Hola! Corregí su TP, la nota es I. Hay varios errores tanto en el código –especialmente los ejercicios 1 y 3– como en la demo, por lo que es necesario reentregar. No duden en hacer las consultas que necesiten.

A continuación les comento las correcciones en detalle, pero antes quería aclarar que podrían haber acotado el contenido de la entrega a lo mínimo indispensable:

- El markdown (ej9.md) no es necesario, con el PDF compilado alcanzaba.
- No es necesario entregar el PDF del enunciado.
- Les quedó un archivo `README.md` sin nada relevante.

Ejercicio 1

El enunciado aclara que el tipo de `procId` debe ser `Procesador a a`, y menciona que *"devuelve un único resultado, que contiene la estructura original completa"*, ¿por qué cambiaron el tipo a `Procesador [a] a` y lo definieron como `id`?

Ejercicio 2

- Para `foldAT` podían usar la sintaxis de `case` vista en clase, y un `where` para evitar la repetición de `foldAT fNil fTer` tres veces.
- Para `foldTrie` podrían haber usado un `where` para definir aparte la función del `map`, simplemente por una cuestión de legibilidad.

Ejercicio 3

- Para `unoxuno` usaron `foldr`, pero en esencia la recursión que hacen consiste en transformar individualmente cada elemento de la lista recibida. Les recomiendo entonces que lo repiensen usando `map` en vez de `foldr`.
- Para `sufijos` usaron recursión explícita pero no está permitido (excepto para el ejercicio 2). Tienen que reescribirlo sin usar recursión explícita.

Ejercicio 4

Acá lo que hicieron está bien, solo hay un detalle muy sutil para mencionar. A las variables `ri`, `rm` y `rd` les dieron esos nombres porque entiendo que se refieren a "recursión del árbol izquierdo/medio/derecho", y está muy bien. ¿Pero por qué llamaron a la raíz `rr` y no simplemente `r`? Es claramente distinta de las otras tres porque es la raíz misma y no "el resultado de la recursión de la raíz" (concepto de por sí erróneo porque no se trata de un parámetro recursivo).

Ejercicio 5

Para `preorderRose` no necesitan el `if` porque `concat []` reduce a `[]`.

Ejercicio 6

La implementación de `caminos` está bien pero no está escrita de la forma más óptima en términos de legibilidad y claridad. Les propongo que la repiensen sin usar listas por comprensión, y aprovechando `concatMap`, tal como hicieron con `palabras`.

De todas formas, la solución que dieron sería mucho más declarativa si eligieran mejores nombres para las variables que usan, en vez de `hijos` e `hijos'`. Podrían hablar de caminos, subcaminos, cabezas, etc. Si prefieren ir por esta alternativa, es válido también.

Ejercicio 8

Para la composición de procesadores podrían haber aprovechado el operador de composición `(.)` para tener una definición más compacta:

```
(.!) p1 p2 = (concatMap p1) . p2
```

Ejercicio 9

- Para el caso inductivo no están bien enunciadas las fórmulas lógicas intervinientes. Para empezar, arrancan escribiendo $P(i) \wedge P(m) \wedge P(d) \implies P(\text{Tern } r\ i\ m\ d)$, que está bien, pero faltan los para todos que ligan las variables `r`, `i`, `m` y `d`. Por otro lado, esos para todos sí los escribieron, pero solo para la hipótesis inductiva. Es decir, afirman que su HI es:

  $$\forall i :: ATa. \forall m :: ATa. \forall d :: ATa\ .\ (P(i) \wedge P(m) \wedge P(d))$$

  Esto es lógicamente equivalente a decir que vale $P(t)$ para todo arbol t. Es decir, su HI está afirmando lo que el enunciado pide demostrar. De manera similar, también está mal enunciada la tesis inductiva porque le aplican los para todos directamente. Tendrían que cambiar esta parte introductoria del paso inductivo teniendo en cuenta que lo que realmente quieren probar es que (dicho en castellano) **para toda raíz *m* y árboles *i*, *m* y *d*, si valen *P(i)*, *P(m)* y *P(d)* (esa es su HI), entonces vale *P(Tern r i m d)***. De todas formas, los pasos que siguieron posteriormente no fueron afectados por esto.
- Falta justificar los pasos donde reemplazan los `foldAT` por su definición para los casos recursivos (ecuación que nombraron `{f1}`).
- La regla β se usa para aplicar una lambda a un solo argumento. Si la aplican a cuatro argumentos, como es su caso, en realidad están aplicando la regla β cuatro veces, y habría que aclararlo en la justificación.
- Similarmente, el lema que demostraron lo están aplicando múltiples veces y no sólo una, porque lo están aplicando a una concatenación de tres listas (y cuatro cuando desarrollan el lado derecho), que, en realidad, es una concatenación de dos. Lo correcto sería hacer un paso por cada vez que aplican el lema marcando cuáles son las dos listas que están tomando (depende de cómo asocia el operador `++`).
- Al reemplazo de `elem x [r]` por `x == r` le falta una justificación/paso. Al aplicar {e1} les queda `x == r || elem x []`, pero dado que los principios para desarrollar las demostraciones deben seguirse de forma estricta, usando `{e0}` les quedaría `x == r || False`. Parece trivial pero deberían aclarar que `x == r || False` se puede reemplazar por `x == r` justificando "por Bool" o "propiedades de booleanos" (no hace falta demostrarlo). Similar a como citaron la conmutatividad de `||` en el último paso, que está bien.
- En la demostración del Lema 1, ¿por qué mencionan el principio de extensionalidad de funciones? ¿Dónde lo están aplicando?
- Similar a como comenté más arriba, para el caso inductivo del lema están enunciando mal la HI ya que la acompañan con un $\forall xs :: [a]$ adelante, que transforma su HI en lo que quieren demostrar.
- Faltaría un último paso más para el paso inductivo del lema, y es que las dos expresiones a las que llegaron del lado izquierdo y lado derecho son iguales por la asociatividad del operador `||`.

Saludos! Damián