**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. Applicative Order Evaluation – In applicative order evaluation, the expressions for the bindings in the let expression are evaluated first, in the order they appear. The values resulting from the evaluation of the expressions are then bound to the corresponding variables in the let expression. Finally, the body of the let expression is evaluated with these bindings in place.

Normal Order Evaluation – In normal order evaluation, the expressions for the bindings in the let expression are not evaluated immediately. The unevaluated expressions, along with the bindings, are passed as is to the body of the let expression. The body of the let expression is evaluated with the bindings in place. Evaluation of the expressions for the bindings is deferred until their values are needed in the execution of the body.

* Type Error:

(+ 5 #t)

* Unbound Variable Error:

(define x 10)

(+ x y)

* Evaluation Error:

(/ 10 0)

* Syntax Error:

(+ 5 10))

1. 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.
2. 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.

1. 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.