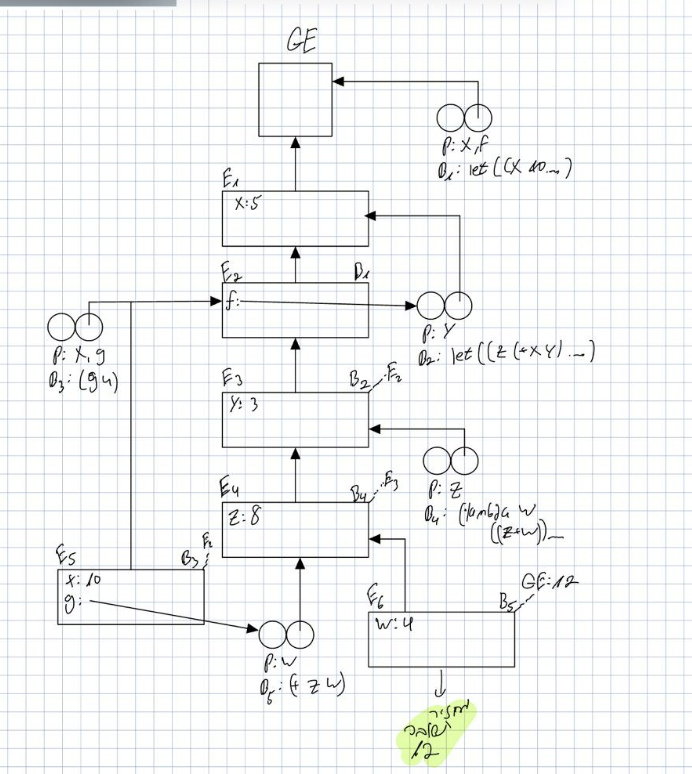
**Assignment 2**

1. Language variations:
   1. No, every program in L1 can be transformed into an equivalent program in L11. The define form is simply syntactic sugar for naming values or functions. Therefore, we can replace each use of a defined name with its actual value or expression, and the behavior of the program will remain the same.
   2. No, every L2 program can be transformed into an equivalent L21 program, because define is merely syntactic sugar for naming values and functions, which can be replaced by lambda expressions and immediate application. Even recursion, which typically uses define, can be expressed in L21 using fixed-point combinators like the **y-combinator**. Thus, no program in L2 fundamentally requires define for its semantics, only for convenience.
   3. No, there are no are programs in L22 that cannot be transformed into equivalent programs in L2. For programs with lambda containing more then one parameter we can create a nested lambda for example:  
      (lambda (a b)…) will transferred into (lambda(a) (lambda(b) …).  
      Because functions in L2 are functional(no side affects, printing, etc …) so the last expression in the lambda body will be the one that we will need to return.
   4. Yes, there are programs in L2 that cannot be transformed into equivalent programs in L23. As we studied in the lectures, L2 is Turing complete, so it allows the implementation of higher-order functions like map, which can take a lambda and a list, and apply the lambda to each element (as we did in L3). However, in L23, where functions cannot be passed as arguments, it is not possible to implement map in a general way. Therefore, programs that rely on passing functions as arguments such as map cannot be expressed in L23.
2. Let extend L3 syntax to support dictionaries in multiple ways:
   1. As a primitive operators:
      1. We will add to <prim-op> the following:
         1. dict | dict? | get
   2. As a special form:
      1. We will add new form <dict-entry>:
         1. <dict-entry> ::= (symbol <cexp>) / DictEntry(key:symbol, val: CExp)
      2. We will add to <cexp> the following:
         1. (dict (<dict-entry>)\*) / DictExp(entries:DictEntry[])
         2. (get <cexp> <cexp>) / DictRefExp(dict: CExp, key: CExp)
   3. As a L3 user procedures – in the code.
   4. Theoretical answers:
      * + 1. 2.1,2.3 – No need to for modification of the implementation because dict is implemented as a primitive operator and we are assuming that the evaluation of primitive operators is only when we call them.  
             2.2 – In 2.2, modification of the implementation is required because when evaluating the dict special form, the implementation currently evaluates each DictEntry value immediately during the construction of the dictionary. In normal order evaluation, this should be delayed, and the values should remain unevaluated until they are explicitly needed (for example, when accessed via get).
          2. Switching the evaluator from the substitution model to the environment model does not require any changes to the three dictionary implementations (2.1, 2.2, 2.3). The environment model only alters where and how variable bindings are stored and found inside the interpreter; it leaves the values themselves untouched. A variable that contained a dictionary in the substitution model still refers to exactly the same dictionary under the environment model—the interpreter now finds that value by walking the frame chain instead of performing textual substitution. Because every version of the dictionary is purely a data-manipulation abstraction, all three versions continue to work as-is in the environment model. No modifications are needed.
          3. In 2.1 (primitive operator) and 2.3 (user procedure):  
             If we call (dict (a 1) (b 2)) so the interpreter would first try to evaluate (a 1) and (b 2) as proc—exp which will return failure.
          4. Are there expressions which can be defined as a field's value in dict special form (2.2) but not in dict primitive operator (2.1) or user procedure (2.3)?   
             **ANSWER**: yes, for example for the following expression in 2.2 -(dict (a (+ 1 2))) we are forced to write the post-evaluate value in 2.1 or 2.3. That because we are not evaluated the arguments before being passed in.  
             ----  
             Are there expressions in L32 which cannot be transformed to equivalent expressions in L3 (according to 2.5 method below)? **ANSWER**: Fields whose value is included in the syntactic structure will be converted to the same value, such as numbers and strings. Fields that require calculation by the interpreter will be converted to a literal expression, losing the original semantics.
          5. The 2.2 Special Form implementation of dictionaries offers several advantages. It provides a clean and natural syntax for users, allowing dictionaries to be written intuitively as  
              (dict (a 1) (b 2)) without the need for quoted lists or awkward pair notation. Since it's a special form, it can control the evaluation of its fields explicitly, enabling support for arbitrary expressions as field values. This makes it both expressive and safe under both applicative and normal order evaluation strategies. However, this approach also comes with some disadvantages. It requires extending the parser to recognize and handle the new dict special form, which adds complexity to the language's syntax analysis phase. Moreover, because it’s not a regular function or primitive operator, it’s less composable and cannot be passed around like a first-class procedure. Despite these limitations, its readability and flexibility make it the most practical and user-friendly choice for extending the language with dictionary support.
3. a. 

b. A close-up of a graph paper

AI-generated content may be incorrect.