ESPRESSIVITÀ DEL LAMBDA CALCOLO TIPATO SEMPLICE

MICHAEL LODI
MATR. 607041

APPROFONDIMENTO PER L'ESAME DI
TIPI E LINGUAGGI DI PROGRAMMAZIONE

1. Introduzione

Il lambda calcolo puro, essendo Turing-completo, può esprimere tutte e sole le funzioni calcolabili. Dati due termini, stabilire se sono equivalenti è un problema indecidibile.

Passando però a considerare il lambda calcolo tipato semplice ($\lambda_\rightarrow$) la situazione cambia. In queste note, cercheremo di rispondere a due domande:

(1) Il problema di stabilire se due termini t_1 e t_2 di $\lambda_\rightarrow$ hanno la stessa forma normale, è decidibile?
   (a) Con che complessità?
(2) Quali funzioni è possibile esprimere con $\lambda_\rightarrow$?

La risposta al punto (1) deriva direttamente dal teorema di normalizzazione forte.

Teorema. *Il lambda calcolo tipato semplice gode della proprietà di normalizzazione forte: ogni termine è fortemente normalizzante (cioè, qualsiasi strategia di riduzione adottata, porta sempre a una forma normale in un numero finito di passi).*

Dunque, un algoritmo banale per rispondere alla domanda del punto (1) prevede semplicemente di ridurre entrambi i termini a forma normale e confrontarli. Il problema risulta dunque *decidibile*.

Inaspettatamente, la complessità di questo problema (non solo dell'algoritmo banale, ma di ogni altro algoritmo) è *enorme*. La studieremo estesamente a partire dalla sezione 2.

Dal teorema di normalizzazione sappiamo già inoltre che sicuramente *nessuna funzione propriamente parziale è definibile in* $\lambda_\rightarrow$. Cercheremo di analizzare più nel dettalgio la risposta al punto (2) a partire dalla sezione 7.

2. Il teorema di Statman

Prima di enunciare il teorema, vediamo alcune nozioni preliminari.

2.1. **Complessità elementare.** La classe di complessità $\mathbb{ELEMENTARY}$ è la classe delle funzioni *elementari ricorsive* o *Kalmár-elementari* (da ora: *elementari*). Tali funzioni sono definite in modo analogo alle funzioni *primitive ricorsive*, con la differenza che la ricorsione primitiva è sostituita dalla somma limitata e dal prodotto limitato.

Le funzioni *elementari* lavorano sui numeri naturali e sono definite a partire da funzioni base

- *funzione costante zero*: $o(x) = 0$
- *funzione successore*: $s(x) = x + 1$
- *proiezioni*: $p_i(x_1, ..., x_i, ..., x_n) = x_i$
- *sottrazione* positiva: $sub(x, y) = \begin{cases} x - y & se\ y < x \\ 0 & se\ y \geq x \end{cases}$

e altre funzioni costruite a partire da quelle base tramite

- *composizione*: $f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$ è elementare se h è elementare e ogni g_i è elementare.
- *somma limitata*: $bsum(m, x_1, \ldots, x_n) = \sum_{i=0}^{m} g(i, x_1, \ldots, x_n)$ è elementare se g è elementare
- *prodotto limitato*: $bprod(m, x_1, \ldots, x_n) = \prod_{i=0}^{m} g(i, x_1, \ldots, x_n)$ è elementare se g è elementare

Dai risultati di complessità sappiamo che

$$\mathbb{EXPTIME} \subsetneq \mathbb{ELEMENTARY} \subsetneq \mathbb{PR} \subsetneq \mathbb{R}$$

Possiamo inoltre caratterizzare la classe di complessità $\mathbb{ELEMENTARY}$ come la classe di problemi che possono essere decisi in tempo limitato da una torre di esponenti alta k (con k fissato) secondo la funzione $exp_k(n)$, dove n è la lunghezza dell'input.

Definizione 1. Definiamo la funzione exp come

$$\begin{aligned} \exp_0(n) &= n \\ \exp_{m+1}(n) &= 2^{\exp_m(n)} \end{aligned}$$

In generale quindi

$$\exp_m(n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}_{m}$$

Non abbiamo, come invece accade in $\mathbb{PR}$, la torre di esponenziali di altezza dipendente da n.

2.2. **Il teorema.** Il teorema è stato introdotto da Statman in [11], mentre Mairson ne ha dato una più semplice dimostrazione, che seguiremo qui, in [5].

Teorema 1 (Statman)**.** *Dati due termini tipizzabili in $\lambda_{\to}$, decidere se i due termini riducono alla stessa forma normale non è* elementare*: non può essere deciso in $\exp_k(n)$ passi, essendo k fissato e n la lughezza dei due termini.*

In altre parole, ***ogni procedura di decisione*** *richiede in generale più di $\exp_k(n)$ passi, per ogni k.*

Nelle prossime sezioni ci occuperemo di sviluppare gli strumenti e la dimistrazione di tale teorema.

3. Teoria dei Tipi

Consideriamo il linguaggio della teoria dei tipi Ω, che è un frammento della teoria dei tipi proposizionale di Henkin, che a sua volta è la restrizione della teoria dei tipi di Church a un solo tipo base.

Definizione 2 (Teoria Ω)**.** Il linguaggio della teoria dei tipi Ω è un linguaggio del prim'ordine, in cui ogni variabile ha un tipo costituito da un numero naturale (scritto come apice), e ci sono due costanti: true e false.

Le formule atomiche sono $\mathsf{true} \in x^1$, $\mathsf{false} \in x^1$, $x^n \in y^{n+1}$.

Tutte le altre forume sono costruite a partire dalla formule atomiche, con l'utilizzo dei connettivi usuali: $\wedge$, $\vee$, $\rightarrow$, $\neg$ e dei quantificatori $\forall$ ed $\exists$.

La semantica di Ω è data come segue. Sia $\mathcal{B}_0 = \{\mathsf{true}, \mathsf{false}\}$ e $\mathcal{B}_{k+1} := \mathcal{P}(\mathcal{B}_k)$. Le *variabili* x^k, y^k, z^k..., con $k \geq 0$, assumono valori negli elementi di $\mathcal{B}_k$, le *formule* $x^n \in y^{n+1}$ sono interpretate con la consueta nozione insiemistica di appartenenza, i *quantificatori* sono "di ordine superiore", nel senso che quantificano sugli elementi del tipo (insieme) della variabile a cui sono legati; infine i *connettivi* e le *costanti* hanno l'usuale interpretazione.

Fatto 1. *Vale che* $card(\mathcal{B}_i) = \exp_{i+1}(1)$

Dimostrazione. Per induzione.

Caso base. $card(\mathcal{B}_0) = \exp_1(1) = 2^{\exp_0(1)} = 2$

Induzione. Per IH sappiamo che $card(\mathcal{B}_i) = \exp_{i+1}(1)$. Proviamo che $card(\mathcal{B}_{i+1}) = \exp_{i+2}(1)$.

$$\begin{aligned} & card(\mathcal{B}_{i+1}) \\ \text{per def. di } \mathcal{B}_{i+1} \quad &= \quad card(\mathcal{P}(\mathcal{B}_i)) \\ \text{per definizione di } \mathcal{P}() \quad &= \quad 2^{card(\mathcal{B}_i)} \\ \text{per IH} \quad &= \quad 2^{\exp_{i+1}(1)} \\ \text{per definizione di exp} \quad &= \quad \exp_{i+2}(1) \end{aligned}$$

□

4. Outline della dimostrazione

Il problema di decidere se una formula ϕ di Ω è vera, richiede tempo non elementare. Questo è stato dimostrato in [7].

La dimostrazione del di Statman è una riduzione a questo problema: mostra come usare il lambda calcolo tipato semplice per simulare le formule di Ω e la procedura di eliminazione dei quantificatori per ridurre (logicamente e nel lambda calcolo) una formula ϕ a true o false.

Mairson, per mantere autocontenuta la dimostrazione, mostra come simulare una generica MdT non-elementare tramite Ω (e $\lambda_{\rightarrow}$).

Per avere un'intuizione, possiamo visualizzare la catena di riduzioni (dove $\preceq$ può essere letto come "*al più difficile quanto*"):

$$\text{MdT non elementare} \preceq \Omega \preceq \lambda_{\to}$$

5. Codificare la teoria dei tipi in $\lambda_{\to}$

5.1. **Liste.** Sia $\{t_1, t_2, ..., t_k\}$ un insieme di termini di $\lambda_{\to}$, ognuno con tipo fissato α. Possiamo definire una lista di tali termini come un termine che avrà tipo

$$\mathrm{L} : (\alpha \to \tau \to \tau) \to \tau \to \tau$$

in cui τ è un *qualsiasi* tipo di $\lambda_{\to}$. Per semplificare la notazione diremo che $L : List(\alpha)$. Il termine L sarà così definito:

$$\mathrm{L} := \lambda cons : (\alpha \to \tau \to \tau).\lambda nil : \tau.cons\ t_1\ (cons\ t_2 \dots (cons\ t_k\ nil)...)$$

Esempio. A partire da questo temrine, possiamo ad esempio calcolare la lunghezza di una lista, tramite la funzione

$$\textsc{length} := \lambda L : List(\alpha)_{\mathsf{Int}}.L\ (\lambda x : \alpha.\textsc{succ})\ 0$$

in cui 0 e SUCC sono costruiti come di consueto con i numerali di Church, $\mathsf{Int} := (\nu \to \nu) \to \nu \to \nu$ e τ in $List(\alpha)$ è istanziato con Int.

Possiamo facilmente convincerci che la funzione calcoli la lunghezza: essa "butta via" gli elementi della lista e li conta tramite SUCC.

5.2. **Booleani.** Sia ν un qualsiasi tipo di $\lambda_{\to}$. Definiamo $\mathsf{Bool} := \nu \to \nu \to \nu$. I valori booleani e i classici connettivi sono costruiti nel modo usuale:

$$\begin{aligned}
\textsc{true} &:= \lambda x : \nu.\lambda y : \nu.x : \mathsf{Bool} \\
\textsc{false} &:= \lambda x : \nu.\lambda y : \nu.y : \mathsf{Bool} \\
\textsc{and} &:= \lambda p : \mathsf{Bool}.\lambda q : \mathsf{Bool}.\lambda x : \nu.\lambda y : \nu.p\ (q\ x\ y)\ y : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} \\
\textsc{or} &:= \lambda p : \mathsf{Bool}.\lambda q : \mathsf{Bool}.\lambda x : \nu.\lambda y : \nu.p\ x\ (q\ x\ y) : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} \\
\textsc{not} &:= \lambda p : \mathsf{Bool}.\lambda x : \nu\lambda y : \nu.p\ y\ x : \mathsf{Bool} \to \mathsf{Bool} \\
\textsc{impl} &:= \lambda a : \mathsf{Bool}.\lambda b : \mathsf{Bool}.\textsc{or}\ (\textsc{not}\ a)\ b : \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool}
\end{aligned}$$

5.3. **Codifica dei tipi.** L'insieme $\mathcal{B}_0$ può essere rappresentato come la lista $\mathbf{D}_0$ contenente TRUE e FALSE:

$$\mathbf{D}_0 := \lambda c : \mathsf{Bool} \to \tau \to \tau.\lambda n : \tau.c\ \textsc{true}\ (c\ \textsc{false}\ n) : List(\mathsf{Bool})$$

A questo punto, per ogni intero $k > 0$, definiamo un termine $\mathbf{D}_k$ di lunghezza $\Theta(k)$ che rappresenta $\mathcal{B}_k$ come la lista di tutti i sottoinsiemi (o meglio, ricorsivamente, le loro codifiche) di elementi di tipo $\mathcal{B}_{k-1}$. Per fare questo, abbiamo bisogno di definire un costruttore di *powerset*.

Per prima cosa, definiamo un termine INJECT che, dato un elemento $x : \alpha$ e una lista $l : List(List(\alpha))$, appende l a una lista costruita aggiungendo (in testa) x ad ogni elemento di l.
Ad esempio (con $\alpha = \mathsf{Bool}$ e una notazione succinta per le liste) INJECT FALSE $[[], [\textsc{true}]]$ riduce

a [[FALSE], [FALSE, TRUE], [], [TRUE]]. Possiamo definire il termine INJECT : $\alpha \to List(List(\alpha)) \to List(List(\alpha))$ nel modo seguente

$$\text{INJECT} := \lambda x : \alpha.\lambda l : List(List(\alpha)).\lambda c : List(\alpha) \to \tau \to \tau.\lambda n : \tau. \\ l\ (\lambda e : List(\alpha).c\ (\lambda c' : \alpha \to \tau \to \tau.\lambda n' : \tau.c'\ x\ (e\ c'\ n')))\ (l\ c\ n)$$

Intuitivamente, si itera sulla lista una funzione che, preso ogni elemento e (che è una lista a sua volta) ne costruisce uno nuovo con x in testa.

Possiamo a questo punto definire il termine POWERSET : $List(\alpha)_{List(List(\alpha))} \to List(List(\alpha))$ come segue:

$$\text{POWERSET} := \lambda S : List(\alpha)_{List(List(\alpha))}. \\ S\ \text{INJECT}\ (\lambda c : List(\alpha) \to \tau \to \tau.\lambda n : \tau.c\ (\lambda c' : \alpha \to \tau \to \tau.\lambda n' : \tau.n')\ n)$$

Per abbreviare la notazione, definiamo

$$\begin{aligned} \Delta_0 &:= \mathsf{Bool} \\ \Delta_{k+1} &:= List(\Delta_k) \end{aligned}$$

e dunque, per $k \geq 0$,

$$\mathbf{D}_k : \Delta_{k+1}$$

In generale potrà essere definito

$$\mathbf{D}_{k+1} := \text{POWERSET}\ \mathbf{D}_k$$

In particolare notiamo

$$\mathbf{D}_0 := [\text{TRUE}, \text{FALSE}] : \Delta_1 = List(\Delta_0) = List(\mathsf{Bool})$$

e ricorsivamente

$$\begin{aligned} \mathbf{D}_1 := \text{POWERSET}\ \mathbf{D}_0 &: \Delta_2 = List(\Delta_1) \\ &\dots \\ \mathbf{D}_{k+1} := \text{POWERSET}\ \mathbf{D}_k &: \Delta_{k+2} = List(\Delta_{k+1}) \end{aligned}$$

Per la corretta tipizzazione, nella definizione di $\mathbf{D}_{k+1}$, l'occorrenza più a sinistra di POWERSET verrà istanziata con il tipo

$$List(\Delta_k)_{\Delta_{k+2}} \to \Delta_{k+2}$$

Osservazione 1. Se $erasing(T)$ cancella le informazioni di tipo dal termine T, allora avremo che

$$|erasing(\mathbf{D}_k)| = \Theta(k)$$

Infatti è facile vedere come la lunghezza di termine sia lineare in k, visto che verranno ripetuto k volte il termine POWERSET. Ma ovviamente la lunghezza della loro forma normale, per definizione di $\mathcal{P}()$, sarà almeno quanto una pila di esponenti, ovvero $\Omega(\exp_{k+1}(1))$.

5.4. **Codifica degli insiemi.** Esiste un concetto "naturale" di uguaglianza tra elementi di $\mathcal{B}_k$. Quando poi gli elementi sono insiemi, possiamo ovviamente considerare anche i concetti naturali di sottoinsieme e di elemento di un insieme.

Possiamo quindi realizzare le formule atomiche della teoria dei tipi usando questi concetti e sfruttando l'idea di iterazione su liste. Per prima cosa definiamo

$$\text{EQ}_0 := \lambda x^0 : \mathsf{Bool}.\lambda y^0 : \mathsf{Bool}.\text{OR } (\text{AND } x^0\ y^0)\ (\text{AND } (\text{NOT } x^0)\ (\text{NOT } y^0))$$

che ci dà l'uguaglianza tra due termini di tipo Δ_0 (Bool).

A questo punto procediamo con tre definizioni ricorsive, nelle quali useremo i termini di tipo Δ_j per iterare una funzione booleana. Per fare ciò, in tali termini, istanziamo $\tau := \mathsf{Bool}$. Definiamo quindi

- un costrutto per l'appartenenza di un elemento a un insieme

$$\begin{aligned}\text{MEMBER}_{k+1} := \lambda x^k : \Delta_k.\lambda y^{k+1} : \Delta_{k+1}.\\ y^{k+1}\ (\lambda e^k : \Delta_k.\text{OR } (\text{EQ}_k\ x^k\ e^k)\ \text{FALSE}\\ : \Delta_k \to \Delta_{k+1} \to \Delta_0\end{aligned}$$

- un costrutto per la relazione di sottoinsieme

$$\begin{aligned}\text{SUBSET}_{k+1} := \lambda x^{k+1} : \Delta_{k+1}.\lambda y^{k+1} : \Delta_{k+1}.\\ x^{k+1}\ (\lambda e^k : \Delta_k.\text{AND } (\text{MEMBER}_{k+1}\ e^k\ y^{k+1})\ \text{TRUE}\\ : \Delta_{k+1} \to \Delta_{k+1} \to \Delta_0\end{aligned}$$

- un costrutto per l'uguaglianza tra due elementi di k-esimo tipo

$$\begin{aligned}\text{EQ}_{k+1} := \lambda x^{k+1} : \Delta_{k+1}.\lambda y^{k+1} : \Delta_{k+1}.\\ (\lambda op : \Delta_{k+1} \to \Delta_{k+1} \to \Delta_0.\text{AND } (op\ x^{k+1}\ y^{k+1})\ (op\ y^{k+1}\ x^{k+1}))\ \text{SUBSET}_{k+1}\\ : \Delta_{k+1} \to \Delta_{k+1} \to \Delta_0\end{aligned}$$

I termini MEMBER_k, SUBSET_k, EQ_k, dopo che sono state cancellate le informazioni di tipo, hanno tutti lunghezza $\Theta(k)$.

Questo può essere facilmente calcolato con una equazione di ricorrenza lineare di ordine costante. Infatti sostanzialmente il termine EQ_k effettua (indirettamente) una chiamata ricorsiva al termine EQ_{k-1}. Se ne effettuasse due (cioè se non si usasse lo stratagemma di *op*), avremmo una lunghezza esponenziale.

Più formalmente:

Teorema. *Sia $S(n)$ la relazione di ricorrenza definita come*

$$\begin{cases} T(n) = costante & n \leq m \leq h \\ T(n) = \sum_{i=1}^{h} a_i T(n-i) + cn^\beta & n > m \end{cases}$$

allora, ponendo $a = \sum_{i=1}^{h} a_i$ avremo che

$$T(n) \grave{e}\, O(n^{\beta+1}) \quad se \quad a = 1$$
$$T(n) \grave{e}\, O(a^n n^\beta) \quad se \quad a \geq 2$$

Se misuriamo la lunghezza di EQ_k avremo

$$\begin{cases} L(k) = O(1) & se\, k = 0 \\ L(k) = 1S(k-1) + ck^0 & altrimenti \end{cases}$$

e dunque $L(k) = O(k^1)$. Se avessimo due chiamate ricorsive, diventerebbe $a = 2$ e dunque avremmo una lunghezza $O(2^k)$.

5.5. **Interpretazione.** Con le definizioni precedenti, possiamo interpretare tutte le formune logiche della teoria dei tipi con il lambda calcolo tipato semplice.

L'insieme $\mathcal{B}_k$ verrà codificato con il termine $\mathbf{D}_k$, di tipo $\Delta_{k+1} = List(\Delta_k)$, cioè una lista di tutti gli elementi che gli appartengono (ognuno dei quali avrà una codifica di tipo Δ_k).

Le costanti true e false sono interpretate con i termini TRUE e FALSE, di tipo Bool.

Un elemento (insieme) x^k sarà interpretato da un termine di tipo Δ_k, ovvero una lista di elementi di tipo Δ_{k-1}, cioè proprio gli elementi che appartengono a tale insieme.

La formula primitiva $x^k \in y^{k+1}$ è interpretata dal termine $\text{MEMBER}_{k+1}\ x^k\ y^{k+1}$, che avrà tipo Bool e ridurrà a TRUE o FALSE se x^k e y^{k+1} sono termini chiusi.

I connettivi sono codificati tramite le funzioni booleane viste, tutte con risultato di tipo Bool.

I quantificatori e la loro eliminazione richiedono un'analisi più accurata. Supponiamo di avere codificato una funzione booleana ψ della teoria dei tipi con un termine $\hat{\psi}$ di tipo $\Delta_k \to \text{Bool}$. Allora, decidere la verità della formula

$$\forall x^k.\psi(x^k)$$

può essere espresso come la riduzione del termine

$$\mathbf{D}_k\ (\lambda x^k : \Delta_k.\text{AND}\ (\hat{\psi}\ x^k))\ \text{TRUE}$$

e allo stesso modo decidere la verità della formula

$$\exists x^k.\psi(x^k)$$

può essere espresso come la riduzione del termine

$$\mathbf{D}_k\ (\lambda x^k : \Delta_k.\text{OR}\ (\hat{\psi}\ x^k))\ \text{FALSE}$$

Il funzionamento di questi termini è intuitivo e sensato, dato che si itera su insiemi finiti.

Da quanto detto, si dimostra il seguente:

Teorema 2. *Una generica formula ϕ della teoria dei tipi Ω è vera* se e solo se *la sua interpretazione $\hat{\phi} : \mathsf{Bool}$ in $\lambda_\to$ è $\beta\eta$-equivalente a* TRUE : Bool.

6. Simulare una Macchina di Turing non-elementare

Mostriamo ora come usare la teoria dei tipi (o, equivalentemente, il lambda calcolo tipato semplice) possa essere usata per simulare una Macchina di Turing per tempo non-elementare.

Assumiamo che la nostra MdT abbia come alfabeto $\{0, 1\}$, così che il contenuto del nastro sia un (lungo) numero binario. Visto che $|\mathcal{B}_{k+1}| = 2^{|\mathcal{B}_k|}$, è facile mostrare che ogni elmento $x^{k+1} \in \mathcal{B}_{k+1}$ può essere visto come tale numero binario : gli elementi di x^{k+1} (che sono al più $|\mathcal{B}_k|$) rappresentano le posizioni dei bit settati a 1.

Esempio 1. Consideriamo $\mathcal{B}_1$. Sappiamo che $|\mathcal{B}_1| = 2^{|\mathcal{B}_0|} = 4$. Dunque con un elemento di $\mathcal{B}_1$ possiamo rappresentare un nastro lungo 2. Se stabiliamo che $\mathsf{false} \in \mathcal{B}_0$ rappresenti il bit meno significativo e true quello più significativo, allora avremo una corrispondenza con gli elementi di $\mathcal{B}_1$:

$\emptyset$	00
$\{\mathsf{false}\}$	01
$\{\mathsf{true}\}$	10
$\{\mathsf{true}, \mathsf{false}\}$	11

Possiamo definire un ordine totale ($<_{k+1}$) e un test ($\mathbf{succ}_{k+1}$) per elementi x^{k+1}.

In particolare scriveremo

$$\begin{aligned} x^0 <_0 y^0 &:= \neg x^0 \wedge y^0 \\ x^{k+1} <_{k+1} y^{k+1} &:= \exists z^k . z^k \in y^{k+1} \wedge z^k \notin x^{k+1} \\ &\quad \wedge \forall w^k . z^k <_k w^k \to (w^k \in x^{k+1} \leftrightarrow w^k \in y^{k+1}) \end{aligned}$$

Infatti $x < y$ se esiste un bit (z) che vale 1 in y e 0 in x, e i bit più significativi di z sono identici.

Il test per il successore può essere definito come

$$\begin{aligned} \mathbf{succ}_{k+1}(x^{k+1}, y^{k+1}) &:= \exists z^k . z^k \in y^{k+1} \wedge z^k \notin x^{k+1} \\ &\quad \wedge \forall w^k . z^k <_k w^k \to (w^k \in x^{k+1} \leftrightarrow w^k \in y^{k+1}) \\ &\quad \wedge \forall w^k . w^k <_k z^k \to (w^k \in x^{k+1} \wedge w^k \notin y^{k+1}) \end{aligned}$$

Infatti $y = x + 1$ se esiste un bit (z) che vale 1 in y e 0 in x, i bit più significativi di z sono identici in entrambi i numeri, mentre i bit meno significativi sono 1 in x e 0 in y.

Dunque, un elemento $x^{n+1} \in \mathcal{B}_{n+1}$ può codificare il contenuto di un nastro lungo $|\mathcal{B}_n|$.

Sia poi y^{n+1}; se $|y^{n+1}| = 1$, allora tale variabile può codificare la posizione corrente della testina.

Una coppia $\langle x^{n+1}, y^{n+1} \rangle$ può essere rappresentata nella teoria degli insiemi (con una definizione sostanzialmente analoga a quella di Wiener) come $\{\{\{\}, x^{n+1}\}, \{y^{n+1}\}\} \in \mathcal{B}_{n+3}$.

Se, come di solito si fa, codifichiamo anche lo stato direttamente nella stringa x^{n+1}, una configurazione della MdT (che chiameremo ID) può essere rappresentata con una coppia, tramite un elemento di $\mathcal{B}_{n+3}$.

Usando la teoria dei tipi e il lambda calcolo, è possibile codificare[1] una *relazione* binaria $\rho \subseteq \mathcal{B}_{n+3} \times \mathcal{B}_{n+3}$, dove $\rho(ID, ID')$ signfica che ID' è raggiundibile da ID in un passo di transizione. In particolare, avremo il termine

$$\hat{\rho} : \Delta_{n+3} \to \Delta_{n+3} \to \mathsf{Bool}[\nu := \Delta_{n+3}]$$

in cui abbiamo istanziato Bool come $\Delta_{n+3} \to \Delta_{n+3} \to \Delta_{n+3}$, di modo da poter iterare sugli ID.

Possiamo a questo punto definire la *funzione* di transizione $\delta : \Delta_{n+3} \to \Delta_{n+3}$ come

$$\delta := \lambda ID : \Delta_{n+3}.\mathbf{D}_{n+3}\ (\lambda ID' : \Delta_{n+3}.\lambda ID'' : \Delta_{n+3}.(\hat{\rho}\ ID\ ID')\ ID'\ ID'')\ \textsc{emptyset}_{n+3}$$

Intuitivamente, la funzione restuisce l'ID' più a sinistra, nella lista di tutti i possibili ID (cioè $\mathbf{D}_{n+3}$), che soddisfa la relazione. Se non lo trova, restiuisce un termine convenzionale (in questo caso l'insieme vuoto).

A questo punto, ci rimane solo da iterare la funzione δ. Useremo i numerali di Church, con i tipi istanziati opportunamente

$$\overline{2} := \lambda f.\lambda x.f\ (f\ x)$$

Possiamo quindi definire il termine (in cui x è ridenominato ID)

$$\mathrm{C} := \overline{2}\ \overline{2}\ ...\ \overline{2}\ \delta \longrightarrow^*_\beta \lambda ID.\delta(\delta(...(\delta\ ID)...)) : \Delta_{n+3} \to \Delta_{n+3}$$

Notiamo che il $\overline{2}$ più a destra deve avere tipo $(\Delta_{n+3} \to \Delta_{n+3}) \to \Delta_{n+3} \to \Delta_{n+3}$. Pertanto il suo predecessore dovrà essere una funzione da quel tipo a quel tipo. In generale, se il j-esimo $\overline{2}$ ha tipo κ, il suo predecessore avrà tipo $\kappa \to \kappa$. Dunque, se ci sono n occorrenze di $\overline{2}$, allora verrano effettuate $\exp_n(1)$ applicazioni di δ, ovvero un numero "non-elementare".

Basterà infatti applicare C a un altro termine che codifica un ID iniziale, estrarre dal termine risultate lo stato finale e controllare che sia uno stato di accettazione, avendo una risposta di tipo Bool.

7. Potere Espressivo: nozioni preliminari

In questa seconda parte proviamo ad valutare il potere espressivo del lambda calcolo tipato semplice

Nel lambda calcolo puro non ci sono problemi a rappresentare gli interi di Church. In $\lambda_\to$, invece, la composizione di funzioni può essere applicata solo ad argomenti di un tipo specifico.

Precisiamo per prima cosa le nozioni di tipi in $\lambda_\to$:

Definizione 3 (Tipi base e tipi semplici in $\lambda_\to$)**.** Fissiamo un insieme di *tipi base.* Per i nostri scopi useremo i **tipi base** o e B_0.

Allora, i *tipi semplici* sono definiti induttivamente:

[1]per dettagli sulla codifica e sulla sua analisi quantitativa si veda [**?**]

- Un tipo base è un tipo semplice;

- Se τ e ν sono tipi semplici, allora $\tau \to \nu$ è un tipo semplice.

Ci sono due possibilità per rappresentare funzioni numeriche in $\lambda_\to$:

(1) scegliere un particolare tipo e richiedere che tutti i numeri naturali siano rappresentati da composizioni di funzioni di quel tipo
(2) permettere rappresentazioni diverse dei naturali che possono mischiarsi in un'espressione

Definizione 4 (Rappresentazione sul tipo τ)**.** Sia τ un tipo semplice. La rappresentazione sul dominio τ del numero naturale n è il termine

$$\overline{n} := \lambda f : \tau \to \tau . \lambda x : \tau . \underbrace{f(\ ... \ (f}_{n \ volte} x)...)$$

Il suo tipo sarà $(\tau \to \tau) \to \tau \to \tau$.

Analizziamo per il momento il **caso 1**.

In quanto segue, sarà centrale la nozione di *polinomi estesi*.

Definizione 5 (Polinomi estesi)**.** I polinomi estesi sono la più piccola classe di funzioni sui naturali, che contiene

- le *costanti* 0 e 1
- le *proiezioni*
- l'*addizione*
- la *moltiplicazione*
- la funzione condizionale $cond(n_1, n_2, n_3) = \begin{cases} n_2 & se \ n_1 = 0 \\ n_3 & altrimenti \end{cases}$

ed è chiusa per composizione.

8. Il teorema di Schwichtenberg

Per valutare il potere espressivo del lambda calcolo tipato semplice, dobbiamo prima di tutto dare una definizione precisa *funzioni definibili*.

Definizione 6 (Funzioni strettamente definibili)**.** Sia $\mathsf{Int} := (o \to o) \to o \to o$, dove o è un *tipo base*. Una funzione (totale) $f : \mathbb{N}^k \to \mathbb{N}$ è strettamente definita da un termine F ben tipizzato di $\lambda_\to$ se e solo se

(1) $f(n_1, ..., n_k) = m \iff \mathrm{F} \ \overline{n_1} \ ... \ \overline{n_k} \longrightarrow_\beta^* \overline{m}$
(2) $\mathrm{F} : \underbrace{\mathsf{Int} \to \mathsf{Int} \to ... \to \mathsf{Int}}_{k \ volte} \to \mathsf{Int}$

Teorema 3 (Schwichtenberg)**.** *Le funzioni strettamente definibili sono esattamente i polinomi estesi.*

Dimostrazione. Procederemo in tre fasi.

1 - Dimostriamo che *tutti i polinomi estesi sono strettamente definibili.* Per farlo, ci basta fornire tre termini di $\lambda_{\to}$ per addizione, moltiplicazione e condizionale (costanti e proiezioni sono banali). Avremo:

$$\begin{aligned} \text{ADD} &:= \lambda p : \mathsf{Int}.\lambda q : \mathsf{Int}.\lambda f : o \to o.\lambda x : o.(p\ f)\ (q\ f\ x) \\ \text{MULT} &:= \lambda p : \mathsf{Int}.\lambda q : \mathsf{Int}.\lambda f : o \to o.\lambda x : o.q\ (p\ f)\ x \\ \text{COND} &:= \lambda p : \mathsf{Int}.\lambda q : \mathsf{Int}.\lambda r : \mathsf{Int}.\lambda f : o \to o.\lambda x : o.p\ (\lambda y : o.r\ f\ x)\ (q\ f\ x) \end{aligned}$$

Dunque, dato il polinomio $P(x, y)$[2], sappiamo che esiste un termine di $\lambda_{\to}$ che lo definisce.

2 - Sia $\Gamma = \{f : o \to o, a : \mathsf{Int}, b : \mathsf{Int}, x_1 : o, ..., x_r : o\}$ e sia $\Gamma \vdash M : o$. Allora, dimostriamo che esiste un polinomio $P(m, n)$ e un numero $1 \leq i \leq r$ (con r arbitrario) tale che $M\,[\overline{\mathrm{m}}/\mathrm{a}, \overline{\mathrm{n}}/\mathrm{b}] =_\beta f^{P(m,n)}\, x_i$ [3]per ogni $m, n \neq 0$.

Per dimostrarlo, procediamo per induzione sulla struttura di M.

Se M è una variabile x_i, poniamo $P(m, n) = 0$.

Altrimenti, M è sicuramente un'applicazione e, poiché ha tipo o, sarà sicuramente in una delle forme

$$\begin{aligned} M &= f\ M_1 \\ M &= a\ N\ M_1 \\ M &= b\ N\ M_1 \end{aligned}$$

in cui $\Gamma \vdash N : o \to o$ e $\Gamma \vdash M_1 : o$. Per IH su M_1 avremo i e $P_1(m, n)$.
Se $M = f\ M_1$ allora, per costruzione sugli interi di Church, $P(m, n) = P_1(m, n) + 1$.
Se $M = a\ N\ M_1$ (il caso di b è analogo), possiamo assumere senza perdere in generalità che valga $N = \lambda x_{r+1}.M_2$, in cui $\Gamma, x_{r+1} : o \vdash M_2 : o$. (Se questo non dovesse valere, potremmo comunque usare la η conversione su N , ed M non cambia a meno di β-equivalenza).
Per IH sappiamo che esistono P_2 e $j \leq r + 1$ tali che $M_2\,[\overline{\mathrm{m}}/\mathrm{a}, \overline{\mathrm{n}}/\mathrm{b}] =_\beta f^{P_2(m,n)}\, x_j$. Allora abbiamo immediatamente che

$$M\,[\overline{\mathrm{m}}/\mathrm{a}, \overline{\mathrm{n}}/\mathrm{b}] =_\beta \overline{m}\ (\lambda x_{r+1}.f^{P_2(m,n)}\, x_j)\ (f^{P_1(m,n)}\, x_i)$$

Se $j \leq r$ allora si vede che il termine riduce a $f^{P_2(m,n)}\, x_j$, e dunque abbiamo trovato un polinomio.
Se $j = r + 1$ allora si può vedere come la funzione applica m volte il polinomio P_2 a partire da P_1, e dunque avremo $f^{P(m,n)}x_i$, in cui $P(m, n) = m \cdot P_2(m, n) + P_1(m, n)$.

3 - Dimostriamo infine che *tutte le funzioni definibili in $\lambda_{\to}$ sono polinomi estesi.* Supponiamo che una funzione g sia definibile da un termine G, in forma normale. Assumiamo per semplicità che g abbia due argomenti. La funzione dovrà prendere in input due interi e restituire un intero, dunque

[2]per semplicità ci limitiamo a polinomi in due variabili
[3]$f^{P(m,n)} = \underbrace{f(f...f}_{P(m,n)\ volte}(x_i)..)$

senza perdere in generalità possiamo riscrivere il termine G come

$$\lambda a : \mathsf{Int}.\lambda b : \mathsf{Int}.\lambda f : o \to o.\lambda x_1 : o.M$$

Dal punto precedente, sappiamo che esiste un polinomio tale che $g(m,n) = P(m,n)$, per ogni $m, n \neq 0$. Con ragionamenti analoghi, ci si può convincere che esistono monomi Q e R tali che $g(m,0) = Q(m)$ e $g(0,n) = R(n)$ per ogni m, n. Calcoliamo infine $g(0,0) = k$. Allora possiamo esprimere g come un polinomio esteso:

$$g(m,n) = cond(m,\ cond(n,\ k,\ R(n)),\ cond(n,\ Q(m),\ P(m,n)))$$

□

9. Rappresentare i Naturali su più domini

Come abbiamo visto, con gli interi rappresentati in modo fissato, la classe delle funzioni rappresentabili è relativamente piccola. Proviamo dunque ad aumentare questa classe, rilassando i vincoli precedenti. Iniziamo con il permettere la *rappresentazione dei numeri naturali su domini diversi.*

Sia B_0 un tipo base. Poniamo $B_{i+1} := B_i \to B_i$. A questo punto possiamo definire la rappresentazione dei naturali su B_i come

$$\mathsf{Int}_i := (B_i \to B_i) \to B_i \to B_i$$

da cui segue

$$\mathsf{Int}_{i+1} \equiv \mathsf{Int}_i \to \mathsf{Int}_i$$

Con questa rappresentazione è facile definire l'esponenziale come

$$\textsc{exp}_i := \lambda m : \mathsf{Int}_{i+1}.\lambda n : \mathsf{Int}_i.m\ n$$

infatti $\textsc{exp}_i\ \overline{m}\ \overline{n}$ riduce alla rappresentazione di n^m di tipo Int_i.

Dando le seguenti definizioni di coppia e proiezioni sulle coppie e istanziandole opportunamente a seconda dei tipi necessari

$$\begin{aligned} \langle m,n\rangle &:= \lambda s.\textsc{cond}\ s\ m\ n \\ \textsc{first} &:= \lambda p.p\ \overline{0} \\ \textsc{second} &:= \lambda p.p\ \overline{1} \end{aligned}$$

è facile definire il predecessore $\textsc{pred} : \mathsf{Int}_{i+3} \to \mathsf{Int}_i$ come

$$\textsc{pred} := \lambda n : \mathsf{Int}_{i+3}.(\textsc{second}\ (n\ \lambda z : \mathsf{Int}_{i+1}.\langle \textsc{add}_i\ \overline{1}\ (\textsc{first}\ z),\ (\textsc{first}\ z)\rangle\ \langle\overline{0},\overline{0}\rangle))$$

in cui $\overline{0}, \overline{1} : \mathsf{Int}_i$ e le coppie hanno tipo Int_{i+1}.

Possiamo infine scrivere inoltre una funzione che riduce il dominio di una rappresentazione:

$$\mathrm{R}_i := \lambda n : \mathsf{Int}_{i+1}\lambda f : B_i \to B_i.(n\ \lambda h : B_i \to B_i.\lambda x : B_i.f\ (h\ x))\ \lambda x : B_i.x)$$

Infatti $\mathrm{R}_i : \mathsf{Int}_{i+1} \to \mathsf{Int}_i$.

Da quanto visto in precedenza, possiamo enunciare il seguente

Teorema 4. *Tutte le funzioni generate dalla costanti 0 e 1, dall'addzione, moltiplicazione, condizionale, esponenziazione e predecessore possono essere rappresentate da un termine di $\lambda_{\to}$, usando rappresentazioni di interi su domini diversi.*

10. Spingerci al limite

Nell'articolo [3], tramite un upper bound alla dimensione della forma normale di un termine, vengono dimostrati alcuni importanti teoremi. Per analizzarli, vediamo prima alcune importanti definizioni.

Definizione 7 (Altezza di un termine)**.** Definiamo l'altezza di un termine E, $hgt(E)$ come

$$\begin{aligned} hgt(x) &= 0 \\ hgt(\lambda x.E) &= 1 + hgt(E) \\ hgt(E\ F) &= 1 + \max\{hgt(E), hgt(F)\} \end{aligned}$$

Definizione 8 (Livello)**.** Il livello di un tipo τ è definito come

$$\begin{aligned} lvl(\tau) &= 0 \text{ se } \tau \text{ è un tipo base} \\ lvl(\tau \to \nu) &= 1 + \max\{lvl(\tau), lvl(\nu)\} \end{aligned}$$

Il livello di un redex è definito come il livello della parte funzionale:

$$lvl(\lambda x : \tau.P : \tau \to \nu) = lvl(\tau \to \nu)$$

Definizione 9 (Grado di un termine)**.** Il grado di un termine E, $deg(E)$ è definito come il massimo livello dei redex in E, ovvero

$$deg(E) = \max\{lvl(\mathscr{R}) | \mathscr{R} \text{ è un redex in } E\}$$

Lemma 1. *Un termine M tale che $deg(M) = d$ e $hgt(M) = h$ riduce, in al più 2^h passi, ad un termine M_1 tale che $deg(M_1) < d$ e $hgt(M_1) \leq 2^h$.*

Dimostrazione. Un termine di altezza h ha meno di 2^h redex di livello massimo. Per cui, si impiegano al più 2^h passi per ridurli tutti, se scegliamo la strategia *rightmost-innermost.*

Dimostriamo ora la proprietà per induzione su $hgt(M)$.

Caso base. Banale, in quanto non posso ridurre.

Caso $M = \lambda x.F$. Per definizione, $deg(F) = d$ e $hgt(F) = h - 1$. Per cui, per IH, F riduce a F_1 tale che $deg(F_1) < d$ e $hgt(F_1) \leq 2^{h-1}$. Per cui $M_1 = \lambda x.F_1$ avrà $deg(M_1) < d$ e $hgt(M_1) \leq 2^{h-1} + 1 \leq 2^h$.

Caso $M = N\,P$. Seguendo la nostra strategia di riduzione, riduciamo prima a $N\,P_1$ e poi a $N_1\,P_1$.

Per definizione $deg(N) \leq d$ e $deg(P) \leq d$. Inoltre $hgt(N), hgt(P) \leq h-1$. Per IH $deg(N_1), deg(P_1) < d$ e $hgt(N_1), hgt(P_1) \leq 2^{h-1}$.

Per $M_1 = P_1 N_1$ varrà che $hgt(M_1) \leq 2^{h-1} + 1 \leq 2^h$. Se poi $N_1 P_1$ non è un redex, o è un redex con $deg(M_1) < d$, allora abbiamo la tesi.

Se invece è un redex di grado d, allora lo eseguiamo e l'altezza sarà comunque $hgt(M_1) \leq 2^{h-1} + 2^{h-1} \leq 2^h$, mentre il grado dei redex residui e dei redex creati sarà comunque più basso, per cui $deg(M_1) < d$. □

Teorema 5. *Un termine di grado d e altezza h ha una forma normale di altezza al più $\exp_d(h)$ e può essere ottenuta in al più $\exp_{d+1}(h)$ passi di riduzione.*

Dimostrazione. Usando il Lemma 1. Per avere un'intuizione, usiamo la seguente tabella:

		grado	**altezza**	**#redex/#passi**
0		d	h	2^h
1		$d-1$	$2^h = \exp_1(h)$	$2^{2^h} = \exp_2(h)$
⋮		⋮	⋮	⋮
⋮		⋮	⋮	⋮
d		0	$\exp_d(h)$	$\exp_{d+1}(h)$

□

Teorema 6. *Se una funzione $f : \mathbb{N} \to \mathbb{N}$ è rappresentabile in $\lambda_\to$, allora f è limitata da una funzione elementare.*

Dimostrazione. Intuitivamente, la lunghezza della sua forma normale (e quindi il suo valore numerico) è limitata da una torre di esponenziali *di altezza fissata.* □

Osservazione 2. Le funzioni che caratterizzano uguaglianza, ordinamento e, quindi, sottrazione non sono rappresentabili con nessuna rappresentazione di interi su più domini (dimostrato da Statman come citato in [3]).

Pertanto le *funzioni rappresentabili in $\lambda_\to$ usando rappresentazioni di interi su domini diversi* sono un **sottoinsieme proprio** delle *funzioni elementari.*

Osservazione 3. Il Teorema 1 (Statman) fornisce un *lower bound* alla complessità del problema dell'equivalenza tra due termini. Il Teorema 6 ci dice che, con la strategia di riduzione qui utilizzata, tale problema ha come *upper bound* la funzione $\lambda n. \exp_n(n)$, che è strettamente maggiore di ogni funzione elementare.

11. Oltre gli Interi

Una strada diversa per aumentare il potere espressivo può essere quello di rilassare il vincolo (2) sulle funzioni definibili.

Potremmo pensare di non porre vincoli sul tipo di input delle funzioni, e di richiedere semplicemente che l'output sia di tipo $\mathsf{Bool} := \nu \to \nu \to \nu$.

Sotto questa convenzione, il potere espressivo di $\lambda_{\to}$ aumenta ancora.

La dimostrazione di Mairson del teorema di Statman vista in precedenza (e che richiede proprio solo il vincolo sull'output booleano), e in particolare la simulazione di una MdT *non-elementare* può essere vista come un risultato di complessità: il lambda calcolo tipato semplice può esprimere *almeno* tutte le fuzioni elementari.

Come spiegato in [4], è difficile utilizzare le dimostrazioni viste come framework computazionali, per le costruzioni "esponenziali" che contengono.

Nell'articolo quindi si prova un'altra strada, mantenendo sempre il calcolo "puro", ma codificando strutture relazionali al prim'ordine ("database") invece dei numerali di Church.

Riferimenti bibliografici

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] Andrea Asperti and Harry G. Mairson. Parallel beta reduction is not elementary recursive. *Inf. Comput.*, 170(1):49–80, October 2001.

[3] Steven Fortune, Daniel Leivant, and Michael O'Donnell. The expressiveness of simple and second-order type structures. *J. ACM*, 30(1):151–185, January 1983.

[4] Gerd G. Hillebrand, Paris C. Kanellakis, and Harry G. Mairson. Database query languages embedded in the typed lambda calculus. Technical report, 1995.

[5] Harry G. Mairson. A simple proof of a theorem of statman. *Theoretical Computer Science*, 103:387–394, 1992.

[6] S. Martini and M. Gabbrielli. *Linguaggi di programmazione: principi e paradigmi*. McGraw-Hill Italia, 2006.

[7] A.R Meyer. The inherent computational complexity of theories of ordered sets. In *Proc. Int'l. Cong. of Mathematicians*, volume 2, pages 477–482, 1974.

[8] H.E. Rose. *Subrecursion: functions and hierarchies*. Oxford logic guides. Clarendon Press, 1984.

[9] H. Schwichtenberg. Definierbare funktionen im lambda-kalkul mit typen. *Archiv Logik Grundlagenforsch*, 17:113–114, 1976.

[10] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics)*. Elsevier Science Inc., New York, NY, USA, 2006.

[11] Richard Statman. The typed lambda-calculus is not elementary recursive. *Theor. Comput. Sci.*, 9:73–81, 1979.