PPL – Assignment3 – Part1

By Yair Derry & Gal Noy

1. By converting the computed values into literal expressions, the valueToLitExp function ensures that the types match the expected types in the closure body. This allows the computed values to be properly substituted into the closure, maintaining type consistency and enabling correct evaluation of the closure.

2. valueToLitExp function, which converts computed values into literal expressions, may not be necessary in a normal order evaluation strategy interpreter. Since the arguments are not immediately evaluated, there is no need to convert them into literal expressions before substitution. The unevaluated expressions can be used directly in the function body, and their evaluation is deferred until it is needed.

3. There are two ways to approach this. The first method involves converting it into a procedure application and then evaluating it. Alternatively, the expression can be evaluated directly by first evaluating the bindings, extending the environment with their values, and then evaluating the body with the extended environment. In the first method, when evaluating the procedure application, a closure is created. This occurs because the procedure is first evaluated, transforming it into a closure. In the second method, no closure is created since all necessary actions are performed directly without intermediate steps.

4.

    - Type Error:

      (+ 5 #t)

    - Unbound Variable Error:

      (define x 10)

      (+ x y)

    - Evaluation Error:

      (/ 10 0)

    - Syntax Error:

      (+ 5 10))

5. A special form is a language construct with special syntactic and semantic rules. Examples include if, let, define, and lambda. Special forms have unique evaluation behaviors and introduce new scopes and bindings. A primitive operator is a predefined operation or function in the core language.

Examples include arithmetic operators like +, -, *, and /, as well as logical operators like and, or, and not. Primitive operators have predefined behavior and are evaluated as regular function calls.

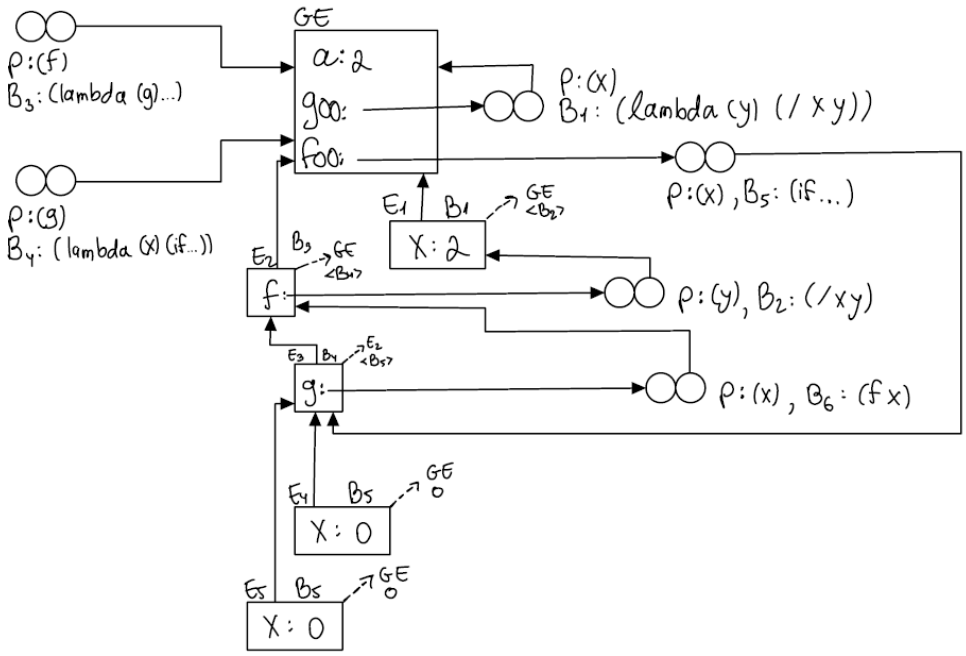
6. The main reason for switching from the substitution model to the environment model is to improve efficiency and avoid redundant computation. The environment model achieves this by using a data structure to store variable bindings instead of repeatedly substituting values into expressions.

   Example:

   In the code snippet (let* ((x 5) (y (+ x 2))) (+ x y)), the substitution model would replace x and y with their values multiple times, resulting in redundant computation. In contrast, the environment model stores the bindings of x and y and directly fetches their values when needed, avoiding redundant computation, and improving efficiency.

7. The main reason for implementing an environment using a box is to support mutable state and enable the update of variable bindings within the environment. The environment model, as initially defined, is based on immutable bindings, meaning once a variable is bound to a value, it cannot be changed. By introducing the concept of a box, which is a mutable container for a value, the environment can be extended to support mutable state. Each variable binding in the environment is associated with a box that holds the current value of the variable. The value inside the box can be updated or modified without changing the underlying environment structure.

8.



Part2

2.1.3. The function "bound?" cannot be implemented as a user function because it needs access to the enclosed environment, which is not feasible with this approach. Additionally, bound? cannot be a primitive operator since primitives are built-in functions implemented at a lower level that lack direct access to the evaluation context or environment.

2.2.3. The implementation of a time measurement functionality, such as the "time" function, can be achieved as a special form in a chosen interpreter, like TypeScript. By including relevant libraries like "date" and customizing the behavior of the special form, the interpreter's creator can enable users to measure the evaluation time of expressions. However, it cannot be implemented as a user function or a primitive since user functions lack direct access to evaluation time, and primitives evaluate the operands before performing any calculations, making it impossible to accurately measure the evaluation time.