Llenguatges de Programació

lambda càlcul

λ

Albert Rubio, Jordi Petit, Fernando Orejas, Gerard Escudero

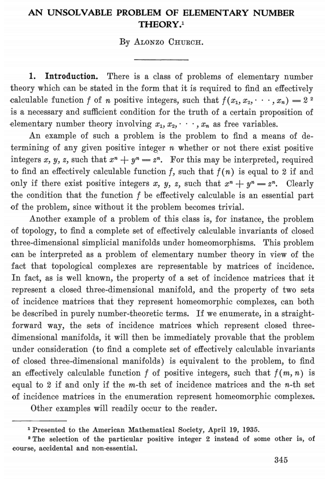
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Contingut

Introducció

El **λ-càlcul** és un model de computació funcional, l'origen dels llenguatges funcionals, i la base de la seva implementació.

Inventat per Alonzo Church, cap al 1930.



AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.[1]

By Alonzo Church.

1. Introduction. There is a class of problems of elementary number theory which can be stated in the form that it is required to find an effectively calculable function f of n positive integers, such that $f(x_1, x_2, \cdots, x_n) = 2$ [2] is a necessary and sufficient condition for the truth of a certain proposition of elementary number theory involving $x_1, x_2, \cdots, x_n$ as free variables.

An example of such a problem is the problem to find a means of determining of any given positive integer n whether or not there exist positive integers x, y, z, such that $x^n + y^n = z^n$. For this may be interpreted, required to find an effectively calculable function f, such that $f(n)$ is equal to 2 if and only if there exist positive integers x, y, z, such that $x^n + y^n = z^n$. Clearly the condition that the function f be effectively calculable is an essential part of the problem, since without it the problem becomes trivial.

Another example of a problem of this class is, for instance, the problem of topology, to find a complete set of effectively calculable invariants of closed three-dimensional simplicial manifolds under homeomorphisms. This problem can be interpreted as a problem of elementary number theory in view of the fact that topological complexes are representable by matrices of incidence. In fact, as is well known, the property of a set of incidence matrices that it represent a closed three-dimensional manifold, and the property of two sets of incidence matrices that they represent homeomorphic complexes, can both be described in purely number-theoretic terms. If we enumerate, in a straightforward way, the sets of incidence matrices which represent closed three-dimensional manifolds, it will then be immediately provable that the problem under consideration (to find a complete set of effectively calculable invariants of closed three-dimensional manifolds) is equivalent to the problem, to find an effectively calculable function f of positive integers, such that $f(m, n)$ is equal to 2 if and only if the m-th set of incidence matrices and the n-th set of incidence matrices in the enumeration represent homeomorphic complexes.

Other examples will readily occur to the reader.

[1] Presented to the American Mathematical Society, April 19, 1935.

[2] The selection of the particular positive integer 2 instead of some other is, of course, accidental and non-essential.

345

Consisteix en agafar una línia de símbols i aplicar una operació de *cut-and-paste*.

$$(\lambda y.x(yz))\ (ab)$$

$$\Downarrow$$

$$x(abz)$$

Fotos: Fair Use, jstor.org, Lambda Calculus for Absolute Dummies

Contingut

- Introducció
- Estructura bàsica
    - Components
    - Avaluació
    - Macros
- Codificacions de Church
- Recursivitat
- Universalitat
- Calculadores

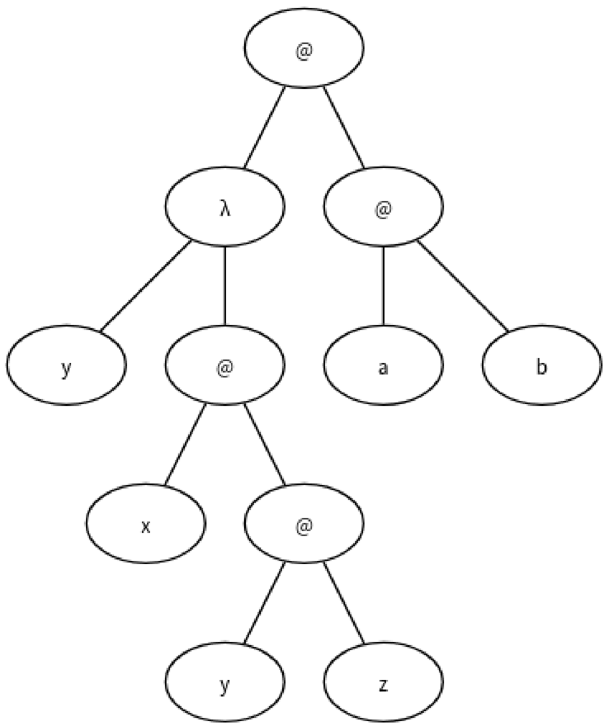
Gramàtica

```
      terme   →  lletra  |  ( terme )  |  abstracció  |  aplicació
abstracció   →  λ  lletra  .  terme
 aplicació   →  terme  terme
```

Exemples de termes:

- x
- $\lambda x.\, x$
- $(\lambda y.\, x(yz))(ab)$

Arbre de $(\lambda y.\, x(yz))(ab)$:



Gramàtica

Les lletres es diuen *variables* i no tenen cap significat. El seu nom no importa. Si dues variables tenen el mateix nom, són la mateixa cosa.

Els parèntesis agrupen termes. Per claredat, s'agrupen per l'esquerra:

$$abcd \equiv (((ab)c)d).$$

La λ amb el punt introdueix funcions. Per claredat, es poden agrupar λs:

$$\lambda x.\, \lambda y.\, a \equiv \lambda x.\, (\lambda y.\, a) \equiv \lambda xy.\, a$$

Operacions

Només hi ha dues operacions per la construcció de termes:

- L'**abstracció** captura la idea de definir una funció amb un paràmetre:

  $$\lambda x.\, u$$

  on u és un terme.

  Diem que λx és el *cap* i que u és el *cos*.

  Intuïció: $f(x, y) = x^2 + 2y + x - 1$ és representat per $\lambda x.\, \lambda y.\, x^2 + 2y + x - 1$

- L'**aplicació** captura la idea d'aplicar una funció sobre un paràmetre:

  $$f\ x$$

  on f i x són dos termes.

  Intuïció: $f(x)$ és representat per $f\,x$

Currificació

Al λ-càlcul totes les funcions tenen un sol paràmetre.

Les funcions que normalment consideraríem que tenen més d'un paràmetre es representen com a funcions d'un sol paràmetre utilitzant la tècnica del *currying*:

- Una funció amb dos paràmetres, com ara la suma, +: int x int ⟶ int, es pot considerar equivalent a una funció d'un sol paràmetre que retorna una funció d'un paràmetre, +: int ⟶ (int ⟶ int).
- Això vol dir que $2 + 3$, amb notació prefixa $(+2\ 3)$, s'interpretaria com $(+2)\ 3$, on $(+2)$ és la funció que aplicada a qualsevol paràmetre x, retorna $x + 2$.

Currificar és transformar una funció que accepta n paràmetres i convertir-la en una funció que, donat un paràmetre (el primer) retorna una funció que accepta $n - 1$ paràmetres (i són semànticament equivalents).

Contingut

- Introducció
- Estructura bàsica
    - Components
    - Avaluació
    - Macros
- Codificacions de Church
- Recursivitat
- Universalitat
- Calculadores

Computació

La **β-reducció** (*cut-and-paste*) és la regla essencial de computació del λ-càlcul:

$$(\lambda x.\, u)\; v \longrightarrow_{\beta} u[x := v]$$

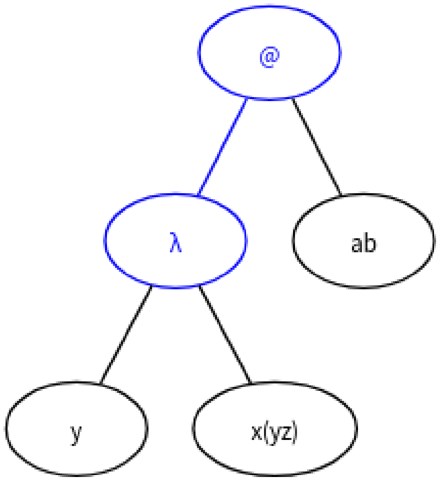
on $u[x := v]$ vol dir reescriure u substituint les seves x per v.

Exemple:

expressió	**acció efectuada**
$(\lambda y.\, x(yz))(ab)$	β-reducció de y
$x((ab)z)$	$\equiv$
$x(abz)$	

No cal que aparegui el patró a l'arrel de l'arbre.

Patró de la β-reducció.



Forma normal

Si una expressió no pot β-reduir-se, aleshores es diu que està en **forma normal**.

Si $t \longrightarrow \ldots \longrightarrow t'$ i t' està en forma normal, aleshores es diu que t' és la forma normal de t, i es considera que t' es el resultat de l'avaluació de t.

Una λ-expressió té, com a màxim, una forma normal.

Variables lliures i lligades

Dins d'un terme, una variable és **lligada** si apareix al cap d'una funció que la conté. Altrament és **lliure**.

Les variables poden ser lliures i lligades alhora en un mateix terme.

Per exemple:

$$(\lambda x.\, xy)(\lambda y.\, y)$$

- y és lliure a la primera subexpressió.
- y és lligada a la segona subexpressió.

El problema de la captura de noms I

Quan s'aplica la β-reducció s'ha de tenir cura amb els noms de les variables i, si cal, reanomenar-les.

El problema es pot veure en el següent exemple: Sigui TWICE:

$$\lambda f\,.\,\lambda x.\ f\,(f\,x)$$

Calculem $(\mathrm{TWICE}\ \mathrm{TWICE})$:

expressió	**acció efectuada**
$\mathrm{TWICE}\ \mathrm{TWICE}$	definició de TWICE
$(\lambda f\,.\,\lambda x.\ f\,(f\,x))\ \mathrm{TWICE}$	β-reducció de f
$(\lambda x.\ \mathrm{TWICE}(\mathrm{TWICE}\ x))$	definició de TWICE
$(\lambda x.\ \mathrm{TWICE}(\lambda f\,.\,\lambda x.\ f\,(f\,x))x)$	

El problema de la captura de noms II

Aplicant la β-reducció directament tindríem:

expressió	**acció efectuada**
$(\lambda x.\, \mathrm{TWICE}(\lambda f\,.\, \lambda x.\, f\,(f\,x))\ x)$	β-reducció de f
$(\lambda x.\, \mathrm{TWICE}(\lambda x.\, x(xx)))$	**ERROR**

El que hauríem de fer és reanomenar la variable lligada x mes interna:

expressió	**acció efectuada**
$(\lambda x.\, \mathrm{TWICE}(\lambda f\,.\, \lambda x.\, f\,(f\,x))\ x)$	canvi de nom $x \longrightarrow y$
$(\lambda x.\, \mathrm{TWICE}((\lambda f\,.\, \lambda y.\, f\,(f\,y))\ x)$	β-reducció de f
$(\lambda x.\, \mathrm{TWICE}((\lambda y.\, x(xy))$	**OK**

α-Conversió

A més de la β-reducció, al λ-càlcul tenim la regla de l'α-conversió per reanomenar les variables. Per exemple:

$$\lambda x.\lambda y.\, xy \longrightarrow_{\alpha} \lambda z.\lambda y.\, zy \longrightarrow_{\alpha} \lambda z.\lambda t.\, zt$$

Aleshores l'exemple del TWICE el podríem escriure:

expressió	**acció efectuada**
TWICE TWICE	definició de TWICE
$(\lambda f.\lambda x.\, f(f\,x))$TWICE	β-reducció de f
$(\lambda x.$ TWICE(TWICE x))	definició de TWICE
$(\lambda x.$ TWICE$(\lambda f.\lambda x.\, f(f\,x))\; x)$	α-conversió $[x/y]$
$(\lambda x.$ TWICE$((\lambda f.\lambda y.\, f(f\,y))\; x)$	β-reducció de f
$(\lambda x.$ TWICE$((\lambda y.\, x(xy))$	

Ordres de reducció

Donada una λ-expressió, pot haver més d'un lloc on es pot aplicar β-reducció, per exemple:

$$(1)\quad (\lambda x.\, x((\lambda z.\, zz)x))t \longrightarrow t((\lambda z.\, zz)t) \longrightarrow t(tt)$$

però també:

$$(2)\quad (\lambda x.\, x((\lambda z.\, zz)x))t \longrightarrow (\lambda x.\, x(xx))t \longrightarrow t(tt)$$

Hi ha dues formes estàndard d'avaluar una λ-expressió:

- Avaluació en **ordre normal**: s'aplica l'estratègia **left-most outer-most**: Reduir la λ sintàcticament més a l'esquerra (1).

- Avaluació en **ordre aplicatiu**: s'aplica l'estratègia **left-most inner-most**: Reduir la λ més a l'esquerra de les que són més endins (2).

Ordres de reducció

En principi, podríem pensar que no importa l'ordre d'avaluació que utilitzem, perquè la β-reducció és **confluent**:

Si $t \rightarrow \cdots \rightarrow t_1$ i $t \rightarrow \cdots \rightarrow t_2$ llavors
$t_1 \rightarrow \cdots \rightarrow t_3$ i $t_2 \rightarrow \cdots \rightarrow t_3$

Tanmateix, si una expressió té una forma normal, aleshores la reducció en ordre normal la trobarà, però no necessàriament la reducció en ordre aplicatiu.

Per exemple, en ordre normal tenim:

$$(\lambda x.\, a)((\lambda y.\, yy)(\lambda z.\, zz)) \longrightarrow a$$

però en ordre aplicatiu:

$$(\lambda x.\, a)((\lambda y.\, yy)(\lambda z.\, zz)) \longrightarrow (\lambda x.\, a)((\lambda z.\, zz)(\lambda z.\, zz)) \longrightarrow \ldots$$

Contingut

- Introducció
- Estructura bàsica
    - Components
    - Avaluació
    - Macros
- Codificacions de Church
- Recursivitat
- Universalitat
- Calculadores

Macros

En el λ-càlcul, les funcions no reben noms.

Per facilitar-ne la escriptura, utilitzarem **macros** que representen funcions i les expandirem quan calgui, com vam fer a les transparències anteriors amb TWICE.

Les macros també es diuen **combinadors**.

⇒ És un recurs "meta" que no forma part del llenguatge (preprocessador).

Exemple: ID $\equiv \lambda x.\, x$

Llavors:

expressió	**acció efectuada**
ID ID	definició ID
$(\lambda x.\, x)$ ID	β-reducció de x
ID	

Contingut

- Introducció
- Estructura bàsica
- Codificacions de Church
  - Booleans
  - Naturals
  - Enters
- Recursivitat
- Universalitat
- Calculadores

Codificació de Church

Com es representen els tipus dades i els seus operadors en λ-càlcul.

Naturals en λ-càlcul: Una codificació estranya?

Dec	Bin	Romà	Xinès	Devanagari	λ-càlcul
0	0		零	०	$\lambda sz.\, z$
1	1	I	一	१	$\lambda sz.\, sz$
2	10	II	二	२	$\lambda sz.\, s(sz)$
3	11	III	三	३	$\lambda sz.\, s(s(sz))$
4	100	IV	四	४	$\lambda sz.\, s(s(s(sz)))$
⋮				⋮	

El natural n és l'aplicació d'n cops la funció s a z.

L'important no és com es representen els naturals, sinó establir una bijecció entre la seva representació i $\mathbb{N}$.

Tampoc estem considerant-ne l'eficiència.

Contingut

- Introducció
- Estructura bàsica
- Codificacions de Church
  - Booleans
  - Naturals
  - Enters
- Recursivitat
- Universalitat
- Calculadores

Booleans I

Church encoding*:

- $T \equiv \lambda t.\lambda f.t$ el primer
- $F \equiv \lambda t.\lambda f.f$ el segon

Com fem el *not*?

- $not \equiv \lambda g.gFT$ "flip"
- $not\ T \longrightarrow (\lambda g.gFT)T \longrightarrow TFT \longrightarrow \ldots \longrightarrow F$ el primer
- $not\ F \longrightarrow (\lambda g.gFT)F \longrightarrow FFT \longrightarrow \ldots \longrightarrow T$ el segon

Exercici: completar les β-reduccions.

* Church encoding - Wikipedia

Booleans II

Com fem el condicional?

- $if \equiv \lambda c.\, \lambda x.\, \lambda y.\, cxy$

el 1er o el 2on?

- **Exercicis**: codificar i avaluar:
  - `if F then poma else pera`
  - `if T then poma else pera`

Com fem l'*and*?

- `and x y = if x then y else F`
- $and \equiv \lambda x.\, \lambda y.\, xyF$
- **Exercici**: demostreu l'anterior.

I l'*or*?

- ...

Contingut

Funcions aritmètiques bàsiques

- $0 \equiv \lambda s.\, \lambda z.\, z \equiv F$
- $n \equiv \lambda s.\, \lambda z.\, s^n z$

Com fem el *succ*?

- $succ = \lambda n.\, \lambda f.\, \lambda x.\, f(n f x)$ — 1a f per afegir, $f\,x$ per consumir
- **Exercici**: avalueu
  - $succ\ 1$

Com fem la *suma*?

- $suma \equiv \lambda m.\, \lambda n.\, n\ succ\ m$ — *succ* per cada f de la n
- **Exercici**: avalueu — aplica n vegades *succ* a m
  - $suma\ 2\ 1$

Més funcions aritmètiques

Altres:

- $mul \equiv \lambda m.\lambda n.\lambda f.n(mf)$
- $power \equiv \lambda m.\lambda n.nm$
- **Exercici**:

  penseu el perquè, interpreteu-les

Avançats:

- $minus \equiv \lambda m.\lambda n.n\ pred\ m$
- $pred \equiv \lambda n.\lambda f.\lambda x.n(\lambda g.\lambda h.h(gf))(\lambda u.x)(\lambda u.u)$

Predicats

isZero?

- $isZero \equiv \lambda n.\, n(\lambda x.\, F)T$ 0 consumeix F i és queda la T
- **Exercici**: avalueu n es queda F i $\lambda x.\, F$ descarta la resta
    - $isZero\ 0$
    - $isZero\ 2$

Relacionals:

- $leq \equiv \lambda m.\, \lambda n.\, IsZero\ (minus\ m\ n)$
- $eq \equiv \lambda m.\, \lambda n.\, and\ (leq\ m\ n)(leq\ n\ m)$

Contingut

Tuples

Parells:

- $pair \equiv \lambda x.\,\lambda y.\,\lambda p.\, pxy$
- Exemple: $pair\ 2\ 3 \equiv \lambda p.\, p\ 2\ 3$

Accés:

- $first \equiv \lambda p.\, p(\lambda x.\,\lambda y.\, x)$
- $second \equiv \lambda p.\, p(\lambda x.\,\lambda y.\, y)$

Exercici: avalueu

- $first\ (pair\ 2\ 3)$
- $second\ (pair\ 2\ 3)$

Enters

Els codifiquem amb una resta en un parell:

```
 2 = pair 2 0
-3 = pair 0 3
```

Funcions:

- $convert \equiv \lambda x.\ pair\ x\ 0$ natural a enter
- $neg \equiv \lambda x.\ pair\ (second\ x)(first\ x)$

S'utilitza una funció $oneZero$ per generar parells amb almenys un zero (amb recursivitat, Y).
Totes les funcions aritmètiques es generen tenint el compte els parells.

De la mateixa forma els racionals són parells d'enters.
Les llistes també es codifiquen a partir de parells (com en Lisp).

Contingut

- Introducció
- Estructura bàsica
- Codificacions de Church
- Recursivitat
- Universalitat
- Calculadores

Recursivitat

Combinador Y (paradoxal o de punt fix):

$$Y \equiv \lambda y.\,(\lambda x.\,y(xx))(\lambda x.\,y(xx))$$

Compleix la propietat:

$$YR \equiv R(YR)$$

Demostració:

expressió	**acció efectuada**
Y R	definició de Y
$(\lambda y.\,(\lambda x.\,y(xx))(\lambda x.\,y(xx)))\mathrm{R}$	β-reducció de y
$(\lambda x.\,\mathrm{R}(xx))(\lambda x.\,\mathrm{R}(xx))$	β-reducció de x
$\mathrm{R}((\lambda x.\,\mathrm{R}(xx))(\lambda x.\,\mathrm{R}(xx)))$	per aquest resultat i l'anterior
$R(YR)$	

Factorial I

El combinador Y ens permet definir la funció factorial. Sigui:

$$H \equiv \lambda f\,.\,\lambda n.\,\mathrm{IF}(n = 0)\ 1\ (n \times (f\ \ (n - 1)))$$

podem veure com YH funciona com el factorial:

expressió	**acció efectuada**
$YH1$	combinador Y
$H(YH)1$	definició de H
$(\lambda f\,.\,\lambda n.\,\mathrm{IF}(n = 0)1(n \times (f(n - 1))))(YH)\ 1$	β-reducció de f
$(\lambda n.\,IF(n = 0)1(n \times (YH(n - 1))))\ 1$	β-reducció de n
$IF(1 = 0)1(1 \times (YH(1 - 1)))$	$IF = fals$
$1 \times (YH(1 - 1)))$	trivial
$YH0$	combinador Y
...	

Factorial II

expressió	**acció efectuada**
$YH0$	combinador Y
$H(YH)0$	definició de H
$\lambda f.\lambda n.\mathrm{IF}(n = 0)1(n \times (f(n-1)))(YH)\ 0$	β-reducció de f
$\lambda n.\mathrm{IF}(n = 0)1(n \times (YH(n-1)))\ 0$	β-reducció de n
$\mathrm{IF}(0 = 0)1(0 \times (YH(0-1)))$	$IF = cert$
1	

Contingut

- Introducció
- Estructura bàsica
- Codificacions de Church
- Recursivitat
- Universalitat
- Calculadores

Universalitat del λ-càlcul

A partir d'aquí, ja només queda anar continuant fent definicions i anar-les combinant.

Eventualment, es pot arribar a veure que qualsevol algorisme és implementable en λ-càlcul perquè pot simular a una màquina de Turing.

Teorema [Kleene i Rosser, 1936]: Totes les funcions recursives poden ser representades en λ-càlcul ($\Longleftrightarrow$ Turing complet).

A diferència de les màquines de Turing que són un model matemàtic d'una màquina *hardware* imperativa, el λ-càlcul només utilitza reescriptura i és un model matemàtic més *software* i funcional.

λ-càlcul amb tipus: existeixen extensions amb tipus; que són les que solen utilitzar els llenguatges funcionals com model.

Contingut

- Introducció
- Estructura bàsica
- Codificacions de Church
- Recursivitat
- Universalitat
- Calculadores

Calculadores

Existeixen moltes calculadores de λ-càlcul *online*:

- https://www.cl.cam.ac.uk/~rmk35/lambda_calculus/lambda_calculus.html
- https://jacksongl.github.io/files/demo/lambda/index.htm
- http://www-cs-students.stanford.edu/~blynn/lambda/ (amb notació Haskell)