1 Part 1: Theoretical Questions [36 pts]

Submit the solution to this part as Part1.pdf. We can't stress this enough: the file has to be a PDF file.

1. What is the purpose of valueToLitExp and what problem does it solve? [4 pts]

Answer: the purpose of valueToLitExp is to convert values back into their corresponding expression form. In the applicative order evaluation, arguments need to be evaluated before they can be substituted into expression. This evaluation turns expressions into values. However, when we want to substitute these values back into the original expressions, we need to convert them back into their expression from using valueToLitExp. this conversion ensures that we can maintain consistency in the evaluation process and properly replace variables with their corresponding expression and finish the evaluating process.

2. valueToLitExp is not needed in the normal order evaluation strategy interpreter (L3-normal.ts). Why? [4 pts]

Answer: In the normal order evaluation strategy interpreter, we don't use valueToLitExp because the normal order evaluating delays the evaluation of the arguments until they are needed. This means that the arguments are passed unevaluated expressions (by name and not by value) to the function, and only when we really want them, they are evaluated to values. Thus, there is no need to convert them back to their let-expressions form.

3. What are the two strategies for evaluating a let expression? [4 pts]

Answer:

1)applicative order evaluation this evaluation first evaluates the arguments and then substitute them in the function body for example:

(let (x 2) (y (+ 2 5)) (+ x y))

-> in this case the applicative order well evaluate (+ 2 5)=7 and thin substituting the x,y values in the body.

2)normal order evaluation first we don’t evaluates the arguments before evaluating the body so we keep them the they are and when we actually need the we evaluate them. For example:

(let (x 2) (y (+ 2 5)) (+ x y))

-> in this case the normal order well first substitute the variables as the are in the body ((+ 2 (+ 2 5))and then evaluating the body to get to the result.

4. List four types of semantic errors that can be raised when executing an L3 program - with an example for each type. [4 pts]

a) Type error: by multiply a string with a number

(* 5 “5”):error

b) number of arguments In function:

((lambda (a b c d) (+ a (* b c d))) 1 2 3):error

c)evaluating an undefined variable:

(* y 5): y in undefined

d)Syntax error:

(define (add a b (/ a b)

5. What is the difference between a special form and a primitive operator? [4 pts]

Answer: Special form are built-in language constructs with unique evaluation rules that differ from regular function application. They control program flow and syntax, and examples include if, let, define, quote, and lambda. They cannot be redefined or extended by users and play a crucial role in the language's interpreter or compiler. In other hand Primitive operators are built-in basic operations in a programming language that perform fundamental computations on primitive data types, while special forms are language constructs with unique evaluation rules and syntax that control program flow.

6. What is the main reason for switching from the substitution model to the environment model? Give an example. [4 pts]

Answer: The main reason for switching from the substitution model to the environment model is to improve efficiency and avoid the costly process time of traversing and substituting variables in each recursive call. In the substitution model, when variables are substituted with their values throughout the program, it requires traversing the entire (AST)to substitute there values. This process can be computationally expensive, especially in the case of recursive functions with multiple occurrences of the same variable. For example: (define factorial (lambda (n) (if (= n 0) 1 (* n (factorial (- n 1))))).

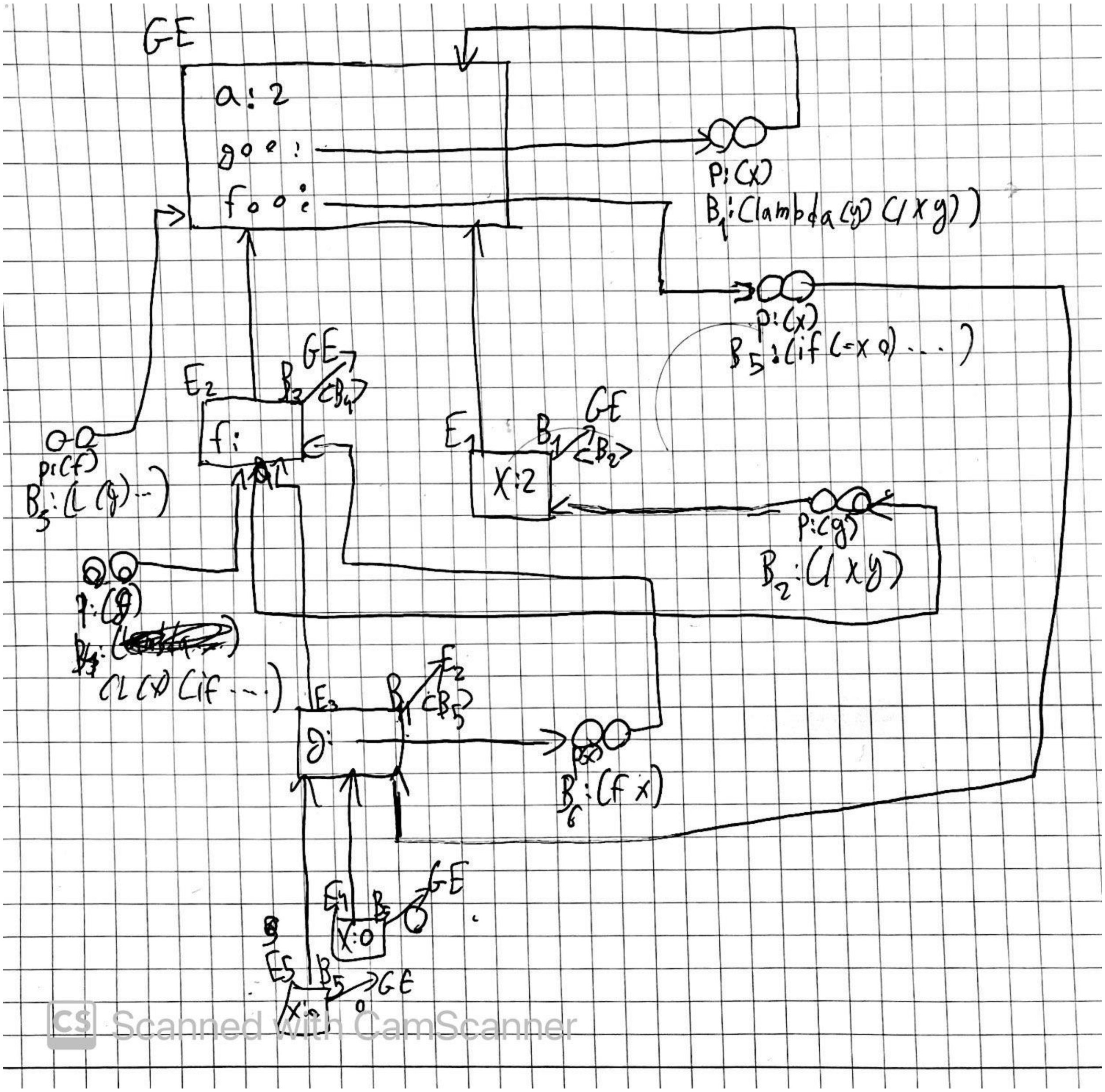
When evaluating (factorial 5) using the substitution model, each recursive call requires substituting n with a new value, resulting in traversing the entire (AST) and substituting in multiple places within the expression (if (= n 0) 1 (* n (factorial (- n 1)))).

The environment model, on the other hand, addresses this issue by maintaining a separate environment or symbol table that stores variable bindings. Instead of repeatedly traversing and substituting variables, the environment model performs efficient variable lookup in the environment. This avoids the need for costly substitutions in each recursive call, resulting in improved efficiency.

7. What is the main reason for implementing an environment using box? [4 pts]

Answer: Using box in environment model is to enable mutable state within the environment. This means that the we can updated and change data dynamically during the program execution.so this has a variant advantages to our program for example: Dynamic Updates, Flexibility, Efficient Memory Management, Mutable State Handling and Shared Access to data in some boxes.

8. Draw an environment diagram for the following computation. Make sure to include the lexical block markers, the control links and the returned values. [8 pts]



2.1.3 Why is bound? expression has to be a special form, and cannot be a primitive or a user function?

The bound? expression requires access to the current environment, which primitive and user-defined functions do not have. Special forms have preferential access to the environment and might provide unique evaluation rules. It is possible to check whether a variable is bound by making bound? a special form and then looking at the environment. This enables the expression to precisely verify variable bindings in the current environment.

2.2.2 Can it be implemented as a user function, primitive or special form?

In contrast to a user function or primitive, the timing tool, denoted by (time cexp>), is often implemented as a special form. User functions and primitives are inappropriate for precise timing since they have no control over the evaluation process and could be impacted by runtime overhead. To accurately monitor execution time, special forms offer the required control and access to start and stop timers at particular times. The language interpreter or compiler can precisely measure and provide time information for code optimization by implementing it as a specific form.