TC - Lab 12

1

- $(\text{NOT } p) \equiv (\lambda p.\, p\ F\ T)$

  $p \equiv T$: $(\lambda p.\, p\ F\ T) \equiv (T\ F\ T) \equiv ((\lambda x \lambda y.\, x)\ F\ T) \equiv ((\lambda y.\, F)\ T) \equiv F$: $\text{NOT } T \xrightarrow{\beta} F$

  $p \equiv F$: $(\lambda p.\, p\ F\ T) \equiv (F\ F\ T) \equiv ((\lambda x.\, \lambda y.\, y)\ F\ T) \equiv ((\lambda y.\, y)\ T) \equiv T$: $\text{NOT } F \xrightarrow{\beta} T$

- Para la recursividad se utiliza el Y-combinador: $Y = \lambda f.\, (\lambda x.\, f\,(x\,x))\,(\lambda x.\, f\,(x\,x))$, que tiene la propiedad que $(Y\,g) \equiv g\,(Y\,g)$. De esta manera se construye el ciclo, al instanciar $(Y\,g)$ dentro de g mediante esta sustitución.

- Es prudente en bajo nivel, cuando no hay soporte nativo de cierta funcionalidad. Por ejemplo al construir un interprete minimal en Lisp

  No es prudente en programas de alto nivel que buscan legibilidad. Por ejemplo servicios web o aplicaciones con objetos.