Part 1

1. In applicative order strategy, we want to compute each expression's value before we are preforming the procedure. When we compute that, we get a value from the interpreter. We want to replace each expression with its computed value, but we have a value type, which the parser doesn't know. So, we want to convert each value to its corresponding LitExp.
2. valueToLitExp is not needed in normal order strategy, because we are using the value immediately after computing it, so after using this value, we don't need it anymore for future usage.
3. The main difference between special forms and primitive operators is that special forms are given by the user and has their own evaluation rules, while primitive operators are built-in functions provided by the language for performing basic operations and has known in advance values.
4. The main reason for switching is that the substitution model is very heavy and not efficient because in this model every time we want to compute a procedure's value, we need to calculate the values of all the parameters and replacing the names of each of them.
5. The difference is that in a special form we can determine the sequence of evaluation for the expression, as opposed to primitive types, where the sequence of evaluation is known and fixed.
6. Substitution model is "heavy" – we need to substitute each expression with it's value, so we can evaluate the same expression multiple times. In the environmental model, each time we encounter in an expression we look up for it's value in the current and previous environments, so we need to evaluate it's value only once. Example:

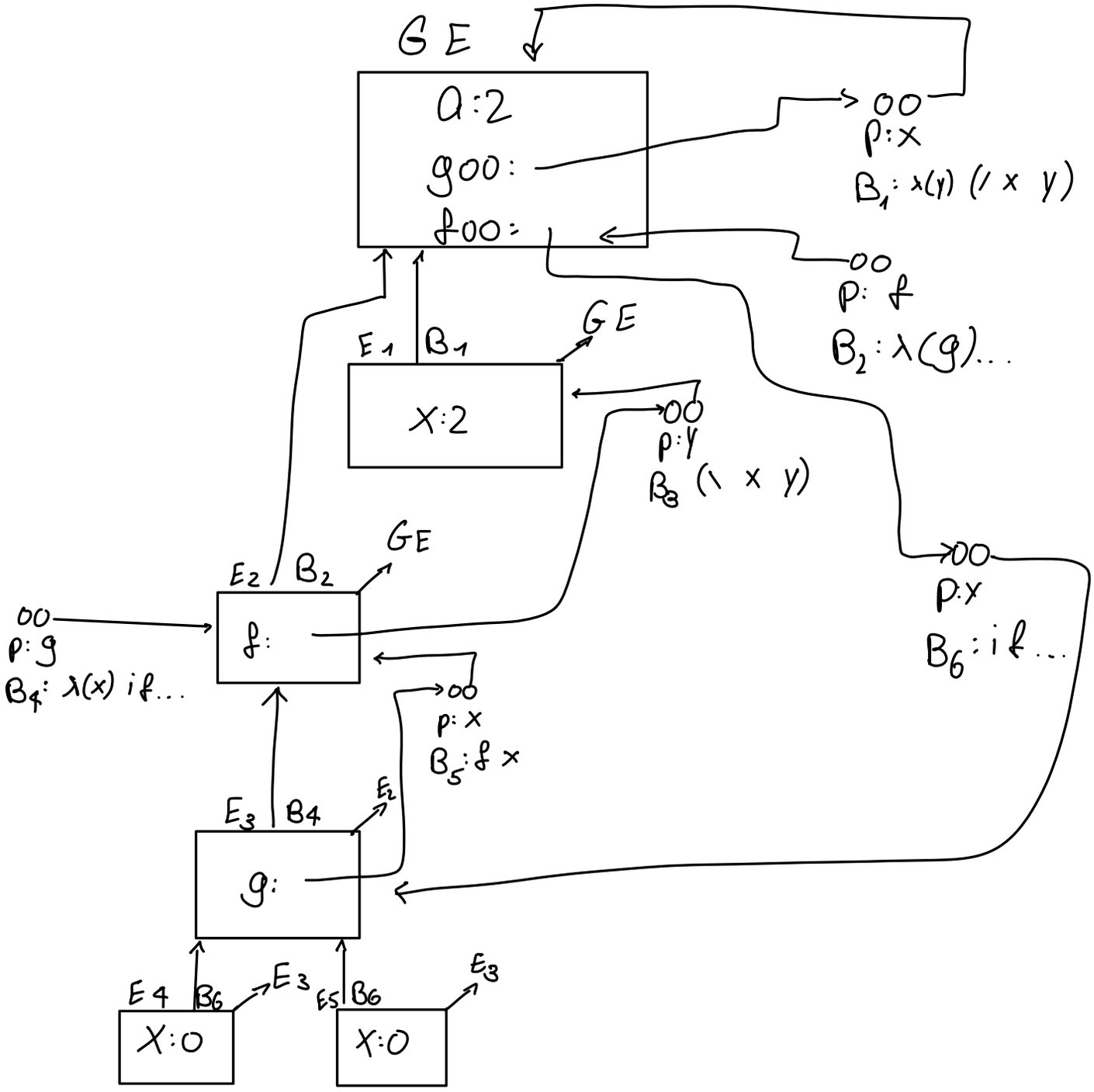
```
(define (square x)
  (* x x))


(+ (square 1) (square 1))
```

In this example, we notice that in substitute model we evaluate '(square 1))' twice, for substitute this expression with it's value. In the environments model we will evaluate '(square 1))' once and then in the second time we will have it's value in the current environment, so we will not have to evaluate it's value again.

7. We need to use box in the environment model, to use recursive correctly. With Boxing, a recursive function can call itself, because it can find itself in the global environment. On the other hand, we don't want to enable the user to change values of variables without any control, so we use box only where we enable to add binding to the global environment for recursive functions, and not for other expressions and not changing existing bindings.

8.



Part 2

2.1.3

'Bound?' expression must be special form and cannot be a primitive type, because when we evaluate a primitive type, we first evaluate its arguments. When we will activate 'bound?' with some argument, we don't know if the current or previous environments, contains this argument's value, so we will get an error. In a special form we can determine the sequence of evaluation for the expression, to prevent undefined behaviors that might crash the program.

2.2.2

'Time' must be special form because we want to evaluate the time it takes to evaluate the value of some expression. If we have a tool for get the current time for the program, we can implement 'time' as a user-function as such:

```
(define (time x)
  (let ((start-time (current-time)))
    (let ((result x))
      (let ((end-time (current-time)))
        (cons result (- end-time start-time))))))
```

'Time' cannot be a primitive type because the evaluation of the arguments will happen first, although we want to evaluate the time during (at the start and in the end) the evaluation of the argument.