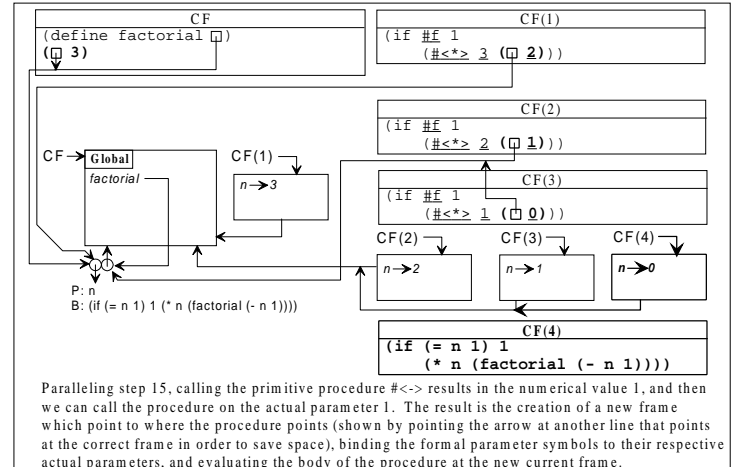
Step 20



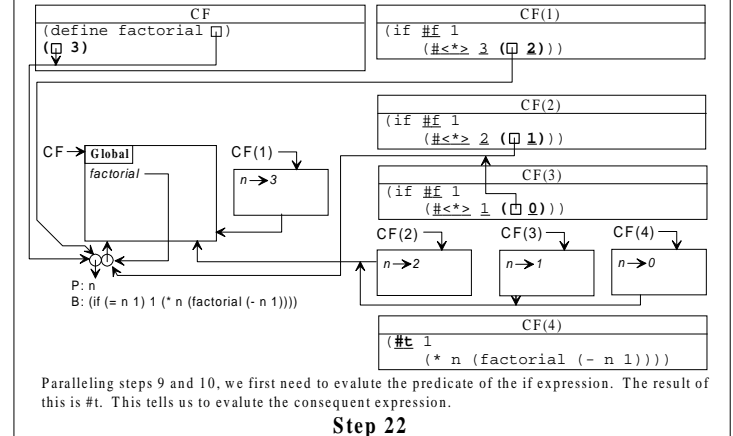
Step 17



Step 18



Step 21



Step 22

