Tas Typeur Rapport

1. Architecture du code

> Le code dans le fichier est groupé en cinq sections et chaque section contient à son tour trois modules selon les modules du sujet comme:

I.Syntaxe	1. λ-calcul simplement typé 2. PCF 3. Traits Impératifs
II.Sémantique	
III.Types	1. λ-calcul simplement typé 2. PCF 3. Traits Impératifs
IV.Génération d'équations	1. λ-calcul simplement typé 2. PCF 3. Traits Impératifs
V.Unification	1. λ-calcul simplement typé 2. PCF 3. Traits Impératifs

---

2.Achèvement du projet

Pour la partie Syntaxe: tous fait

Pour la partie Sémantique: tous fait

Pour la partie Types: tous fait

Pour la partie Génération d'équations : rest <let>

Pour la partie Unification : tous fait

3.Résultats des tests

> ***Timeout en 5s***

- I = λx.x

    - reduction:OK inference:OK
- K = λx.λy.x
    - reduction:OK inference:OK
- KI
    - reduction:OK inference:OK
- KII
    - reduction:OK inference:OK
- S = λx.λy.λz.xzyz
    - reduction:OK inference:OK
- NAT1 = (λx.(x+1))*3
    - reduction:Timeout inference:OK
- NAT2 = λx.(x+x)
    - reduction:OK inference:OK
- NAT3 = NAT2 * I
    - reduction:OK inference:OK
- Omega = (λx.xx)(λy.yy)
    - reduction:OK inference:OK
- ex_iz = (ifzero (5 - 4) then 1 else 2)
    - reduction: timeout inference:OK
- ex_ie = (ifempty (tete [20]) then 5 else (tete []))
    - reduction: timeout inference:OK
- r = λx.x (ref 2)
    - inference:OK inference:OK
- e_let = let y = z in λx.(xy)
    - reduction: timeout inference: pas fait

---

4. Remarque

Utilisez le command pour compiler le fichier lecontypage.ml:

```
ocamlc -o projet lecontypage.ml
```

Et puis exécuter le programme par le command:

```
./projet
```

Étant donné que de nombreux exemples feraient en sorte que le test dure plus longtemps que le temps imparti, afin d'avoir plus d'exemples pour prouver les résultats, j'ai mis:

App (Abs (v1,v2), e2) -> reduit_t (substitue_variable v2 v1 e2) → App (Abs (v1,v2), e2) -> substitue_variable v2 v1 e2

Voici quelques-uns des sacrifices consentis pour le test.

Les résultats des tests:

```
---------------------------------- 2. λ-calcul simplement typé ----------------------------------------------------
Beta réduction pour I = λx.x :
(fun x -> x)
Inférence de id=λx.x:
(fun x -> x) ***TYPABLE*** avec le type (T2 -> T2)
Beta réduction pour K = λx.λy.x :
(fun x -> (fun y -> x))
Inférence de K = λx.λy.x :
(fun x -> (fun y -> x)) ***TYPABLE*** avec le type (T6 -> (T5 -> T6))
Beta réduction pour KI :
(fun V3 -> (fun V1 -> V1))
Inférence de KI:
(fun V3 -> (fun V1 -> V1)) ***TYPABLE*** avec le type (T7 -> (T10 -> T10))
Beta réduction pour KI :
(fun V3 -> (fun V1 -> V1))
Inférence de KI:
(fun V3 -> (fun V1 -> V1)) ***TYPABLE*** avec le type (T7 -> (T10 -> T10))
Beta réduction pour KII :
(fun V5 -> V5)
Inférence de KII :
(fun V5 -> V5) ***TYPABLE*** avec le type (T12 -> T12)
Beta réduction pour S = λx.λy.λz.xzyz :
Inférence de S = λx.λy.λz.xzyz :
(fun x -> (fun y -> (fun z -> ((x z) (y z))))) ***TYPABLE*** avec le type ((T21 -> (T19 -> T18)) -> ((T21 -> T19) -> (T21 -> T18)))
Beta réduction NAT1 = (λx.(x+1))*3 :
Inférence de NAT1 = (λx.(x+1))*3 :
((fun x -> (x + 1)) 3) ***TYPABLE*** avec le type Nat
Beta réduction NAT2 = λx.(x+x) :
(fun x -> (x + x))
Inférence de NAT2 = λx.(x+x) :
(fun x -> (x + x)) ***TYPABLE*** avec le type (Nat -> Nat)
Beta réduction NAT3 = NAT2 * I :
((fun x -> x) + (fun x -> x))
Inférence de NAT3 = NAT2 * I :
((fun x -> (x + x)) (fun x -> x)) ***PAS TYPABLE*** : type fleche non-unifiable avec Nat
Beta réduction NAT3 = NAT2 * I :
((fun x -> x) + (fun x -> x))
Inférence de NAT3 = NAT2 * I :
((fun x -> (x + x)) (fun x -> x)) ***PAS TYPABLE*** : type fleche non-unifiable avec Nat
Beta réduction omega = (λx.xx)(λy.yy) :
((fun y -> (y y)) (fun y -> (y y)))
Inférence de omega = (λx.xx)(λy.yy) :
((fun x -> (x + x)) (fun x -> x)) ***PAS TYPABLE*** : type fleche non-unifiable avec Nat
---------------------------------- 3. PCF -------------------------------------------------
Beta réduction pour ex_let:
Inférence de ex_let:
Beta réduction pour ifzero:
2
Inférence de ifzero:
(ifzero (5 - 4) then 1 else 2) ***TYPABLE*** avec le type Nat
Beta réduction pour ifempty:
Inférence de ifempty:
(ifempty (tete [20]) then 5 else (tete [])) ***PAS TYPABLE*** : Les type ne sont pas les memes
---------------------------------- 4. Traits Imperatifs -----------------------------------------------
Inférence de ex_ref:
(ifzero (ref 5) then 1 else 2) ***PAS TYPABLE*** : type entier non-unifiable avec ref T44
Inférence de r:
((fun x -> x) (ref 2)) ***TYPABLE*** avec le type ref Nat
```