Scuola Superiore Universitaria
di Toppo Wassermann

Classe Scientifico-Economica

Anno Accademico 2022/23

Predecessori nel Lambda Calcolo

Allievo
Alessandro Minisini

Relatore
Prof. Fabio Alessi

Indice

1 Introduzione

Il lambda calcolo o λ-calcolo è un sistema formale definito nel 1936 da Alonzo Church, sviluppato per analizzare formalmente le funzioni e il loro calcolo. Le prime sono espresse per mezzo di un linguaggio formale, che stabilisce quali siano le regole per formare un termine, il secondo con un sistema di riscrittura - chiamato riduzione - che definisce come i termini possano essere semplificati.

In questo elaborato verranno esposte le basi del lambda calcolo, elencando le operazioni principali e gli assiomi che definiscono la *riduzione*, ovvero l'elemento computante del formalismo. L'attenzione verrà poi spostata sui sistemi di numerazione e sulle loro funzioni di base, come il successore e il predecessore. Sarà poi quest'ultimo a dominare la scena: ci si concentrerà sulle varie soluzioni proposte per risolvere il problema del predecessore per i numerali di Church.

Lo scopo principale della tesina, infatti, è descrivere vari punti di attacco al problema citato: si vedrà un approccio basato su coppie di numerali (proposto da Kleene), uno sull'estensione dei numerali, uno su alcune proprietà invarianti della sequenza citata e un altro ancora sull'invertire l'applicazione, operazione fondante del λ-calcolo.

Nell'ultima parte, verrà descritto un isomorfismo tra due sistemi di enumerazione, utile per la costruzione di un rudimentale predecessore.

2 Lambda Calcolo e Notazione

Il *lambda calcolo* è una teoria che nasce dalla necessità di fornire una teoria generale sulle funzioni e, di conseguenza, sulle basi della matematica (almeno in parte). La nozione di funzione viene principalmente descritta in due modi: come *regola* - ovvero un processo di calcolo da un argomento ad un valore, che viene descritto attraverso una definizione - e come *grafo* - ovvero un insieme di coppie argomento/valore. Il lambda calcolo descrive le funzioni come regole, col fine di sottolinearne gli aspetti computazionali.

Le due operazioni fondamentali del lambda calcolo sono *l'applicazione* e *l'astrazione*. L'applicazione consiste, appunto, nell'applicare una funzione ad un argomento (questa operazione, tra la funzione f e l'argomento a, viene scritta come fa). L'astrazione, che potremmo considerare complementare all'applicazione, consiste nell'usare dei termini variabili che permettono, poi, l'applicazione di argomenti: $f := \lambda x.t(x)$ è la funzione f che assegna all'argomento a il valore $t(a)$, ovvero

$$(\lambda x.t(x))\ a = t(a).$$

Le funzioni rappresentate nel lambda calcolo vengono anche chiamate *termini*.

Definizione 2.1. I lambda termini sono parole definite sul seguente alfabeto:

$v_0, v_1, \ldots$ variabili,

λ astrattore,

(,) parentesi.

L'insieme dei lambda termini viene indicato con Λ.

Definizione 2.2. Λ è definito induttivamente come segue:

1. $x \in \Lambda$;
2. $M \in \Lambda \Rightarrow (\lambda x.M) \in \Lambda$;
3. $M,\ N \in \Lambda \Rightarrow (MN) \in \Lambda$;

dove x in (1) e (2) è una variabile qualsiasi.

L'applicazione è considerata associativa a sinistra, ovvero $MNS = (MN)S$. Espressioni del tipo $\lambda x.MN$ sono da intendersi come $\lambda x.(MN)$: il corpo dell'astrazione si estende il più a destra possibile; se necessario si usano le parentesi. Le astrazioni del tipo $\lambda x.\lambda y.M$ possono essere condensate in $\lambda xy.M$.

2.1 Relazioni e formule

La relazione di equivalenza principale sui λ-termini è la *β-conversion*. Questa relazione è definita per assiomi. Per la formulazione di quest'ultimi, è necessario fornire un operatore di sostituzione: $M[x := N]$ indica il risultato del sostituire N al posto di x in M.

Definizione 2.3. Il lambda calcolo presenta formule del tipo

$$M = N$$

dove $M, N \in \Lambda$ ed è definita dai seguenti assiomi:

- $(\lambda x.M)N = M[x := N]$, ($\beta$-conversion)
- $M = M$;
- $M = N \Rightarrow N = M$;
- $M = N, N = L \Rightarrow M = L$;
- $M = N \Rightarrow MZ = NZ$;
- $M = N \Rightarrow ZM = ZN$;
- $M = N \Rightarrow \lambda x.M = \lambda x.N$ (regola ξ)

Al concetto di uguaglianza è legato il concetto di riduzione. In questa tesina ci si sofferma sulla riduzione β, una relazione binaria su Λ.

Definizione 2.4. La β-conversion induce le seguenti relazioni:

$\rightarrow_\beta$ β-riduzione in un passo,

$\twoheadrightarrow_\beta$ β-riduzione,

$=_\beta$ β-uguaglianza (anche detta β-convertibilità).

La riduzione $\rightarrow_\beta$ corrisponde alla β-conversion, infatti

$$(\lambda x.M)N \rightarrow_\beta M[x := N].$$

La riduzione $\twoheadrightarrow_\beta$ è la chiusura transitiva e riflessiva di $\rightarrow_\beta$, mentre $=_\beta$ β è la relazione di equivalenza generata da $\twoheadrightarrow_\beta$.

La β-riduzione è molto studiata nell'ambito del lambda calcolo in quanto caratterizza la veridicità di una formula nella teoria stessa.

Proposizione. $M =_\beta N \Leftrightarrow M = N$

2.2 Combinatori e lunghezza di un termine

Il lambda calcolo non possiede tipi, costanti o operazioni. È conveniente, quindi, definire dei termini (chiamati combinatori) che rendano più leggibili le espressioni che verranno usate in seguito.

Definizione 2.5. (Identità) $\mathbf{I} := \lambda x.x$,

(Punto fisso) $\mathbf{Y} := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$,

(Vero) $\mathbf{T} := \lambda xy.x$,

(Falso) $\mathbf{F} := \lambda xy.y$,

(Coppia) $(M, N) := \lambda z.zMN$,

(Proiezioni) $(P)_0 := P\mathbf{T}, \qquad (P)_1 := P\mathbf{F}$,

(Composizione) $M \circ N := \lambda x.M(Nx)$.

Per concludere il capitolo, viene definita la lunghezza di un termine: essa coincide con il numero di variabili, applicazioni e lambda al suo interno. Ad esempio, il termine Coppia ha lunghezza 4 - numero di lambda = 1, numero di variabili = 1 (la z), numero di applicazioni = 2 ($(z \leftarrow M) \leftarrow N$).

3 Sistemi di numerazione

Definizione 3.1. Un *sistema di numerazione* è una sequenza

$$d = d_0, d_1, \ldots$$

costituita da termini chiusi tali che, per dei lambda termini S_d^+ e $\mathtt{Zero}_d$

$$S_d^+ d_n = d_{n+1},$$
$$Zero\mathtt{Zero}_d\ d_0 = \mathbf{T}, \quad \mathtt{Zero}_d\ d_{n+1} = \mathbf{F}$$

per ogni $n \in \mathbb{N}$. S_d^+ e $\mathtt{Zero}_d$ sono detti *successore* e *test per lo zero* di d.

Notare che un sistema di numerazione è definito da d_0 e S_d^+.

3.1 Un primo esempio

Prima di parlare dei più noti *numerali di Church*, è doveroso notare che è possibile definire vari sistemi di numerazione all'interno del lambda calcolo. Uno di questi, ad esempio, è il seguente:

Definizione 3.2. Per ogni $n \in \mathbb{N}$, il termine $\hat{n}$ è definito come segue

$$\hat{0} = \mathbf{I}, \quad \widehat{n+1} = (\mathbf{F}, \hat{n}).$$

mentre successore e test per lo zero sono definiti in questo modo

$$S^+ := \lambda x.(\mathbf{F}, x), \quad \texttt{Zero} := \lambda x.x\mathbf{T}$$

Notare che la definizione di `Zero` coincide con quella di proiezione.

Avendo a disposizione il termine di proiezione, è molto semplice definire il predecessore di questi numerali:

$$P^- := \lambda n.(n)_0 = \lambda n.n\mathbf{F}.$$

3.2 Numerali di Church

I numerali di Church sono il sistema di numerazione più noto e sono stati codificati da Alonzo Church nel 1936. Ogni numero n viene rappresentato dall'n-esima applicazione di una funzione f ad un argomento, x.

Definizione 3.3. Sia $c = c_0, c_1, \ldots$ con

$$c_n := \lambda fx.f^n(x)$$

dove f^n indica l'applicazione della funzione f reiterata per n volte. Ad esempio, i numeri $0, 1, 2$ sono rappresentati nel seguente modo:

$$c_0 := \lambda fx.x \quad c_1 := \lambda fx.fx, \quad c_2 = \lambda fx.f(fx).$$

Definire il successore è immediato, infatti si nota che:

$$c_{n+1} = \lambda fx.f^{n+1}x =_\beta \lambda fx.f(f^n x) =_\beta \lambda fx.f(c_n fx).$$

Quindi $S_c^+ := \lambda nfx.f(nfx)$.

4 Predecessori

Per i numerali di Church, non è così immediato definire né il test per lo zero né il predecessore. Il fatto principale, che sta alla base della maggior parte dei predecessori successivamente analizzati, è il seguente:

$$c_n = c_n \ S_c^+ \ c_0.$$

Ovvero, il numerale c_n è l'n-esima applicazione del successore S_c^+ a c_0.

4.1 Coppie (Kleene)

Sia $p = p_0, p_1, \ldots$ una successione così definita:

$$p_0 := (c_0, c_0), \quad p_1 := (c_0, c_1), \quad \ldots \quad , p_n = (c_{n-1}, c_n).$$

p_n rappresenta quindi la coppia dei termini c_{n-1} e c_n. A p_0 viene asseggnata la coppia (c_0, c_0) perché $P_c^- \ c_0$ non è ben definito. Il successore di queste coppie è semplice da formulare

$$S_p^+ := \lambda p.((p)_1, S_c^+((p)_1))$$

Avendo p_0 e successore, è semplice dare il lambda termine di un generico p_n, dal quale è semplice estrarre c_{n-1}:

$$P_c^- = \lambda n.(n \ S_p^+ \ p_0)_0 \tag{1}$$

La forma normale di P_c^- è la seguente.

$\lambda n.n \ (\lambda ps.s \ (p(\lambda xy.y)) \ (\lambda fx.f \ (p(\lambda xy.y)f \ x))) \ (\lambda p.p \ (\lambda fx.x) \ (\lambda fx.x)) \ (\lambda xy.x)$

La lunghezza di questo termine è 41.

4.2 Estensione

Nell'Equazione 1, S_p^+ ha una coppia come argomento, sebbene venga utilizzata solo la sua seconda componente. Questo indica che usare una coppia non è efficiente. Si ponga quindi $p_{n+1} := c_n$. Occorre trovare un termine p_0 e un termine S_p^+ che vadano bene per la sequenza, ovvero che soddisfino $p_{n+1} = c_{n+1} \ S_p^+ \ p_0$. Quello che si sta cercando, quindi, è un modo per calcolare c_n partendo da c_{n+1}.

Per raggiungere questo obiettivo, si *estende* la sequenza p includendola in una più grande.

Siano

$$\texttt{None} := \lambda ky.y \qquad \texttt{Some} := \lambda xky.kx.$$

Questi due termini definiscono l'estensione di p, chiamata $\hat{p}$: infatti $\hat{p}_0 := \texttt{None}$, mentre $\hat{p}_1 := \texttt{Some}\ c_0, \ldots$. In generale:

$$\hat{p}_n\ k\ y = \begin{cases} y & \text{se } n = 0 \\ k\ c_{n-1} & \text{altrimenti} \end{cases} = \begin{cases} y & \text{se } n = 0 \\ k\ (S_c^{+(n-1)} c_0) & \text{altrimenti} \end{cases} \tag{2}$$

Il termine $\hat{p}_{n+1}$ non rappresenta propriamente c_n ma $\texttt{Some}\ c_n$, dal quale possiamo sempre ricavare c_n.

L'operazione S_p^+, per ottenere $\hat{p}_n$, è la seguente:

$$S_p^+ := \lambda p.\texttt{Some}(p\ S_c^+\ c_0).$$

Questa, assieme a $\hat{p}_0$, permette di definire la formula per $\hat{p}_n$ e, di conseguenza, un nuovo predecessore per c_n:

$$P_c^- := \lambda n.(n\ S_p^+\ \hat{p}_0)\ \mathbf{I}\ c_0.$$

La forma normale del termine è la seguente:

$$\lambda n.n\ (\lambda pky.k\ (p\ (\lambda nfx.f\ (n\ f\ x))\ (\lambda fx.x)))\ (\lambda ky.y)\ (\lambda x.x)\ (\lambda fx.x).$$

La sua lunghezza è 35, leggermente più corta di quella relativa al predecessore di Kleene.

Parametrizzando il successore e il termine per $n = 0$, nell'Equazione 2, si ottiene che

$$\hat{p}_{nfx} := \lambda k. \begin{cases} x & \text{se } n = 0 \\ k\ (f^{(n-1)} x) & \text{altrimenti} \end{cases} \tag{3}$$

Ancora una volta, c_n può essere convertito in p_{n+1},e viceversa. Il primo elemento della nuova sequenza e il suo successore sono rispettivamente:

$$\hat{p}_{0fx} := \lambda k.x \qquad S_{pfx}^+ := \lambda pk.k\ (p\ f)$$

dai quali si può ricavare l'intera sequenza $\hat{p}_{fx}$. Il predecessore che se ne deriva è il seguente:

$$\lambda nfx.(n\ S_{pfx}^+\ \hat{p}_{0fx})\ \mathbf{I}$$

La sua forma normale, di lunghezza 18, è

$$\lambda nfx.n\ (\lambda pk.k\ (p\ f))\ (\lambda k.x)\ (\lambda y.y)$$

Questo è il predecessore più corto conosciuto, ed è circa la metà di quello di Kleene.

4.3 Proprietà invarianti

Come visto nella Sezione 4.2, è conveniente far corrispondere alla sequenza $p_1, p_2, \ldots$ una riscrittura dei valori $c_0, c_1, \ldots$: a p_{n+1} è stato assegnato il valore `Some` c_n. In questa sezione, invece, le due sequenze andranno a coincidere: tutta l'attenzione sarà posta sul trovare un termine p_0 che permetta, poi, di generare tutti i numerali di Church.

È utile distinguere alcuni invarianti dei numerali. Ad esempio, per ogni c_n, l'applicazione $c_n\ \mathbf{I}$ si riduce sempre a $\mathbf{I}$; l'identità è un punto fisso dei numerali di Church. Per sfruttare questo fatto, si utilizza un p_0 che si comporti diversamente, ad esempio $\lambda fx.c_0$.

La nuova sequenza sarà generata da

$$p_0 := \lambda fx.c_0 \qquad S_p^+ := \lambda p.p\ \mathbf{I}\ (S_c^+\ p)$$

e porta al seguente predecessore:

$$\lambda n.n\ (\lambda p.p\ \mathbf{I}\ (S_c^+\ p))(\lambda fx.c_0)$$

la cui forma normale corrisponde a

$$\lambda n.n\ (\lambda p.p\ (\lambda x.x)(\lambda fx.f\ (p\ f\ x)))\ (\lambda fxsz.z).$$

La lunghezza di questo predecessore è 24, circa la metà di quello di Kleene.

A differenza del precedente, l'applicazione che discrimina p_0 da p_{n+1} - l'applicazione all'identità - richiede tempo lineare per essere calcolata. È possibile modificare il test affinché la sua riduzione si possa fare in tempo costante. La prima modifica, consiste nello spostare l'onere computazionale nel calcolo di più successori.

$$\lambda n.n\ (\lambda p.p\ (\lambda z.S_c^+(p))\ \mathbf{I})\ (\lambda fx.c_0)$$

Questo sembra peggiorare il numero di riduzioni della computazione, ma da questo termine è più semplice ottimizzare. Innanzitutto, sostituiamo S_c^+ con la sua forma normale

$$\lambda n.n\ (\lambda p.\underline{p}\ (\lambda zfx.f\ (\underline{p\ f}\ x))\ \mathbf{I})\ (\lambda fx.c_0)$$

e, siccome il termine p (sottolineato) funge da iteratore, si può eliminare p ed f dalla parentesi più interna (i due termini sottolineati) e si può rimpiazzarli con un'astrazione generica g. Per non perdere l'astrazione su f, la variabile viene portata nel termine più esterno.

$$\lambda n.n\ (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ \mathbf{I})\ (\lambda fx.c_0)$$

I termini $\mathbf{I}$ e $(\lambda fx.c_0)$ vengono eliminati: questi servivano nel vecchio predecessore per definire correttamente i predecessori di c_1 e c_2. Ora, per trovare i sostituti dei due termini cancellati, si calcola esplicitamente il valore del predecessore di c_1.

$$\begin{aligned} P_c^-\ c_1 &= (\lambda n.n\ (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)\ B\ c_0 \\ &= c_1\ (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)\ B \\ &= (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)\ B \\ &= \lambda f.B(\ \lambda gx.f\ (g\ x))\ A) \end{aligned}$$

Questo deve essere uguale a c_0. Per fare ciò, si pone $B := \lambda abx.x$. Sostituendo, infatti, si ottiene proprio c_0.

$$\begin{aligned} \lambda f.B(\ \lambda gx.f\ (g\ x))\ A) = & \\ &= \lambda f.(\lambda abx.x)(\ \lambda gx.f\ (g\ x))\ A) \\ &= \lambda f.(\lambda x.x) \\ &= \lambda fx.x \\ &= c_0 \end{aligned}$$

Ora verrà sistemato A calcolando il predecessore di c_2:

$$\begin{aligned} P_c^- c_2 = & \\ &= c_2\ (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)\ (\lambda abx.x) \\ &= (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)((\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)(\lambda abx.x)) \\ &= (\lambda pf.p\ (\lambda gx.f\ (g\ x))\ A)\ c_0 \\ &= \lambda f.c_0(\lambda gx.f\ (g\ x))\ A \\ &= \lambda f.A \end{aligned}$$

Ricordando che questo termine deve essere uguale a c_1, e che c_1 è equivalente a $\mathbf{I}$, si deduce che il termine corretto da assegnare ad A è f. Abbiamo così ottenuto un predecessore analogo a quello di partenza, ma più efficiente.

$$\lambda n.n\ (\lambda pf.p\ (\lambda gx.f\ (g\ x)\ f))\ (\lambda abx.x)$$

4.4 Un nuovo successore

Definiamo un nuovo successore per i numerali di Church

$$\hat{S_c^+} = \lambda p.p\ S_c^+\ c_1.$$

Innanzitutto, è necessario dimostrare che questo è, effettivamente, un successore per la sequenza. Si ha che:

$$\widehat{S_c^+}\ c_i = c_i S_c^+ c_1 = c_{i+1} S_c^+ c_0 = c_{i+1}$$

Sia ora $p_0 = \lambda fx.c_0$ e $S_p^+ = \widehat{S_c^+}$. Il predecessore di c_n è il valore p_n stesso:

$$P_c^- = \lambda n.n S_p^+ p_0$$

La forma normale di questo predecessore, di lunghezza 25, è la seguente:

$$\lambda n.n\ (\lambda p.p\ (\lambda cfx.f(cfx))\ (\lambda fx.fx))\ (\lambda fxsz.z)$$

Infine, occorre dimostrare che questo è, effettivamente, un predecessore per i numerali di Church.

.

$$\begin{aligned} P_c^- c_{n+1} &= c_{n+1}\ \hat{S_c^+}\ c_1 \\ &= c_n\ \hat{S_c^+}\ ((\lambda p.p S_c^+ c_1)\ (\lambda fx.c_0)) \\ &= c_n\ \hat{S_c^+}\ c_0 \\ &= c_n\ S_c^+\ c_0 \\ &= c_n \end{aligned}$$

□

Dalla forma normale, è possibile ottimizzare questo predecessore usando la composizione al posto dell'applicazione di $\hat{S_c^+}$. Si ottiene

$$\lambda n.n\ (\lambda pf.p\ f \circ f)\ (\lambda fx.\mathbf{I})$$

che, in forma normale, ha lunghezza 21.

$$\lambda n.n\ (\lambda pf.p\ (\lambda gx.f\ (gx))f)\ (\lambda fxs.s)$$

4.5 *Unapplication*

È naturale pensare al predecessore come ad un'applicazione inversa: infatti, per passare da c_{n+1} a c_n, graficamente parlando, è sufficiente rimuovere l'applicazione più esterna dal termine.

Il lambda calcolo non ha un'operazione specifica per invertire un'applicazione: queste, infatti, non possono essere naturalmente tolte dall'espressione

di un termine. Tuttavia, il lambda calcolo può rappresentare tutte le possibili computazioni, compresa quella che si sta cercando.

È possibile codificare l'applicazione reiterata $f^n x$ (per qualche f e x). Si consideri una costruzione simile a quella presentata nella Sezione 4.2: `None` verrà utilizzato per rappresentare x e `Some` per l'applicazione della funzione f:

$$\texttt{None}_x := \lambda k.x \qquad \texttt{Some}_x := \lambda ak.ka$$

Le definizioni hanno come parametro x, il che le rende più semplici rispetto a quelle definite nella sezione precedente. Quindi, $f^n x$ è rappresentata da $\texttt{Some}_x^{(n)}\ \texttt{None}_x$, che verrà chiamato p_{nx}; si può notare la somiglianza a p_{nfx}, utilizzato nell'equazione 3, con $\texttt{Some}_x$ al posto di f. Inoltre, p_{nx} corrisponde esattamente a c_n.

Notiamo che:

$$\begin{aligned} p_{(n+1)x}\mathbf{I} = & \\ &= \texttt{Some}_x^{n+1}\ \texttt{None}_x\ \mathbf{I} \\ &= (\lambda ak.ka)(\texttt{Some}_x^n\ \texttt{None}_x)\mathbf{I} \\ &= \mathbf{I}(\texttt{Some}_x^n\ \texttt{None}_x) \\ &= p_{nx}. \end{aligned}$$

Dunque $p_{nx}\mathbf{I}$ si riduce a $p_{(n-1)x}$ quando $n \geq 0$. La codifica di p_{nx} in c_n, chiamata anche *reificazione* di c_n, e data da:

$$\texttt{reif}_x := \lambda n.n\ \texttt{Some}_x\ \texttt{None}_x$$

La decodifica, o *riflessione*, è data da

$$\texttt{refl}_f := \mathbf{Y}\lambda sp.p\ (\lambda q.f\ (s\ q))$$

che interpreta ricorsivamente p_{nx}, sostituendo $\texttt{None}_x$ e $\texttt{Some}_x$ rispettivamente con x e l'applicazione di f. Chiaramente vale che $c_n\ f\ x = \texttt{refl}_f(\texttt{reif}_x\ c_n)$ per ogni n.

Si nota che in $\texttt{refl}_f$ è presente l'operatore di punto fisso $\mathbf{Y}$, il che lo rende privo di forma normale. Per ovviare a questo problema - che renderebbe impossibile il confronto tra predecessori - si cerca di sostituirlo con altri termini. Dalla definizione di $\mathbf{Y}$ e c_n, si nota che:

$$\mathbf{Y}F = F(\mathbf{Y}F) \qquad c_{m+1}FG = F(c_{m+1}FG)$$

Si potrebbe dire, quindi, che c_{m+1} è un'approssimazione finita di $\mathbf{Y}$, valida al massimo per m ricorsioni. Siccome la ricorsione in $\texttt{refl}_f$ (che si può

schematizzare in $\mathbf{Y}AB$) termina sempre, deve per forza esistere un numero m tale per cui $\mathbf{Y}AB = (A^m D)B = c_m ADB$ per un termine arbitrario D. Possiamo introdurre, quindi

$$\texttt{refl'}_f := \lambda m.m(\lambda sp.p\ (\lambda q.f\ (s\ q)))D$$

il quale è tale per cui $c_n\ fx = \texttt{refl'}_f\ c_m\ (\texttt{reif}_x\ c_n)$ per ogni $m > n$. Per il predecessore, si può usare proprio c_n per approssimare $\mathbf{Y}$. Il termine D può essere scelto completamente a caso: influirà solo nel calcolo del predecessore di c_0, che non ha un significato ben definito.

Componendo, quindi, reificazione, predecessore di p_{nx} e riflessione si ottiene un nuovo predecessore per i numerali di Church:

$$\lambda nfx.\texttt{refl'}_f\ n\ ((\texttt{reif}_x n)\ \mathbf{I})$$

la cui forma normale ha lunghezza 31.

$$\lambda nfx.n\ (\lambda sp.p\ (\lambda q.f(sq)))\ n\ (n\ (\lambda ak.ka)\ (\lambda k.x)\ (\lambda z.z))$$

4.6 Isomorfismo

Avendo a disposizione diversi sistemi di numerazione, è possibile stabilire delle funzioni che trasformano un numero da una rappresentazione all'altra. In questa sezione verranno stabilite delle funzioni tra i numerali di Church e il sistema di numerazione presentato nella Sezione 3.1.

Si inizia costruendo un termine che permetta di passare dal numerale di Church c_n al numerale $\hat{n}$: sfruttando il termine S^+, si ottiene

$$\texttt{encode} := \lambda n.n\ S^+\ \hat{0}$$

Per la decodifica, è necessario procedere ricorsivamente *srotolando* le coppie innestate da cui è formato $\hat{n}$. Il passo base della ricorsione sarà definito dal test dello zero, definito sempre nella Sezione 3.1.

Lo pseudocodice della decodifica è il seguente:

$$\texttt{decode} := \lambda n.\ \texttt{if Zero}\ n\ \texttt{then}\ c_0\ \texttt{else}\ S_c^+\ (\texttt{decode}\ (n)_1)$$

È possibile rappresentare la condizione `if` nel seguente modo:

$$\texttt{decode} := \lambda n.\ (\texttt{Zero}\ n)\ c_0\ (S_c^+\ (\texttt{decode}\ (n)_1))$$

siccome il test *Zero* è esso stesso un valore booleano (definiti nella Sezione 2.2).

Per rappresentare la ricorsione, invece, si può usare il combinatore di punto fisso $\mathbf{Y}$.

$$\begin{aligned}
&\texttt{decode} := \lambda n.\ (\texttt{Zero}\ n)\ c_0\ (S_c^+\ (\texttt{decode}\ (n)_1))\\
&\texttt{decode} := (\lambda dn.\ (\texttt{Zero}\ n)\ c_0\ (S_c^+\ (d\ (n)_1)))\ \texttt{decode}\\
&\texttt{decode} := \mathbf{Y}(\lambda dn.\ (\texttt{Zero}\ n)\ c_0\ (S_c^+\ (d\ (n)_1)))
\end{aligned}$$

Come nella sezione precedente, l'utilizzo del combinatore $\mathbf{Y}$ toglie la possibilità di avere una forma normale. Anche in questo caso, però, si può usare l'approssimazione data dai numerali di Church: infatti, $\texttt{decode}\ \hat{n}$ termina sempre, quindi esiste un valore m tale per cui $\mathbf{Y}\texttt{decode}\ \hat{n} = (\texttt{decode}^m\ \hat{n}) = c_m\ \texttt{decode}\ \hat{n}$. Si ottiene quindi la seconda versione di `decode`

$$\texttt{decode'} := \lambda m.m\ (\lambda dn.\ (\texttt{Zero}\ n)\ c_0\ (S_c^+\ (d\ (n)_1)))$$

Questa, ancora una volta, è tale per cui $\texttt{decode}\ \hat{n} = \texttt{decode'}\ c_m\ \hat{n}$ per ogni $m > n$.

Rimane da definire il predecessore del sistema di numerazione $\hat{n}$:

$$P^- := \lambda n.(n)_1 = \lambda n.n\mathbf{F}.$$

Quindi, il predecessore per i numerali di Church che sfrutta l'isomorfismo è il seguente:

$$P_c^- := \lambda n.n\ \texttt{decode'}\ (P^-\ (\texttt{encode}\ n))$$

la cui forma normale è

$$\begin{aligned}
&\lambda n.n\ (\lambda m.m\ (\lambda dp.\ (\lambda x.x(\lambda ab.a))\ p)\ (\lambda xy.y)\ ((\lambda nfx.f(nfx))\\
&\qquad\qquad (d\ p(\lambda ab.b))))\ ((\lambda x.x(\lambda ab.b))\ (\lambda n.n\ (\lambda xp.p\ (\lambda ab.b)\ x)\ (\lambda z.z)).
\end{aligned}$$

La sua lunghezza è 56. Sicuramente, questo è il predecessore meno efficiente. È doveroso notare, però, che gran parte dei termini dipendono dal sistema di numerazione scelto: potrebbe essere interessante confrontare il risultato che si ottiene applicando il procedimento descritto a diversi sistemi di numerazione.

Questo approccio al problema è molto simile a quello illustrato nella Sezione 4.5. Il dover definire un isomorfismo tra c_n e $\hat{n}$ è tuttavia una richiesta non necessaria: le funzione $\texttt{reif}_x$ e $\texttt{refl}_f$ esprimono solo una parte dell'isomorfismo, e la loro composizione non è esattamente l'identità.

Riferimenti bibliografici

Kiselyov, O. (2020) *Many more predecessors: a representation workout.* Cambridge University Press

Barendregt, H. P. (1981) *The Lambda Calculus: Its Syntax and Semantics.* Amsterdam: Elsevier.