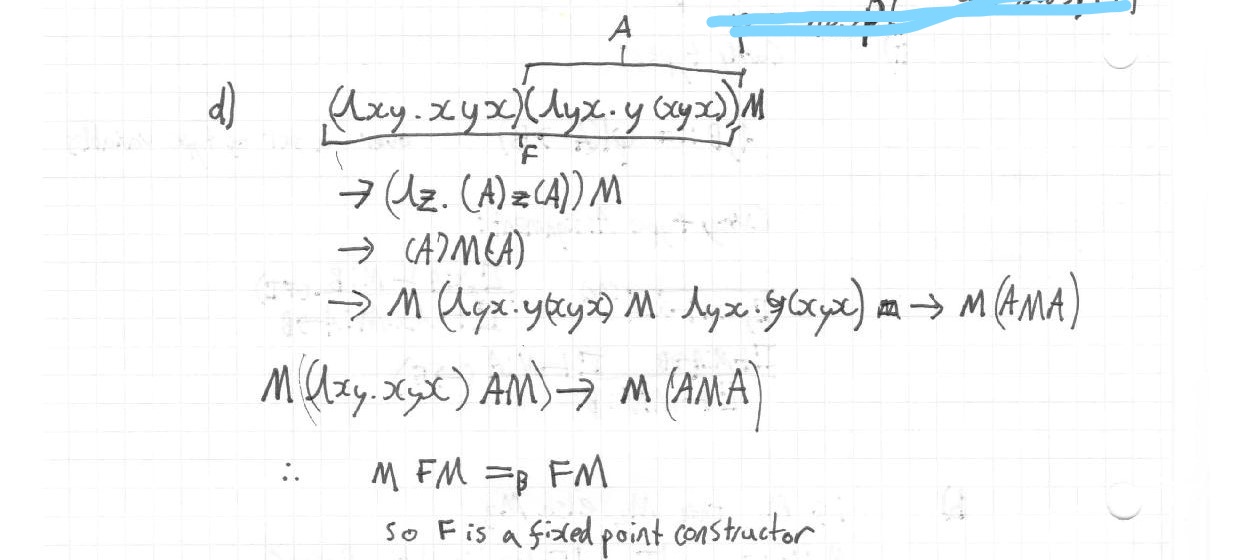
* 1. 1. Lambda Terms defined by Definition 1.1 in notes. Beta Reduction defined by inference rules in section 1.2 of notes (Part of definition 1.4).
     2. Curry Types are defined by Definition 2.1 i) in notes. Curry Type Assignment is defined by the derivation rules in definition 2.2 i) in notes.
  2. Exercise 2.20 i) and ii) in notes, answered as exercise 2.21 i) and ii) in the model answers for the notes.

|  |  |  |
| --- | --- | --- |
| ppc x = (x:φ, φ) |  |  |
|  | Where φ = newPhi |  |
| ppc λx.M |  |  |
|  | |π == π’,x:A = (π’, A→P) |  |
|  | |otherwise = (π, φ→P) |  |
|  |  | Where (π, P) = ppc M |
|  |  | φ = newPhi |
| Ppc MN | applySub S2 (applySub S1 (contextUnion π1 π2, φ)) |  |
|  |  | Where (π1, P1) = ppc M |
|  |  | (π2, P2) = ppc N |
|  |  | S1 = unify P1 P2→φ |
|  |  | S2 = unifyContexts (applySub S1 π1) (applySub S1 π2) |
|  |  | φ = newPhi |
| ppc true | <∅; Bool> |  |
| ppc false | <∅; Bool> |  |
| ppc if C then M else N | S3S2S1<Π1∪Π2∪Π3; P3> | Where <Π1; P1> = ppc C  <Π2; P2> = ppc M  <Π3; P3> = ppc N  S1 = Unify C Bool  S2 = Unify (S1M)(S1N)  S3 = UnifyContexts (S2S1Π1)  (S2S1Π2) (S2S1Π3) |
|  |  |  |

* 1. 

￼

This term is not typeable, as the subterm x y x requires x to be applied to itself (and y) which cannot be typed within this type system. (Also apologies for my handwriting in the above picture)

* 1. No, this term is not typeable. An attempt to normalise the subterm a b b (a b b) will leave you with (λx.xx)(λx.xx) whi ch is untypable. Therefore this term is 1) not strongly normalisable and

2) involves self application

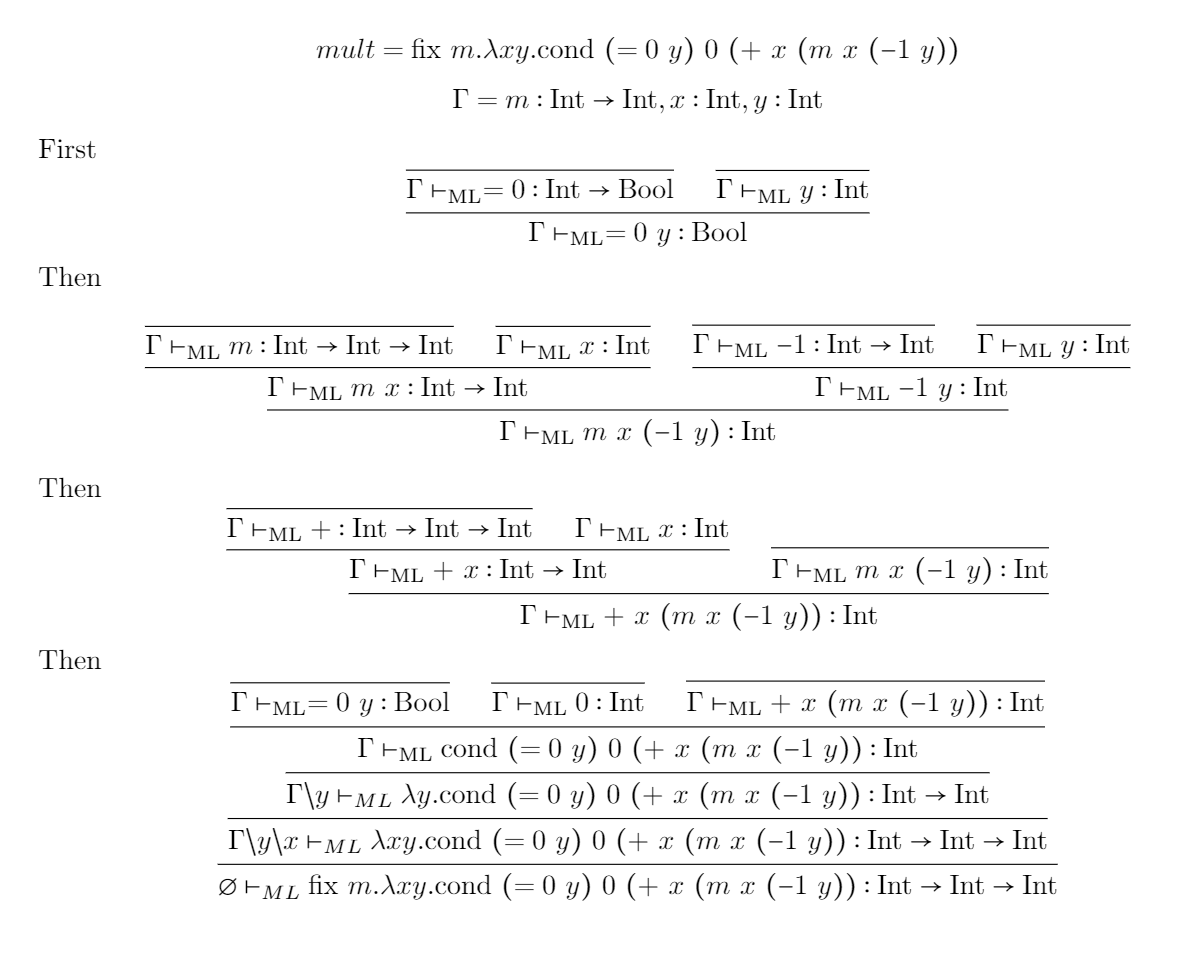
so cannot be typed in the curry system.

1. 1. Definition 5.1 i) and iii) in notes.
   2. Definition 5.2 i) in notes, Definition 5.4 in notes.
   3. 1. Set Γ = y: 1, x: 1→2. Then 2 (Ax), one (→E) followed by a (→I) followed by a (fix). Pretty Simple
      2. Not typable

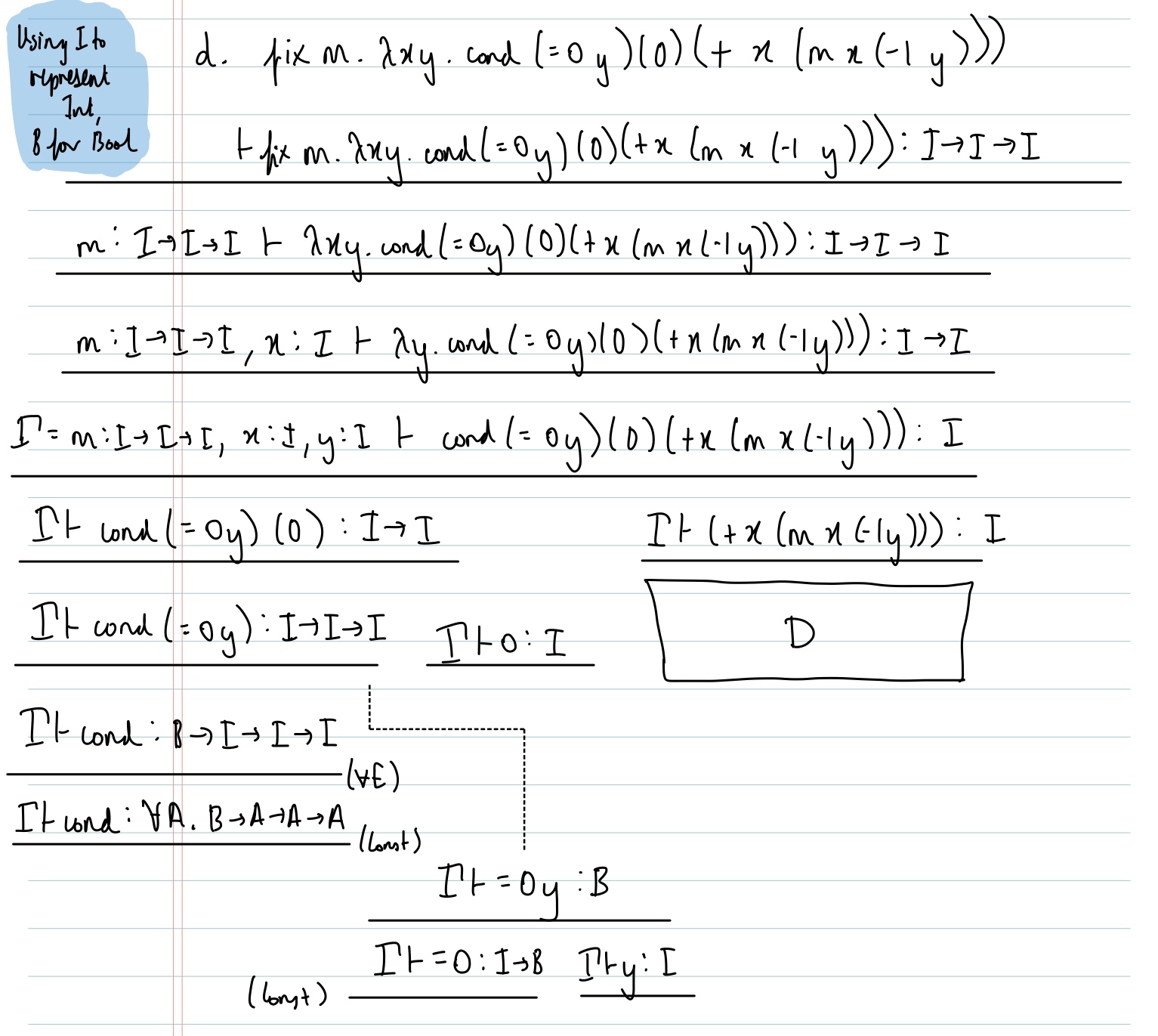
Attempt at an explanation:

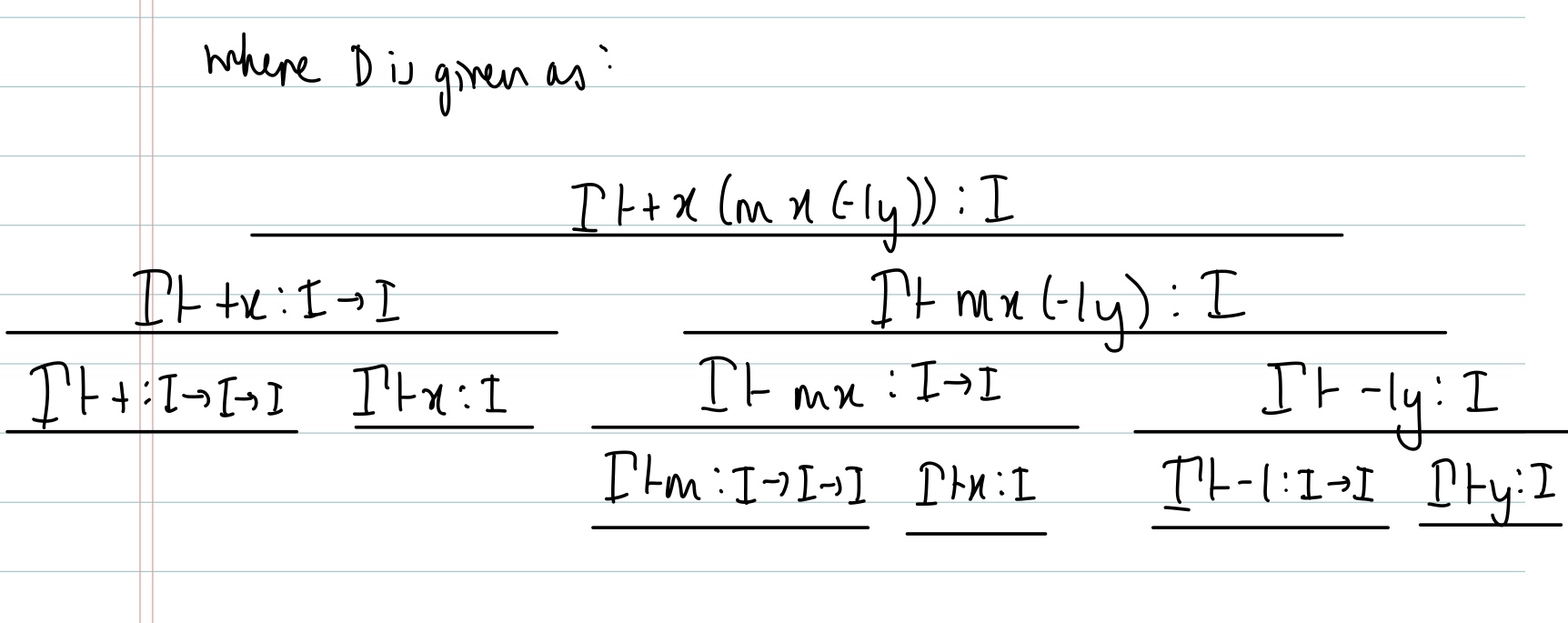
The principal pair of type of l yz. xz (yz) (which we can calculate using pp\_c) is <x: 1 -> 2 -> 3; (1 -> 2) -> 1 -> 3 > to be typable we would need the principal type and the type of x in the principal context to be the same

* 1. Fix m. λxy. cond (=0 y) 0 (+ x (m x (-1 y))) (Derivation long but kinda straightforward. Take Γ = m: Int→Int→Int, x:Int, y:Int to start)



I feel it should be something more like this, since the above solution makes use of a new rule whereas the question gives *cond* an explicit type:





* 1. Exercise 5.17 and 5.18 in model solutions.

1. 1. 1. Definition 9.1 i) in notes.
      2. Definition 9.3 in notes.
   2. 1. Exercise 9.12 i) answered as exercise 9.33 i) in model answers for notes.
      2. Exercise 9.12 ii) answered as exercise 9.33 ii) in model answers for notes.
      3. Exercise 9.12 iii) answered as exercise 9.33 iii) in model answers for notes.
      4. Exercise 9.12 iv) answered as exercise 9.33 iv) in model answers for notes.
      5. The last part of exercise 9.16 answered as exercise 9.37 in model answers for notes.
   3. First comparison is exercise 9.13 answered as 9.34 in model answers for notes.

Second comparison: Neither term is typable in curry system so set for both is empty / Terms are beta equal so equivalent sets of types

1. 1. Definitions 8.1, 8.2, 8.3
   2. The grammar definition in 8.4 is the definition of a recursive type. Equi-recursive type assignment rule: Definition 8.5. Iso-recursive type assignment rules + reduction rule + Expression extension: Definition 8.8. Difference: Iso-recursive approach requires explicitly putting fold and/or unfold within terms and being able to add these during a derivation, Equi-recursive allows you to simply give a term any type that is mu-equal to the type you derived for it.
   3. Exercise 8.19 in notes, answered as Exercise 8.19 in model answers for notes.
   4. 1. List constructor [A] is expressed in example 8.6 in notes?
      2. Figure 11 in the notes is a derivation of the type for the function that calculates the length of a list of arbitrary type.