Programming language theory, 2020 spring.

Start date: 24.02.2019

Deadline: 29.02.2019, 9:00 (means before lesson)

Home work 3:

Exercise 1. using Church Booleans:

F=λx.λy.y, T=λx.λy.x

XOR= λa.λb.a(bFT)(bTF)
NOT= λa.aFT
AND= λa.λb.abF

Exercise 2. using Churh Numerals
pred *= λn.λy.λx.n(λw.λz.z(w y))(λg.x)(λg.g)*
minus *= λp.λq.(q pred)p*
isequal *= λp.λq.AND(GreaterOREqual(p q) GreaterOREqual(q p))*

due to GreaterOREqual = λp.λq.isZero(p pred q), iszero = λp.p(λa.F)T

Exercise 4. Implement factorial using
1) Y-combinator - λy.(λx.y(x x))

multiplication = λp.λq.λx(p(q x)
factorial = Y-combinator (λp.λq.p(multiplication q(pred q))F)

2) Z-combinator - λp(λq.p(λr.(q q)r))(λq.p(λr.(q q)r))

factorial = Z-combinator(λp.λq.p(λx.(multiplication q pred q)F))

I can't understand and check it by my own, so I tried to find more information about it in other sources.

```
Y(f) = (g => (x => g(x(x)))(x => g(x(x))))(f)
     |
     |
     = (x => f(x(x)))(x => f(x(x)))   1
     |
     |
     = f( (x => f(x(x)))(x => f(x(x))) )   2
     |
     |
     |    1 and 2 are exactly the same and 1 is equal to Y(f) as per first expression, therefore
     |
     |
     = f(Y(f))
```

```
Z(f) = (g => (x => g(v => x(x)(v))) (x => g(v => x(x)(v))))(f)
     |
     |
     = (x => f(v => x(x)(v))) (x => f(v => x(x)(v)))   1
     |
     |
     |
     = f(v => (x => f(v => x(x)(v)))(x => f(v => x(x)(v)))(v))   2
     |
     |
     |    1 and 2 are exactly the same and 1 is equal to Z(f) as shown by the second equivalence, therefore
     |
     |
     = f(v => Z(f)(v))
```

https://medium.com/swlh/y-and-z-combinators-in-javascript-lambda-calculus-with-real-code-31f25be934ec

The difference is that it introduces a function in its definition instead of a direct calculation of an expression.

```
Y = g => (x => g(x(x)))(x => g(x(x)))
```

The a expression in Y becomes the b function in Z

```
Z = g => (x => g(v => x(x)(v))) (x => g(v => x(x)(v)))
```

Comparison between Y and Z