Programació funcional

Albert Rubio

Especialitat de Computació
Grau en Enginyeria Informàtica

FIB

Pla de la sessió

- Presentació del tema
- Fonaments: λ-càlcul
- Fonaments: λ-càlcul amb tipus simples
- Presentació de Haskell
- Definicions senzilles
- Pattern matching

Presentació

Aplicacions

- Microsoft. SLAM (OCaml). F# a .NET o Visual Studio
- Jane Street Capital (Wall Street), OCaml.
- Credit Siusse, Haskell, F#.
- Barclays, Haskell.
- EDF Trading (energia), Scala.
- Ericsson, Erlang, Haskell; Nokia també
- Facebook, Google, Haskell
- Twitter, Scala
- Conf.: Commercial Users of Functional Programming

Presentació

Aplicacions

- Derivative pricing.
- Data analysis (per exemple, fulls de càlcul).
- Hardware description languages.
- Geospatial search engine (geographical information).
- Desenvolupament de Software crític.
- ...

Presentació

Objectius

- Conèixer un llenguatge funcional pur
- No assignació (no efectes laterals)
- Haskell
- Conèixer els fonaments del model de càlcul
- Pattern matching, lazy evaluation
- Sistema de tipus potent, ordre superior.
- Base per altres Llenguatges de Programació Funcionals: OCaml, Erlang, F#, Scala, ...

Fonaments: λ-càlcul

- Model de computació funcional.
- Alonzo Chuch, 1932
- Totes les funcions recursives poden ser representades en λ-càlcul (Kleene,Rosser 1936)
- El λ-càlcul pot expressar exactament les funcions computables per una Màquina de Turing (Turing 1937).
- Abstracció i aplicació (variables lligades i substitució)
- Origen del llenguatges funcionals

Fonaments: λ-càlcul

Construcció de λ-termes. Dues "operacions":

- Abstracció: $\lambda x.u$ on u és un λ-terme.
- Aplicació: $(u\ \ v)$ on u i v són λ-termes.
  Associa a l'esquerra:
  $(((u\ \ v_1)\ \ v_2)\ \ \ldots\ \ v_n) = (u\ \ v_1\ \ v_2\ \ \ldots\ \ v_n)$

Intuïció: funcions matemàtiques.
$f(x, y, z) = x^2 + 2y + z + 4$
és
$f = \lambda x.\lambda y.\lambda z.x^2 + 2y + z + 4$

i

$f(6, 3, 12)$ és $(f\ \ 6\ \ 3\ \ 12)$

Fonaments: λ-càlcul

Computació. Una sola regla:

- β-reducció: $(\lambda x.u\ \ v) \rightarrow_\beta u[x := v]$

Intuïció:

$(\lambda x.\lambda y.\lambda z.x^2 + 2y + z + 4\ \ 6\ \ 3\ \ 12) \rightarrow_\beta^3$

$x^2 + 2y + z + 4[x := 6][y := 3][z := 12] =$

$6^2 + 2 \cdot 3 + 12 + 4 = 58$

Variables

- Lligades. Variables que apareixen en una abstracció. En $\lambda x.(y\ \ \lambda z.(x\ \ z))$, x i z són lligades
- Lliures. Les demés (a l'exemple y). Es poden substituir!

Intuició: integrals/derivades $\int_a^b x^2 + y + 2\ \ dx$

Fonaments: λ-càlcul

Substitució:

- $x[x := w]$ és w
- $y[x := w]$ és y si $x \neq y$
- $(u\ \ v)[x := w]$ és $(u[x := w]\ \ v[x := w])$
- $(\lambda x.u)[x := w]$ és $\lambda x.u$
- $(\lambda y.u)[x := w]$ és $\lambda y.(u[x := w])$ si $x \neq y$ i y no lliure a w.

- $(\lambda y.u)[x := w]$ si $x \neq y$ i y apareix lliure a w.
  NO DEFINIT! Versió original inconsistent!

Equivalència:

- α-conversió: $\lambda x.u =_\alpha \lambda y.u[x := y]$ si y és nova

Fonaments: λ-càlcul

Exemples:

- Funció identitat: $I = \lambda x.x$
- Funció projecció: $K = \lambda x.\lambda y.x$
- Funció composition: $S = \lambda x.\lambda y.\lambda z.(x\ \ z\ \ (y\ \ z))$
- Auto aplicació: $w = \lambda x.(x\ \ x)$
- No terminació: $\Omega = (w\ \ w)$

$SKK\ =\ I$

$\Omega = (\lambda x.(x\ \ x)\ \ \lambda x.(x\ \ x)) \rightarrow_\beta (x\ \ x)[x := \lambda x.(x\ \ x)] =$
$(\lambda x.(x\ \ x)\ \ \lambda x.(x\ \ x)) = \Omega \rightarrow_\beta \ldots$

- *Combinatory logic*. Només S i K són Turing-complets!
  Extensionalitat: $\lambda x.(ux) =_\eta u$ si x no lliure a u

Fonaments: λ-càlcul

Exemple: codificació dels nombres naturals

- 0: $c_0 = \lambda x.\lambda y.x = \lambda x\ y.y$
- n: $c_n = \lambda x.\lambda y.(\underbrace{x...(x}_{n}\ y)) = \lambda x\ y.x^n(y)$
- increment: $A_s = \lambda z\ x\ y.(x\ (z\ x\ y))$
- suma: $A_+ = \lambda p\ q\ x\ y.(p\ x\ (q\ x\ y))$
- producte: $A_* = \lambda p\ q\ x\ y.(p\ (q\ x)\ y)$

- $(A_s\ c_n) \rightarrow_\beta c_{n+1}$
- $(A_+\ c_n\ c_m) \rightarrow_\beta c_{n+m}$
- $(A_*\ c_n\ c_m) \rightarrow_\beta c_{n \cdot m}$

Fonaments: λ-càlcul

Propietats de la β-reducció:

- β-reducció és confluent.

  Si $t \to_\beta \ldots \to_\beta t_1$ i $t \to_\beta \ldots \to_\beta t_2$ llavors
  $t_1 \to_\beta \ldots \to_\beta t_3$ i $t_2 \to_\beta \ldots \to_\beta t_3$

- estratègia normalitzant de la β-reducció:

  left-most outer-most.

  Redueix la λ sintàcticament més a l'esquerra.

  Si existeix un terme que no es pot reescriure més, aquesta estratègia el troba!

Fonaments: λ-càlcul amb tipus

Només considerarem tipus simples

- a la Curry
  $\lambda x.x : \alpha \to \alpha$
- a la Church
  $\lambda x : \alpha.x : \alpha \to \alpha$

Declaracions

- A la Curry el tipus dels paràmetres es dedueixen del context.
- Cal declarar el tipus de las variables lliures.

Fonaments: λ-càlcul amb tipus

Exemple: Tipus simples: només tipus bàsics i funcionals.

Sigui Γ un context (declaració) de la forma $\{x : \alpha, y : \beta\}$ un λ-terme t és tipable en Γ si

$$\Gamma \vdash t : \sigma$$

es pot deduir de les següents regles (a la Curry)

Abstracció:

$$\frac{\Gamma \cdot \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash (\lambda x.t) : \sigma \to \tau}$$

Variable:

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

Aplicació:

$$\frac{\Gamma \vdash s : \sigma \to \tau \quad \Gamma \vdash t : \sigma}{\Gamma \vdash (s\ \ t) : \tau}$$

Fonaments: λ-càlcul amb tipus

Exemple: Tipus simples: només tipus bàsics i funcionals.

Sigui Γ un context (declaració) de la forma $\{x : \alpha, y : \beta\}$ un λ-terme t és tipable en Γ si

$$\Gamma \vdash t : \sigma$$

es pot deduir de les següents regles (a la Church)

Abstracció:

$$\frac{\Gamma \cdot \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash (\lambda x{:}\sigma.t) : \sigma \to \tau}$$

Variable:

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

Aplicació:

$$\frac{\Gamma \vdash s : \sigma \to \tau \quad \Gamma \vdash t : \sigma}{\Gamma \vdash (s\ \ t) : \tau}$$

Fonaments: λ-càlcul amb tipus

$\lambda x.(x\ \ x)$ no és tipable.

Exemple de derivació de tipus

$$\dfrac{\dfrac{\dfrac{\{f:\beta\to\alpha, x:\beta\}\vdash f:\beta\to\alpha \qquad \{f:\beta\to\alpha, x:\beta\}\vdash x:\beta}{\{f:\beta\to\alpha, x:\beta\}\vdash (f\ \ x):\alpha}}{\{f:\beta\to\alpha\}\vdash \lambda x.(f\ \ x):\beta\to\alpha}}{\emptyset\vdash \lambda f\lambda x.(f\ \ x):(\beta\to\alpha)\to\beta\to\alpha}$$

- Si t es tipable amb aquest sistema la β-reducció acaba.
- El λ-càlcul amb tipus simples no és Turing complet
- Cal afegir polimorfisme y definició de noves funcions

Presentació de Haskell

- Llenguatge funcional pur. No assignacions.
  No gestió memòria explícita.
- Lazy evaluation: tractar estructures molt grans o infinites.
- Sistemes de tipus potents: *propositions as types*.
  Tipus polimòrfics. Inferència automàtica.
- Funcions d'ordre superior. Funcions com a paràmetres.
  Gestió de llistes!
- Al 1987 degut a la proliferació de FPLs és decideix definir un Standard: HASKELL
- Al 1998 es crea una versió estable Haskell98.

Presentació de Haskell

- Usarem `ghc`. *Glasgow Haskell Compiler*.
  Compilació separada: obtenir codi objecte
  Muntar executable.

- Usarem `ghci`. Com a interpret. Carregant el programa.
  Executant per línia de comandes.

  Inicialment usarem l'interpret

Presentació de Haskell

Comandes ghci,
“:comanda” d’entre les que hi ha per exemple:

- :? ,per invocar el Help.
- :l <nomarxiu> ,per carregar un script.
- :r ,per repetir l’ultima acció.
- :type <expr> ,per consultar el tipus d’una expressió.
- :quit ,per sortir.
- ...

Ho treballarem al laboratori.

Definicions senzilles

- Les definicions de funcions comencen amb minúscula.
- Les funcions es podran definir: amb if-then-else, amb guardes, per “pattern matching”.
- Les funcions es podran definir: per recursivitat o directament.
- Poden tenir una declaració de tipus:
  nomf:: t1->t2->. . .->tn
- L’avaluació d’una expressió es fa aplicant les definicions.

Definicions senzilles

Tipus predefinits bàsics: Bool, Char, Int, Integer, Float

- `Bool: True, False, &&, ||, not`
  Per exemple, podem definir `xOr` com
  ```
  xOr :: Bool -> Bool -> Bool
  xOr x y = ( x || y ) && not ( x && y )
  ```
- Char: entre cometes simples ’’
  Alguns caràcters especials:

  `’\t’, ’\n’,...`

  Funcions de conversió (Data.Char):

  - `ord :: char -> Int`
  - `chr :: Int -> char.`

Definicions senzilles

Tipus predefinits bàsics: Bool, Char, Int, Integer, Float

- Int: `+` , `-` , `*` , `^` (infix)
  `div`, `mod`, `abs` i `negate` (prefix). `‘div‘` (infix)
  - Acotats per 2147483647. Precisió arbitrària: Integer.
  - Podem passar a Float amb: fromInt.
- Float: reals però amb precisió limitada.

  Operacions estàndard més

  ^ :: Float ->Int -> Float i **:: Float ->Float -> Float.

  Conversió: ceiling, floor i round :: Float -> Int.
- Double: per a tenir doble precisió.
- Ratio: (racionals) fraccions d’Integers amb precisió arbitrària.

Definicions senzilles

Tipus estructurats predefinits:

- Tupla: $(camp_1, \ldots, camp_n)$
- Llista: [],(cap:cua), l1++l2.
  sumar :: [Int] -> Int
  sumar [] = 0
  sumar (x:xs) = x+sumar(xs)
  Seqüències aritmètiques:

  ```
  [1..5], ['a',..,'z']
  [1,..] [1,3..]
  ```

  Llistes per comprensió: [<exp> | <q1>, ..., <qn>]
  Qualifiers q1,...,qn:
  Generadors: x <- [elems]
  Filtres: expressió booleana

Definicions senzilles

Tipus estructurats predefinits:

- Exemples:

```
[ x*x | x <- [1..10], even x ]
[ ( x, y ) | x <- [ 1, 2, 3], y <- [ 9, 10, 11] ]
[ ( x, y ) | x <- [ 1, 2, 3], y <- [ 1 ..x] ]
```

- pot ser més eficient un canvi d'ordre:

```
[ ( x, y ) | x <- [ 1 .. 3 ], y <- [ 1 ..2 ], even x ]
[ ( x, y ) | x <- [ 1 .. 3 ], even x, y <- [ 1 ..2 ] ]
```

Definicions senzilles

Per definir variables locals usarem

- *let*:
  let $v_1 = E_1 \ldots v_n = E_n$ in E
  És una expressió i té el tipus d'E.

```
f n =
    let x = (div n 2)
    in 2* (f x)
```

- *where*
  No és una expressió!
  Es posa al final per afegir les definicions que falten.

```
f n = 2* (f x)
  where x = (div n 2)
```

Es poden definir també funcions noves que s'han usat.

Pattern matching

Definició de funcions (simbòlica):

sumar [] = 0
sumar (x:xs) = x+sumar(xs)

expr1 *matches* expr2 si existeix una substitució per les variables de expr1 que la fan igual que expr2.

Exemple:
x:xs *matches* [2,5,8]
Substitució: x=2; xs=[5,8]

Exercicis

1. màxim d'una llista d'enters.
2. donada una llista retornar la llista dels dobles.
3. donada una llista d'enters retornar dues llistes, una que conté els parells i una que conté els senars.
4. fibonacci (eficient).
5. inserir un element en una llista ordenada.
6. comprovar si és primer un nombre.
7. retornar la llista de divisors primers d'un nombre.