Final PLP 18 feb 2021

Julián Braier, Mateo Marenco*

February 2021

Modalidad: Oral virtual de muuchos minutos (tipo 50), intenso.

1 Funcional

1.1 Curry

Me pidió que le explique qué hace la función uncurry y que la defina. Acá manqueé y le escribí curry, upsi.

==========

A mí me pidió que le explique el concepto de curry y por qué querrías hacerlo. La respuesta a lo último es para evaluar una función parcialmente.

1.2 Patrones de recursión

Se puede definir fold en función de map o filter? No, ni dan los tipos. Ponele que quiero escribir la suma con fold. Eso toma una lista de enteros y devuelve un entero sólo. Map y filter devuelven listas del mismo tipo que reciben.

Me dio el tipo:

Data D a = C (D a) (D a)

Cómo sería fold para esa estructura?

FoldD f (C izq der) = f () (hice hasta acá explicándole qué me faltaba y quedó contento con eso)

A qué evalúa esta cosa? foldD C miD. Rta: es la identidad para el tipo D a. Escrita de manera chancha.

==========

A mí me dijo "suponete que alguien me dice que da lo mismo hacer estas dos cosas, ¿es verdad?"

```
foldl f g =?= foldr f g
```

Le dije que en el caso general no, pero bajo ciertas condiciones sí, como que la f debería ser conmutativa y capaz asociativa (todavía no sé jajajaja), y además en algunos casos la lista tendría que ser finita. Lo confundió un poco lo de conmutativa y pensó que no había entendido nada (creo que pensó que había

*No guarda relación con Javier Marenco

entendido que en foldr las listas se evalúan de derecha a izquierda), entonces me pidió que desarrolle la evaluación de foldl f g [1,2,3] y por otro lado la de foldr f g [1,2,3]. Después para aclarar lo que había querido decir con lo de la f conmutativa le di la siguiente f de ejemplo:

```
sumaPesada x y = x + 2*y
```

y le dije que no da lo mismo foldl sumaPesada 0 [1,2,3] que foldr sumaPesada 0 [1,2,3].

Después me preguntó si con foldr puedo escribir cualquier función recursiva sobre listas. Me resultó un poco rara la pregunta pero me pareció que quería que le hable de recr, y de que a veces es necesario tener el resto de la lista para hacer el cómputo. Le dije se puede simular recr con foldr haciendo el truquito de mantener una tupla que en una componente tiene lo que querés calcular y en el otro el resto de la lista.

1.3 Extensiones de cálculo lambda

==========

Me dijo que a un lambda cálculo con Bool y Nat lo queremos extender para usar Ints, o sea que haya también negativos. Lo primero que dijo fue que ahora $pred^n(0)$ debería ser un valor. Me preguntó qué cambios habría que hacer con las reglas de evaluación para que todo funcione bien.

Le dije que habría que quitar la regla pred(0) $\rightarrow$ 0, y también agregar una regla que diga que succ(pred(V)) $\rightarrow$ V para que no se trabe la evaluación de una expresión como succ(pred(0)). Dijo que estaba bien aunque le pareció que faltaba algo, pero en el momento no se le ocurrió qué podía ser y seguimos con otra cosa.

1.4 Fix

Cómo definirías fix en Haskell? Eso funcionaría en Lambda Cálculo? Por qué no?

1.5 Subtipado

Ref es contravariante? Por qué no? Hablamos un toque sobre Sink y Source.

Me dio los siguientes dos tipos y me preguntó si alguno de los dos es subtipo del otro.

- (Ref tau) $->$ Bool
- (Sink tau) $->$ Bool

Rta: (Sink tau) $->$ Bool <: (Ref tau) $->$ Bool

==========

Me hizo hablar un poco de la diferencia entre subtipado y subtipado algorítmico. No lo sabía, y entonces me preguntó por qué querría eliminar la

regla T-Sub y cómo hacerlo sin perder subtipado (ahora supongo que justamente la diferencia entre subtipado y subtipado algorítimo es que la primera usa T-sub y la otra no).

Le respondí que T-Sub genera no determinismo lo cual es problemático para implementar subtipado. Si quitamos T-Sub lo que hay que hacer es "enchufar" el subtipado en las demás reglas, por ejemplo en T-App decir que si una función espera un tau y le pasás un sigma está todo bien siempre y cuando sigma <: tau. Me preguntó exactamente qué otras cosas había que cambiar en un cálculo lambda con Bool y Nat. No le supe responder pero la respuesta era simplemente hacer cambios similares al de T-App para T-isZero, T-Succ, T-Pred; o sea decir que es válido recibir un Bool en vez de un Nat en esos casos.

1.6 Inferencia

Me dio esta regla:

$$\frac{\Gamma, x:\sigma \mid - M:\tau \;\; \Gamma, x:\tau \mid - M:\sigma}{\Gamma \mid - x + M : (\sigma, \tau)}$$

Tenía que dar el algoritmo de inferencia para esa regla.

Me quedé flasheado y me tuvo que ayudar un poquito. Pero la respuesta era esta:

W(x + U) = S $(\Gamma' - \{x : \rho_x\}) \mid - S(x + M) : S(\rho, \rho)$

W(U) = $\Gamma' \mid - M : \rho$

$$\rho_x = \begin{cases} \tau_x & \text{si } x : \tau_x \in \Gamma' \\ t \; variable \; fresca & \text{si } no \end{cases} \tag{1}$$

S = MGU$\{\rho_x = \rho\}$

==========

Me dijo suponete que tenemos un operador nuevo M*N con la siguiente regla, definime cómo sería el algoritmo de inferencia en ese caso

```
    Gamma|- M : tau    Gamma, x: tau |- N: sigma
-------------------------------------------------
         Gamma |- M * N : sigma
```

Empecé a escribir un poco pero en un momento me trabé y terminó quedando incompleto, pero igualmente es muy parecido al caso de inferencia para abstracciones.

2 Lógica

Me preguntó algo así como qué podía pasar si cambiábamos el orden de búsqueda de prolog.

Me dio este programita:

p(X):- q(X).

p(1).

q(X):- r(X),!.
q(b).
r(a).
r(b).
Qué responde prolog a esta consulta?
p(W).
rta:
W = a; W = 1; false.
==========

Me dijo que había un programa en prolog cuyo código fuente no conocíamos pero que sabemos que si le mandábamos el goal $\neg p(X)$ el resultado es primero X=1 y después se cuelga. Me preguntó qué pasa si evaluamos not(p(X)). Le dije más o menos lo que hace el not de prolog y que entonces la respuesta es que devuelve false porque prolog encuentra la instanciación válida X=1 y ahí ya corta y retorna false, o sea que no cambia nada que después se hubiera colgado.

Después, siguiendo con el mismo ejemplo, me preguntó cuál sería el resultado de hacer la consulta

```
not(not(p(X))), X==2.
```

La respuesta es que como el not de adentro retorna false, el de afuera retorna true. Además el not nunca instancia variables, o sea que la X no queda instanciada después de ese punto. Después cuando llega a X==2 simplemente encuentra la única instanciación válida X=2, la retorna y termina ahí.

3 Objetos

Diferencia entre los ligadores λ y σ.

Relacionar con javascript (o sea, hablar de self y que siempre referencia al objeto que recibe el mensaje).

Puedo extraer un método de un objeto? Qué pasa con self en ese caso?