1. 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.
      3. Curry Type Assignment is defined by the derivation rules in definition 2.2 i) in 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 |

* 1. 1. 1. (1→2→3) → (1 →2) →1 →3
        2. 1→2→1
        3. (1→2)→1→2
        4. 1→1
     2. For (λxyz.xz(yz))(λxy.x)(λx.x):

|  |  |  |
| --- | --- | --- |
|  | Unify 1→2→3 1→2→1 | = S2∘S1 |
| S1 = | Unify 2→3 2→1 | = S4∘S3 |
| S3 = | Unify 3 1 | = 3 ↦ 1 |
| S4 = | Unify S32 S35 | = 2 ↦5 |
| S2 = | Unify S11 S14 | = 1↦4 |
|  | Unify 1→2 6→6 | = S6∘S5 |
| S5 = | Unify 2 6 | = 2↦ 6 |
| S6 = | Unify S51 S56 | = 1↦6 |

And therefore (λxyz.xz(yz))(λxy.x)(λx.x): 1→1

For (λxy.xy)(λx.x):

|  |  |  |
| --- | --- | --- |
|  | Unify 1→2 1→1 | = S8∘S7 |
| S7 = | Unify 2 1 | = 2↦1 |
| S8 = | Unify S71 S71 | = 1↦1 |

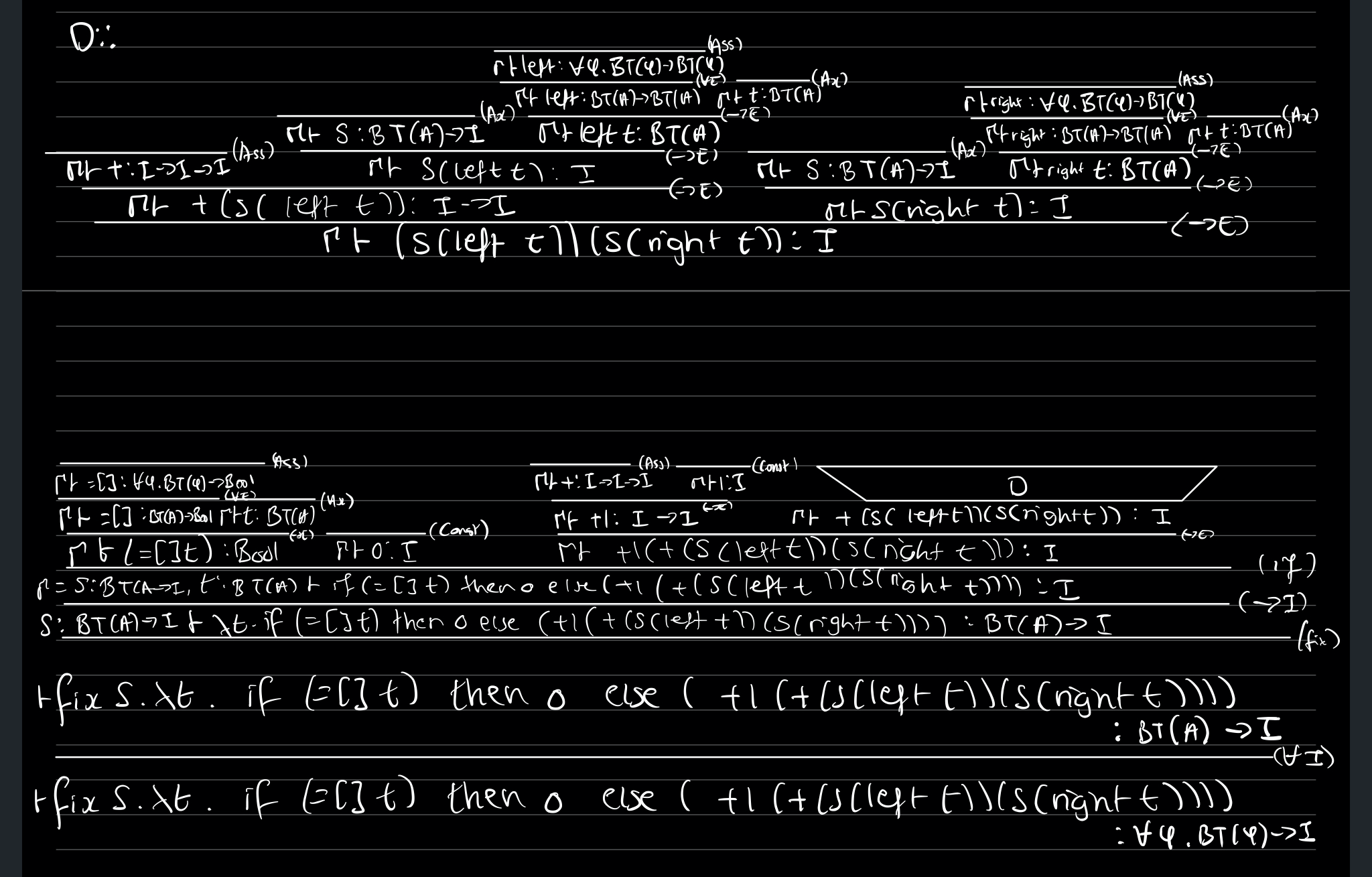
And therefore (λxy.xy)(λx.x): 1→1

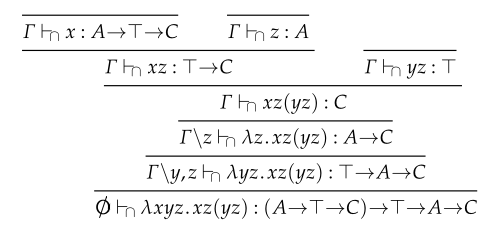
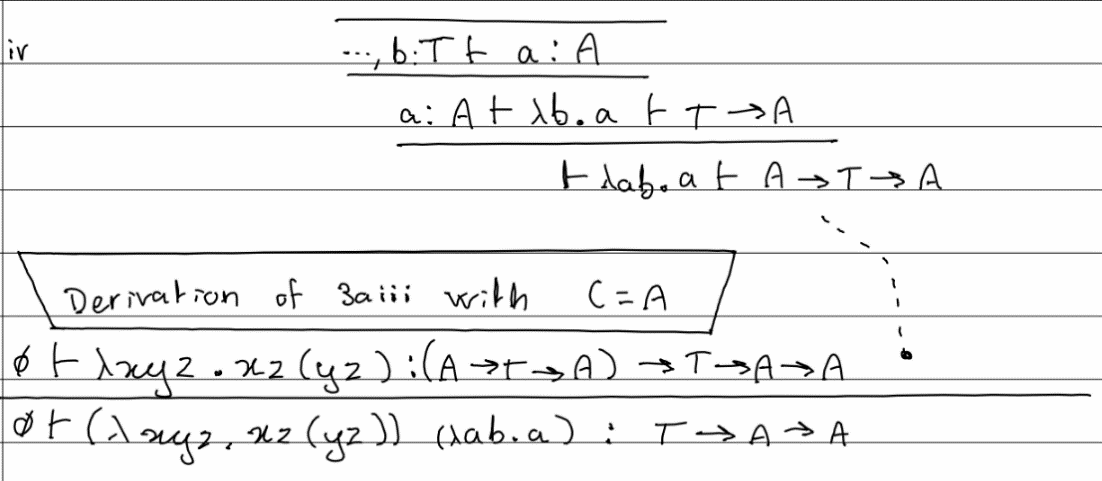
iii)

* + 1. We can conclude that the set of types assignable to both terms are equal and equal to the set of types assignable to the identity.

**d)** Take (λpqr.pq(qr)): (1→2→3)→(1→2)→1→3, (λxyz.xz(yz)): (4→5→6)→(4→5)→4→6, (λcd.cd): (7→8) → 7 → 8, (λe.e): 9→9. An attempt to type this will eventually result in an attempt to unify (4→5) to (4→5→6), which is not possible in Curry’s system. Therefore this term is not typable.

1. 1. Definition 6.1 i) and iii) in notes.
   2. Definition 6.2 i) in notes, Definition 6.4 in notes.
   3. If fix were a term rather than syntax, then for this rule to work it would have to be the case that fix’s type is ((A -> A) -> A), where g and E provide the first two As. this then means that fix, as a term, would fit into the F =w N[F/x] requirement of fixed point operators described on page 33.
   4. `Mycroft’s system allows for recursive types to be used in the fix rule (i.e. g and E can have the same recursive type.) “Thus, the only difference lies in the fact that, in this system, the derivation rule (fix) allow for type-schemes instead of types, so the various occurrences of the recursive variable can be typed with different Curry-types.” (Also Mycroft’s type assignment is undecidable).
   5. Fix s. λt. If =[]t then 0 else + 1 (+ (s left t) (s right t))



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. 
      4. 
   3. Exercise 9.13 in the notes, answered as Exercise 9.34 in model answers for notes.
   4. *(Worth noting here that the 2018-2019 question on intersection types seems at the very least more involved than this if not marginally more difficult)*
2. 1. Definitions 8.1, 8.2 and 8.3 in notes.
   2. Definition 8.4 in notes. Equi-recursive type assignment rule: Definition 8.5 in notes. Iso-recursive type assignment rules: contained within Definition 8.8 in notes. Iso-recursion uses syntactic markers to show explicitly where the =mu steps are, and equi-recursion does not.
      1. Exercise 8.19 in notes, answered as Exercise 8.19 in model answers for notes.