1. Let expression is a special form in L3. A special form is a function with a different behavior than the normal expressions. In special form expressions we do not evaluate every argument, sometimes we bind variables instead or entirely skip the evaluation of an argument (like in if expression). In let expression we do not evaluate the first argument but instead we bind the local variables to our environment.

2. Depends on the way we define the let expression. If we define let by using lamba's expression as syntactic abbreviation, we would create closures in the process. On the other hand, if we define let as defined in our assignment then we don't create any closures. In the assignment we add the bindings to our environment as 'local variables' without creating any closures.

3.
Type 1: dividing a number by zero
Example: (/ 5 0)

Type 2:  adding a number to a boolean expression
Example: (+ 5 #t)

Type 3: using a variable we didn't define in the global environment or local environment.
Example: (car a) -> returns Var not found: a

Type 4: using the and prim op with arguments that are not boolean
Example: (and 5 2)

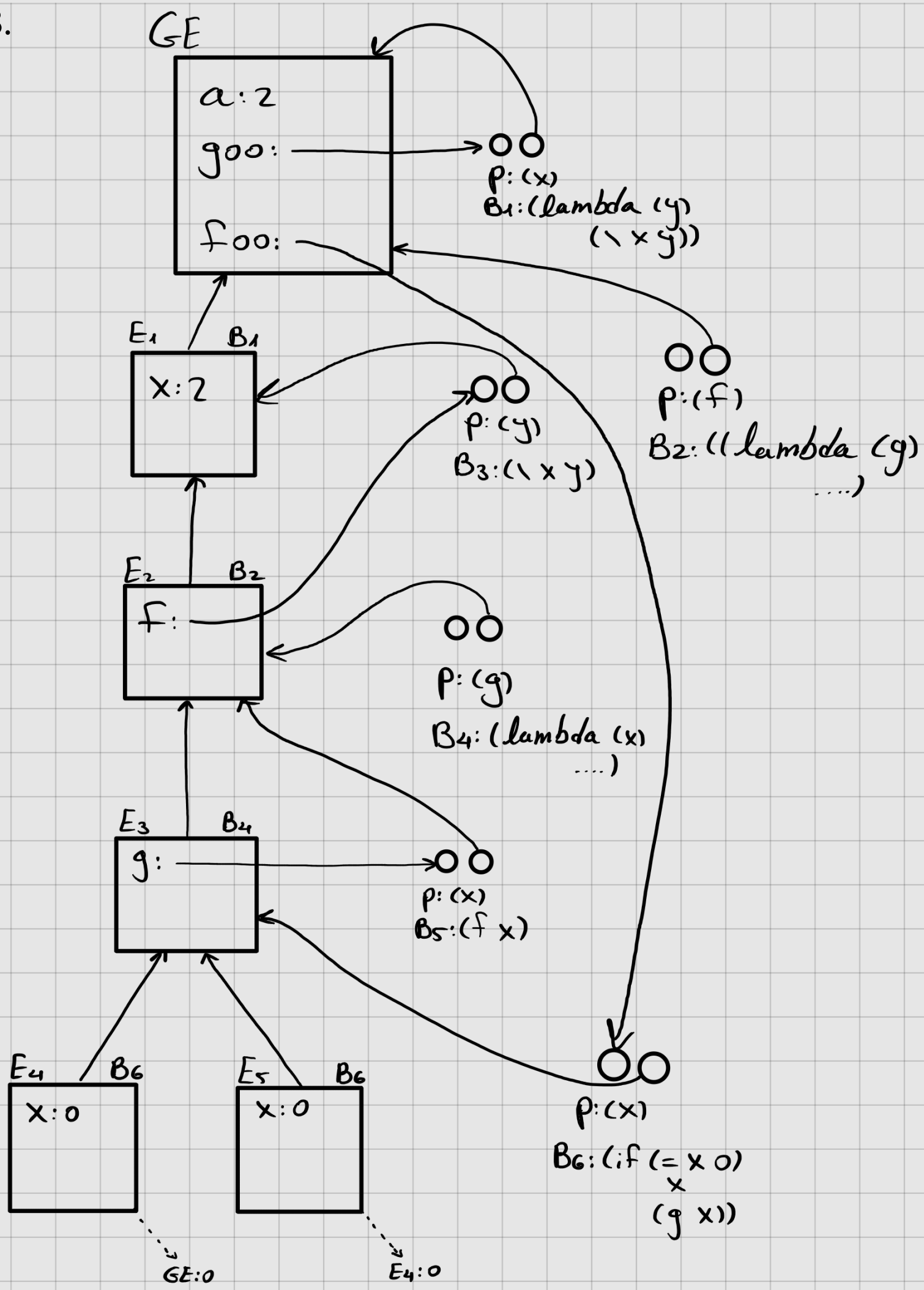
4. The purpose of the function valueToLitExp is to revert Value types into Expressions. It's essential because of the substitution model where app expressions are evaluated by changing the body expressions of the closure into the computed values. We revert it from values to expressions in order to fit the functions type.

5. In the normal evaluation strategy we don't compute the values before the substitution – meaning there is no reason to revert back the types in order to fit the functions.

6. Special form is needed when an app expression requires special treatment, meaning it doesn't necessary compute every argument. Primitive operator on the other hand, we calculate every argument, left to right no matter what.

7. The main reason for switching from the substitution model into the environment model is to avoid rewriting a function every time we call it. In the substitution model we would need to evaluate the arguments, rename and revert the types to expression every time we call an app expression. Switching into the environment model eliminates this issue.
Example:
```
(define y 3)
(
(lambda (x) (+ x y)
(+ 5 2))))
```

8.


GE
a: 2
goo:
foo:
p: (x)
B1: (lambda (y)
(\ x y))
E1
B1
x: 2
p: (y)
B3: (\ x y)
p: (f)
B2: ((lambda (g)
....)
E2
B2
f:
p: (g)
B4: (lambda (x)
....)
E3
B4
g:
p: (x)
B5: (f x)
E4
B6
x: 0
E5
B6
x: 0
p: (x)
B6: (if (= x 0)
x
(g x))
GE: 0
E4: 0