Assignment 3

Answer 1:

```
len [] = 0
len (x:xs) = 1 + len xs

sublist [] = [[]]
sublist (x:xs)
          |len(xs)==0   = [[x]]
          |otherwise    = [[x]]++(calculateall x (sublist xs))++sublist xs
calculateall x (y:ys)
               |len(ys)==0   = [[x]++y]
               |otherwise      = [[x]++y]++(calculateall x ys)
```

Answer 2:

```
len [] = 0
len (x:xs) = 1 + len xs

replic [] = [[]]
replic (xs) = function (len(xs)) xs
function ln (y:ys)
           |len(ys)==0    = [makecopy y ln]
           |otherwise = [makecopy y (ln-len(ys))]++function ln ys
makecopy a n
         |n==0    = []
         |otherwise = [a]++makecopy a (n-1)
```

Answer 3:

```
len [] = 0
len (x:xs) = 1 + len xs

cproduct [] a = [[]]
cproduct b [] = [[]]
cproduct [] [] =[[]]
cproduct (x:xs) ys
            |len(xs)==0    = (prod x ys)
            |otherwise       = (prod x ys)++(cproduct xs ys)
prod k (c:cs)
         |len(cs)==0   =[[k]++[c]]
         |otherwise      =[[k]++[c]]++prod k cs
```

Answer 4:

```
map f [] = []
map f (x:xs) = f x : map f xs

sqsum ys = foldr1 (+) (map (\a->a*a) ys)
```

Answer 5(a):

We use a system of rules to derive the valid typing judgments. These rules constitute a specialized logic whose statements are typing judgments that relate expressions and types.

Typing judgments where the expression contains free variables have to be considered with respect to some context that tells us what the types of those free variables are. These typing contexts are also known as type environments, and we can think of them as finite functions mapping variables to types. So, a typing judgment require a type environment sometimes.

If we want to type an expression e that contains free variables as well as global constants, we must provide an appropriate context with type assignments for all the free variables that occur in e.

Contexts may be supplied as initial assumptions about variables and primitive operators, but they can also evolve as part of derivations of certain kinds of compound expressions involving variable bindings.

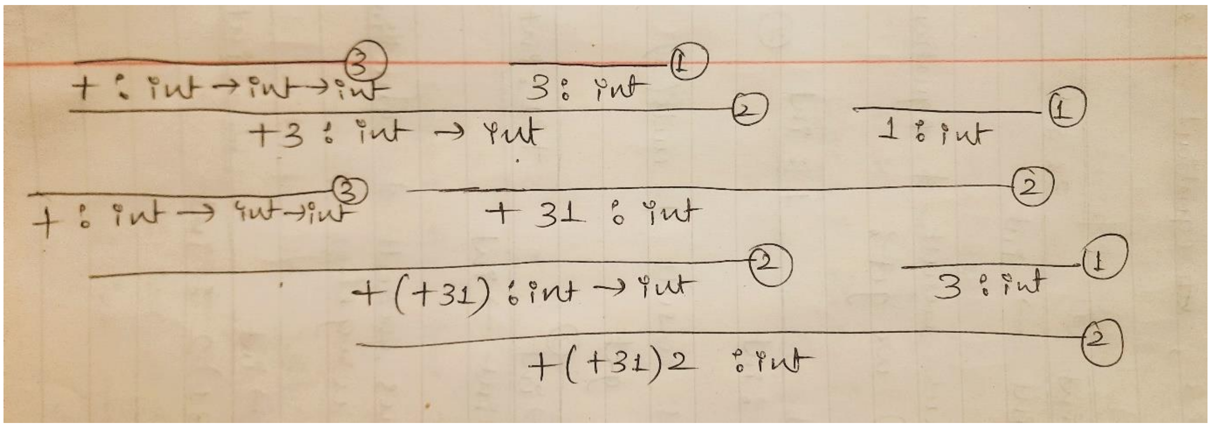
Answer 5(b):

Beyond predicting the form of the ultimate value of e, typing judgment also requires that e be well-typed, meaning that it is internally consistent, and consistent with its context or environment in the case where it contains free variables.

Program is well-typed means every expression e is well-typed, that its components fit together properly according to the rules that is operators are applied to the right kinds of arguments.
To be well typed, the type of the argument must agree with the domain of the function being applied.

Answer 5(c):

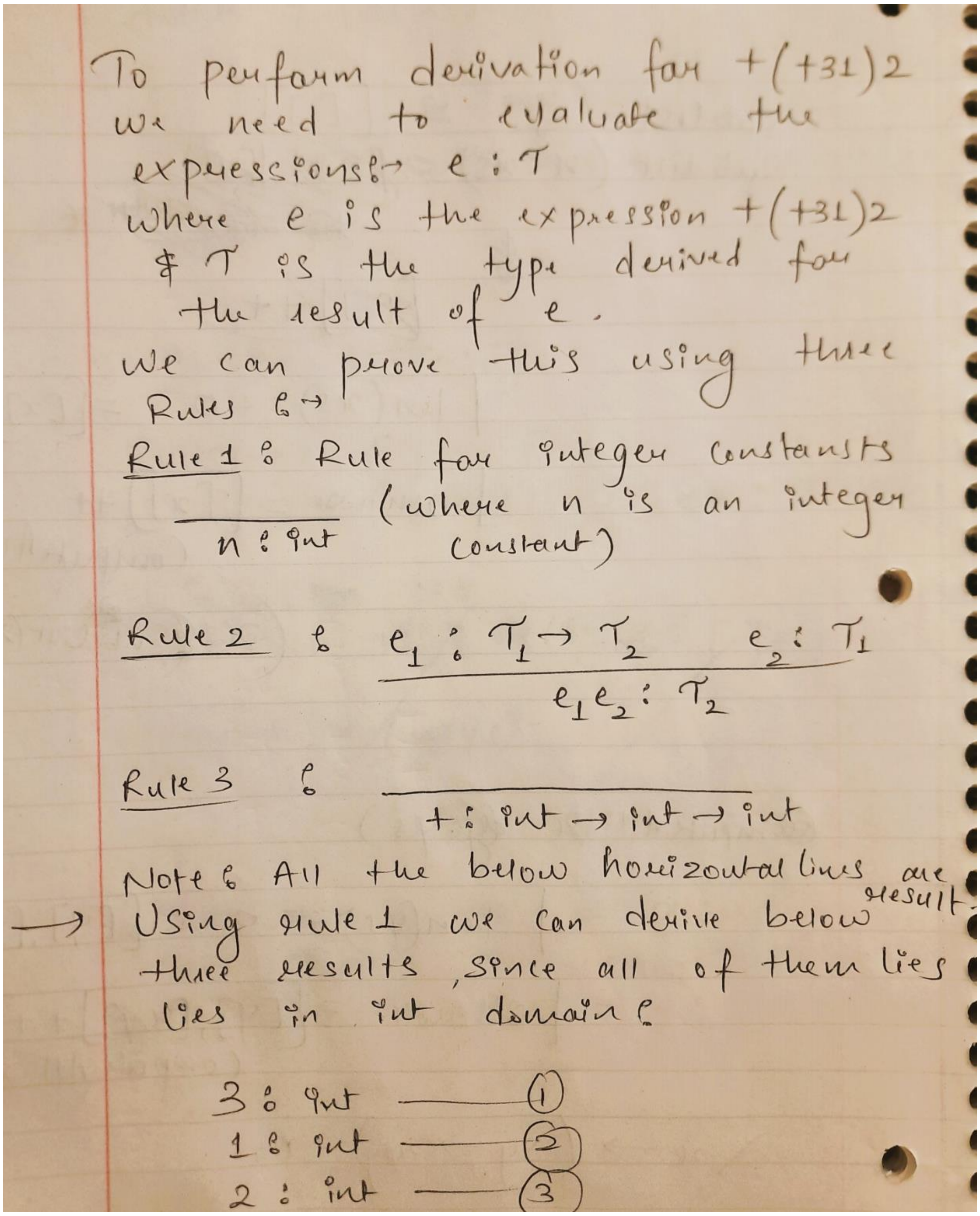
Final derivation:



$$\dfrac{\dfrac{}{+ : int \to int \to int}\,③ \qquad \dfrac{}{3 : int}\,①}{+3 : int \to int}\,② \qquad \dfrac{}{1 : int}\,①$$

$$\dfrac{\dfrac{\dfrac{}{+ : int \to int \to int}\,③ \qquad \dfrac{}{+31 : int}\,②}{+(+31) : int \to int}\,② \qquad \dfrac{}{3 : int}\,①}{+(+31)2 : int}\,②$$

Explaination:



To perform derivation for +(+31)2 we need to evaluate the expressions:→ $e : T$
where e is the expression +(+31)2
& T is the type derived for the result of e.
We can prove this using three Rules :→

Rule 1 : Rule for integer constants

$$\frac{}{n : int}$$ (where n is an integer constant)

Rule 2 :

$$\frac{e_1 : T_1 \rightarrow T_2 \qquad e_2 : T_1}{e_1 e_2 : T_2}$$

Rule 3 :

$$\frac{}{+ : int \rightarrow int \rightarrow int}$$

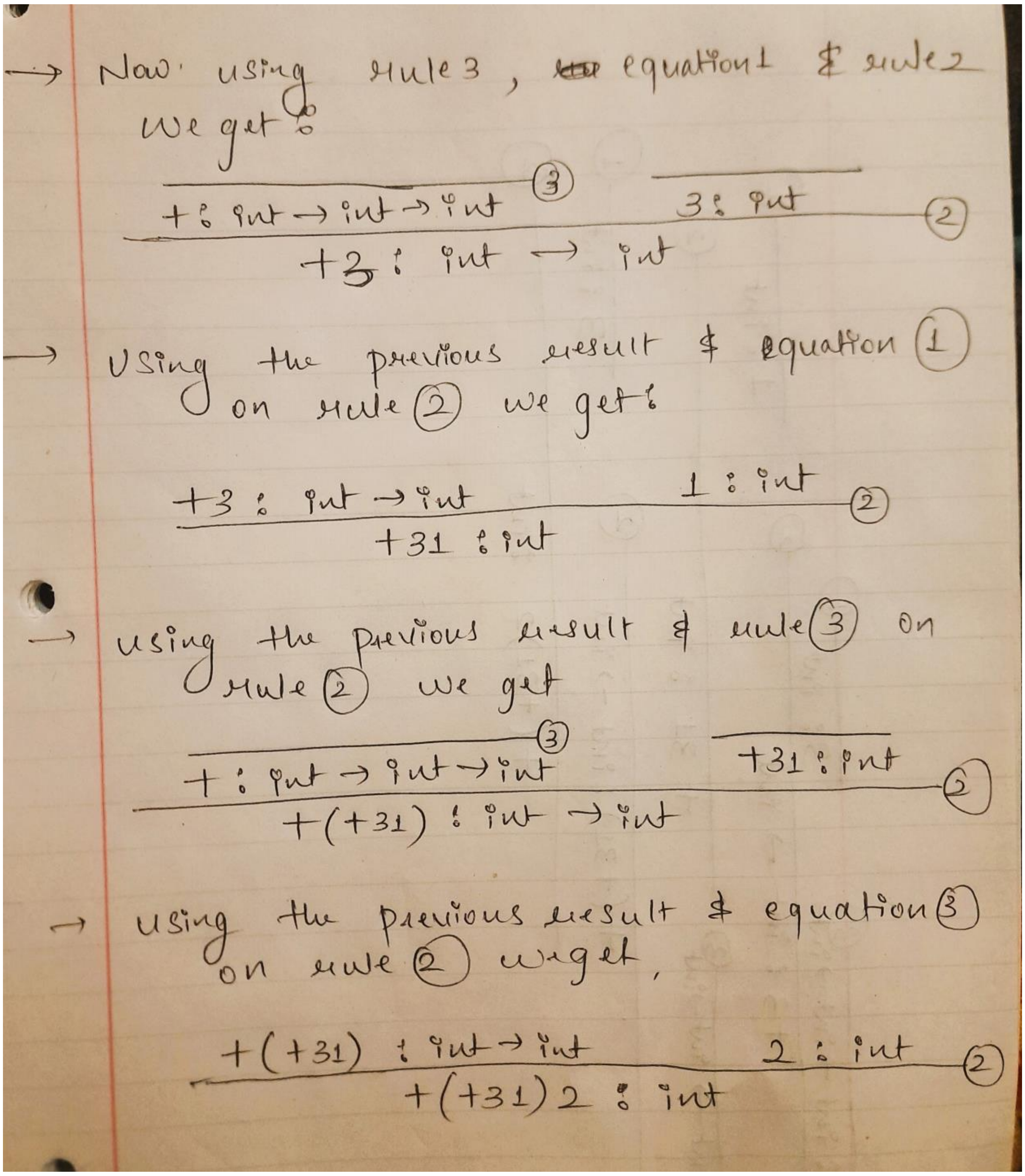
Note : All the below horizontal lines are result

→ Using rule 1 we can derive below three results, since all of them lies lies in int domain :

$3 : int$ ——— ①

$1 : int$ ——— ②

$2 : int$ ——— ③



→ Now using rule 3, ~~[illegible]~~ equation1 & rule2 we get:

$$\dfrac{\dfrac{}{+ : int \to int \to int}\;\boxed{3} \qquad \dfrac{}{3 : int}}{+3 : int \to int}\;\boxed{2}$$

→ Using the previous result & equation $\boxed{1}$ on rule $\boxed{2}$ we get:

$$\dfrac{+3 : int \to int \qquad 1 : int}{+31 : int}\;\boxed{2}$$

→ using the previous result & rule $\boxed{3}$ on rule $\boxed{2}$ we get

$$\dfrac{\dfrac{}{+ : int \to int \to int}\;\boxed{3} \qquad \dfrac{}{+31 : int}}{+(+31) : int \to int}\;\boxed{2}$$

→ using the previous result & equation $\boxed{3}$ on rule $\boxed{2}$ we get,

$$\dfrac{+(+31) : int \to int \qquad 2 : int}{+(+31)2 : int}\;\boxed{2}$$

Answer 5(d):

Decidability for a type system means there is an algorithm that that either computes the type of an expression e or reports an error indicating that the expression is ill-typed in considerable and definite time. This implies algorithm exist for deciding, given Γ and e, whether Γ ⊢ e : τ is derivable for some τ.

Termination of the algorithm is necessary, because a type system is generally not able to answer the question of whether an expression terminates. Typing judgments that cannot be derived by our rules are deemed invalid.

Answer 5(e):

Type checking. The process of type checking takes an expression e and a context Γ and computes a type τ such that $\Gamma \vdash e : \tau$, reporting a type error if such does not exist. So, a type error is an error thrown by modern compilers when the type of values applied to a function or operation is not of the expected type or simply the error returned if the type of the expression cannot be evaluated by rules of type checking.