Paradigmes i Llenguatges de Programació

Pràctica de Programació Funcional
Lambda-càlcul amb Haskell

8 de maig de 2025

Funcions i tipus λ-càlcul clàssic [8 punts]

En aquesta pràctica s'hauran de treballar els aspectes de λ-càlcul vistos a classe, implementant, amb Haskell, les funcions necessàries per tal de poder trobar la forma normal dels λ-termes que en tinguin.

A part de les funcions i tipus auxiliars que es puguin requerir, s'hauran de definir/implementar els següents tipus i funcions:

- el tipus `LT`, per a poder representar λ-termes. És a dir, definiu un `data`.
- el tipus `Substitucio`, pot ser un `data` o un un `type`.
- la funció `subst`, que rebi un `LT` i una `Substitucio` i retorni el terme on se li ha aplicat la substitució (sense captura). Per fer aquesta funció haureu d'implementar altres funcions com `freeVars` i `boundVars`, tals que donat un `LT` ens retornin la llista de variables lliures i lligades del terme (potser millor una funció `freeAndboundVars` que ens retorni una tupla amb les dues llistes?).
- la funció `esta_normal`, que rebi un `LT` i digui si està o no en forma normal.
- la funció `beta_redueix`, que rebi un `LT` que sigui un redex i el resolgui.
- la funció, `redueix_un_n`, que rebi un `LT` i retorni l'`LT` resultant de fer la primera β-reducció segons l'ordre normal.
- la funció, `redueix_un_a`, que rebi un `LT` i retorni l'`LT` resultant de fer la primera β-reducció segons l'ordre aplicatiu.
- la funció `l_normalitza_n`, que rebi un `LT` i retorni una llista de `LT`s amb la seqüència de reducció fins arribar a la forma normal (si en té) seguint l'ordre de reducció normal.
- la funció `l_normalitza_a`, que rebi un `LT` i retorni una llista de `LT`s amb la seqüència de reducció fins arribar a la forma normal (si en té i l'ha trobat) seguint l'ordre de reducció aplicatiu.

- la funció `normalitza_n`, que rebi un `LT` i retorni una tupla amb el nombre de passos de reducció, seguint l'ordre de reducció normal, necessaris fins arribar a la forma normal, i la seva forma normal `LT`, si en té.

- la funció `normalitza_a`, que rebi un `LT` i retorni una tupla amb el nombre de passos de reducció, seguint l'ordre de reducció aplicatiu, necessaris fins arribar a la forma normal, i la seva forma normal `LT`, si en té i l'ha trobat.

Implementeu també les definicions del meta-llenguatge vistes a classe fins a poder avaluar funcions "recursives" com el factorial. **Fins a quin factorial podeu calcular en 1 minut de temps?** Podeu mirar de fer les funcions de β-reducció més eficients?

Caldrà fer `LT` instància de `Show` per tal de mostrar λ-termes, per exemple, $\lambda x_1.\lambda x_2.x_1(x_2\, x_3)$ s'hauria de mostrar això com a `LT` (**el format ha de ser aquest)**:

```
(\x1. (\x2. (x1  (x2 x3))))
```

També caldrà fer `LT` instància de `Eq`, de manera que dos `LT` seran iguals **si són α-equivalents** (fixeu-vos que no podem implementar la igualtat com a λ-equivalència perquè no és decidible).

Se us acudeix alguna noció d'ordre entre `LT` per tal de poder fer el tipus `LT` instància d'`Ord`? Implementeu-ho.

Extra: Notació *de Bruijn* [2 punts]

Els noms de les variables en el λ-càlcul poden acabar siguent una mica complicats si bé "els noms" ens si mateix no tenen cap rellevància. En realitat, podríem veure les ocurrències de les variables lligades com a un punter a la λ-abstracció que l'està lligant. Per exemple, interpretant que **l'ordre de les λ-abstraccions és relatiu a l'ocurrència de la variable**, és a dir, que per una variable, la primera λ-abstracció és la que te més aprop, en el λ-terme $\lambda x.\lambda y.y\, x$, la y està "apuntant" a la primera λ-abstracció mentres que la x a la segona. Això es podria representar el λ-terme anterior com a $\lambda.\lambda.0\,1$. Fixeu-vos que el número representa la distància en nombre de λs a la λ-abstracció que la lliga.

La gramàtica per a λ-termes en notació *de Bruijn* és la següent:

$$\Lambda ::= \mathcal{N} | (\lambda.\Lambda) \ | (\Lambda\, \Lambda)$$

Les variables es representen amb naturals ($\mathcal{N}$) que es refereixen a l'índex (relatiu a la seva ocurrència) de la λ-abstracció que la lliga. Les λ-abstraccions no necessiten especificar el nom de la variable que lliguen (no hi ha noms de variables!) i les aplicacions són com les hem vist fins ara.

Aquí mostrem diversos exemples de λ-termes i la seva corresponent notació *de Bruijn*:

$\lambda x.x$	$\lambda.0$
$\lambda z.z$	$\lambda.0$
$\lambda x.\lambda x.x$	$\lambda.\lambda.0$
$\lambda x.\lambda y.x$	$\lambda.\lambda.1$
$\lambda x.\lambda y.\lambda s.\lambda z.x\ s\ (y\ s\ z)$	$\lambda.\lambda.\lambda.\lambda.3\,1\,(2\,1\,0)$
$(\lambda x.x\ x)\ (\lambda x.x\ x)$	$(\lambda.0\,0)(\lambda.0\,0)$
$\lambda x.\lambda y.x\ (\lambda z.x\ y\ z)\ y$	$\lambda.\lambda.1\ (\lambda.2\,1\,0)\,0$

Fixeu-vos bé en el darrer exemple on es pot veure que una mateixa variable lligada pot aparèixer amb diferents nombres, alhora que nombres iguals no tenen perquè correspondre a la mateixa variable lligada.

Però, què passa amb les variables lliures? Per poder representar les variables lliures necessitarem un *context* que associi variables lliures a nombres i evitar captures. Assumim que els contextos sempre associen les variables lliures amb nombres consecutius començant per zero. Per exemple, suposem que el context Γ ens diu que x és 0 i que y és 1, la representació en *de Bruijn* segons Γ del λ-terme $(x\,y)$ seria $(0\,1)$ però la de $\lambda z.x\,y\,z$ seria $\lambda.1\,2\,0$. Fixeu-vos que s'ha hagut d'incrementar els valors per a x i y per tal de no capturar.

Definiu el tipus `LTdB` per a λ-termes en notació *de Bruijn*. Feu-lo instància de la classe `Eq` (α-equivalència) i de la classe `Show` (com abans, useu \ enlloc de λ).

Finalment, feu també:

- la funció `a_deBruijn` que rebi un `LT` i un `Context` (definiu-ne el tipus) el passi a `LTdB`.
- la funció `de_deBruijn` que rebi un `LTdB` i el passi a `LT` (fixeu-vos que també caldrà retornar un context).

Per aquestes dues funcions us serà útil utilitzar algun paràmetre addicional (comenceu tractant el cas sense variables lliures).

Lliurament

- Les pràctiques s'han de fer en parelles. No cal que siguin les mateixes que a la pràctica de Prolog. Teniu un fórum específic per triar parelles per aquesta pràctica.
- No estan permeses les copies ni l'ús d'eienes d'IA generativa (tipus copilot o ChatGPT).
- No està permès utilitzar cap mòdul extern (és a dir, no es pot utilitzar `import`). Incomplir aquesta restricció suposarà treure un 0. Sí que es poden utilitzar les funcions del `Prelude`, com per exemple `map,filter,foldr,foldl,sum,and,...`. En cas de dubte, pregunteu.
- Heu de lliurar un informe **detallat** en **PDF** que contingui tot el codi, amb el tipus de les funcions i amb les explicacions necessaris per a entendre-les. Cal que el codi sigui llegible. Utilitzeu formats que ajudin a la llegibilitat. Cal que al document hi hagi exemples d'execució per cada funció. Cal que expliqueu els aspectes més interessants del vostre codi.

Utilitzeu funcions d'ordre superior sempre que pugueu, el codi ha de ser funcional i s'ha d'evitar repeticions.

- A més del PDF, també heu d'entregar el fitxer de codi (un sol fitxer). El codi ha de compilar correctament, altrament la nota serà 0.
- Cal que implementeu totes les funcions que es demanen al document (en podeu afegir d'auxiliars), i que el nom de la funció i l'ordre dels paràmetres sigui exactament el que s'ha descrit.
- Data lliurament: **5 de juny**. El termini és estricte.
- Es pot demanar a qualsevol grup que faci una entrega presencial de la seva pràctica si es té dubtes sobre la originalitat o si cal més informació per a la correcció. S'espera que els dos membres del grup siguin capaços d'explicar la pràctica entregada, i de tornar implementar parts del codi entregat.