UNIVERSIDAD DEL VALLE DE GUATEMALA
Teoría de la Computación
Sección 20

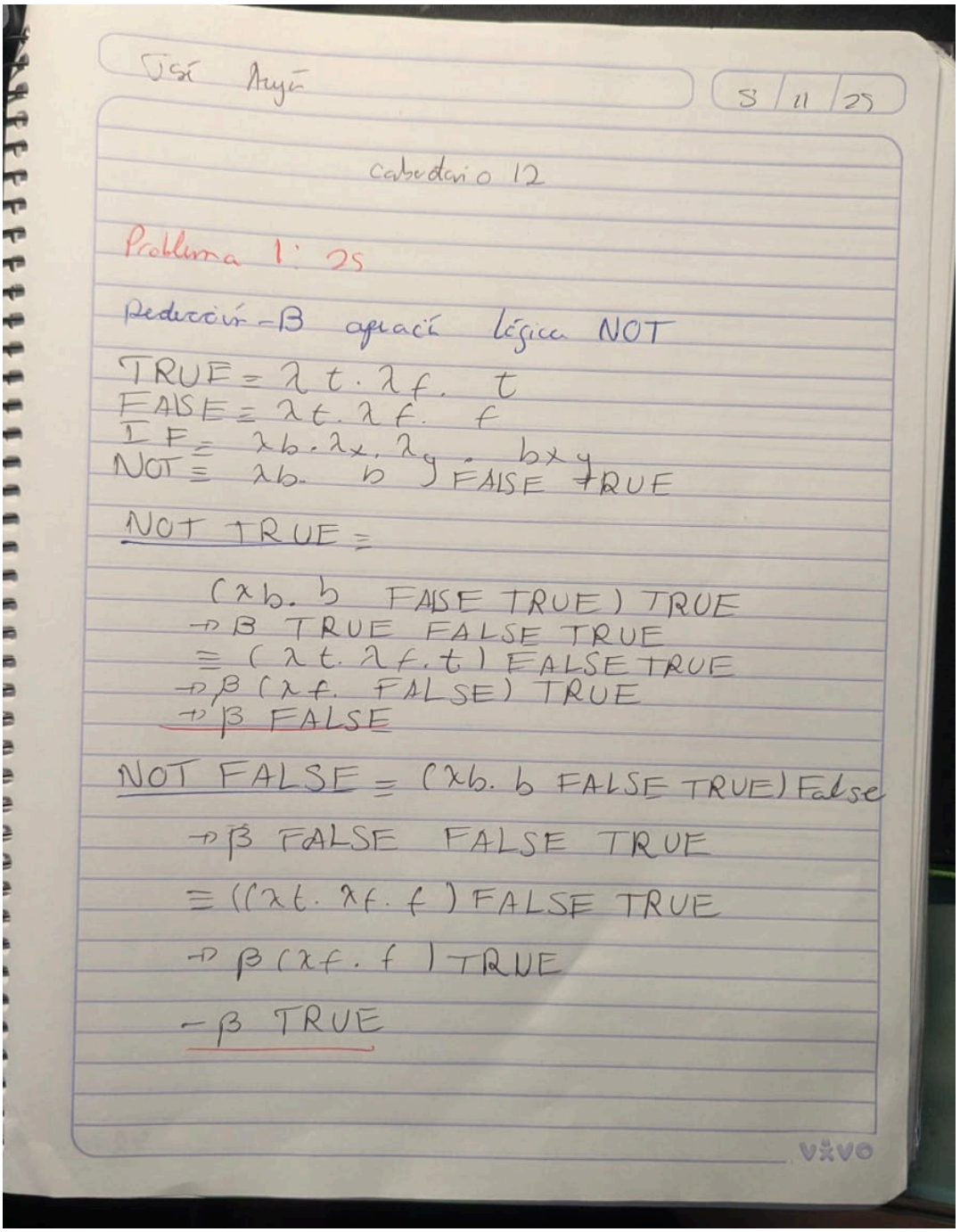
Proyecto #3

José Andrés Auyón Cóbar - 201579

Link de repositorio: https://github.com/auyjos/Lab12_TLC.git

Link de vídeo: https://youtu.be/Psh4hSg4c1c

1)



2) ¿Cómo se ve la recursión y los “ciclos” en λ-cálculo?

Recursión (combinador fijo Y)

El λ-cálculo puro no tiene nombres auto-referenciados, así que usamos un **combinador de punto fijo**:

Y ≡ λf. (λx. f (x x)) (λx. f (x x))

La idea: si definimos un “cuerpo” recursivo F que recibe su propia versión como primer argumento, entonces:

f ≡ Y F ; f = F f

Ejemplo (esqueleto factorial)

F ≡ λself. λn. IF (isZero n) 1 (mul n (self (pred n)))
fac ≡ Y F

Evaluar fac k desata las expansiones por Y que reinyectan self como fac, logrando la recursión.

“Ciclos” / iteración

La iteración se codifica con recursión. Un WHILE se puede representar como:

WHILE ≡ λcond. λbody.
Y (λloop. λs. IF (cond s) (loop (body s)) s)

- s es el estado.
- Si cond s es TRUE, aplica body y vuelve a llamar loop (otra iteración).
- Si es FALSE, retorna s.
  Esto modela un ciclo imperativo mediante llamadas recursivas.

No terminación (bucle infinito):
Ω ≡ (λx. x x) (λx. x x)
Ω se reduce para siempre (no normaliza), ejemplificando un “ciclo” sin fin.

3) ¿Cuándo conviene este estilo y cuándo no?

Conviene:

- **Razonamiento formal y corrección** (pruebas, intérpretes, verificación).
  Ejemplo: especificar y probar un optimizador de expresiones usando λ-términos y booleans de Church; es fácil demostrar propiedades por inducción estructural.

- **Transformaciones de programas/DSLs** y **compiladores** (claridad semántica).
  Ejemplo: una pasada de inlining y β-reducción sobre un mini-lenguaje funcional.

No conviene:

- **IO, concurrencia, y efectos del mundo real** sin un marco adicional
  Ejemplo: escribir un servidor HTTP de alto rendimiento directamente en λ-cálculo puro es impráctico.

- **Performance de bajo nivel** (control fino de memoria/CPU).
  Ejemplo: cómputo numérico intensivo donde necesitas SIMD/threads; usarías C/C++/Rust y luego se podrían modelar partes en un estilo funcional a alto nivel.

Usar el λ-cálculo/estilo funcional puro para especificar, razonar y transformar; baja a lenguajes con efectos controlados o a nivel de sistema para interacción con el entorno y rendimiento.

Referencias:

Abelson, H., & Sussman, G. J., with Sussman, J. (1996). *Structure and Interpretation of Computer Programs* (2nd ed.). MIT Press.
https://mitpress.mit.edu/9780262510875

Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics* (rev. ed.). North-Holland.

Barendregt, H. P., & Barendsen, E. (1998). *Introduction to Lambda Calculus* (lecture notes).
https://www.cs.ru.nl/~freek/courses/tt-2011/bb.pdf

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics, 58*(2), 345–363. https://doi.org/10.2307/2371045

Church, A. (1941). *The Calculi of Lambda-Conversion*. Princeton University Press.

Hindley, J. R., & Seldin, J. P. (2008). *Lambda-Calculus and Combinators: An Introduction* (2nd ed.). Cambridge University Press.

Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press.

Reynolds, J. C. (1993). Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation, 11*(4), 363–397. (Original work published 1972). https://doi.org/10.1007/BF00264248

Plotkin, G. D. (1975). Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science, 1*(2), 125–159. https://doi.org/10.1016/0304-3975(75)90017-1